Consumer-view of consistency properties: definition, measurement, and exploitation

Alan Fekete
(University of Sydney)
Alan.Fekete@sydney.edu.au

PaPoC, London, April 2016

Limitations of this talk

- Discuss some of the variety of approaches
  - Probably too much focus on work I was involved in
  - I have not followed closely enough all the work in different communities
    - Apology in advance to anyone whose work was neglected (please let me know)
  - On several topics, other people here are the real experts, so ask them!

Introduction

- Distributed fault-tolerant storage systems
- Typically replicate data to survive node failures and partitions; perhaps also for better query response
- Often with “(perhaps sloppy) quorums”:
    - read sent to all, returns once it heard from R replicas, result is most up-to-date among them
    - write sent to all, returns once it heard from W replicas
        - updates re-propagated in background to deal with failures

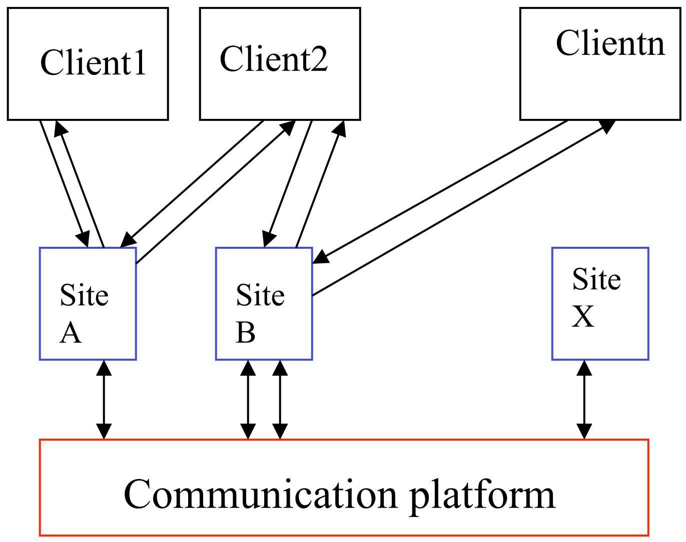
Road Map

- Definitions
  - Single item properties for a key-value store
  - Cross-item properties for a key-value store
  - Variations on the interface
  - Complex data types
- Measurement of consistency
- Exploitation of weak consistency
- Transactions

Acknowledgements: collaborators at Sydney, Berkeley; participants at Dagstuhl and Monte Verita workshops, especially Marcos Aguilera

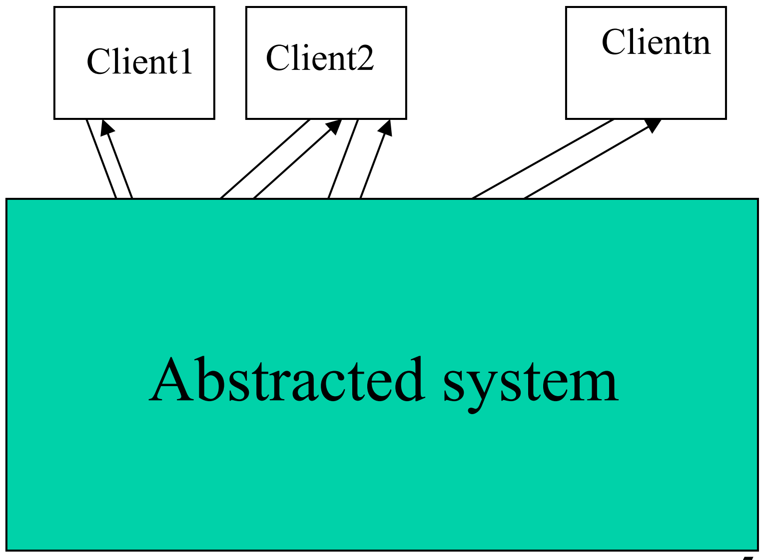
Defining a consistency property

- The system: clients request operations and get results



Sometimes, each client is bound to one site

- Abstract internals
  - Property of allowed sequence of events at the boundary



How to define acceptable executions?

- Some decidable test to apply to the sequence?
- Or “transparency” approach: there exists some execution of an idealized system, that looks the same (to each client separately, or to all clients at once)
  - eg Ideal system does not have replication!

Key-Value Stores

- put(key,value)
- v=get(key,value)

- Each operation extends from request till response
- Often, each request comes from a particular session/client, and on a single session there is no overlap between operations

Single item semantics

- Consider the requests/responses for a given key
- Strongest property: Atomic/linearizable
  - Each operation seems to act at an instant between request and response
  - Testing is complicated, but made easier with distinct values in puts, and non overlapping puts
    - If there are no overlapping ops, then a get will return the value from the latest put

CAP Theorem

- You cannot provide linearizable consistency in a system that may partition, unless you allow some requests to be rejected
- Proved by Gilbert, Lynch (SIGACT News 2002)

Coordination

- The real issue affecting user SLAs is whether the system coordinates across nodes
  - With partition, coordination reduces availability
  - Even without partition, coordination increases latency
- Abadi’s PACELC (IEEE Computer 2012)

Eventual consistency

- Each get returns a value from some previous put
- If puts cease to occur, eventually all gets will return the same value
  - Counterfactual!
  - Usually, if we wait long enough without puts, then all gets return the same v
- Advocated by Vogels (CACM'09) for cases where writes must be always available, and activity on separate keys is mostly independent

Eventual consistency variants

- Several possible implementations have been proposed
  - Some are consistent except during failures
  - Others are consistent only after a certain time has elapsed

Session semantics

- Extra properties such as
  - Monotonic reads
    - If one get sees v, then subsequent gets in the session also see v (or later changes)
  - Read-your-writes
    - If session has put v, subsequent gets return v (or later changes)

Cross-item properties

- What, if any, constraints relate operations on different items?
  - No relationship
  - Causal consistency
    - Avoids many anomalies that can trip up users
    - Eg change value of “my job” then change value of “my satisfaction”
    - Eg read value of “your relationship” then post “comment”

Causal Consistency

- Define causal (Lamport) precedes order between operations
  - One operation follows previous ops in the same session
  - Get follows the put whose value it returns
  - and *Transitive Closure*
- Every get must return a value at least as recent as any precedent put on that key

API Variations

- Multi-put, Multi-get
- Abstract Data Type
  - Operations other than get/put
- Collections
  - Predicate-based access

Data types

- More sophisticated operations, not just put/get
  - Eg key/value with put-if (test-and-set)
  - Eg counter with increment/decrement
  - Eg set with insert/delete/contains
  - Eg document with set-attribute, update-attribute
- Detail such as return-status is vital (decrement-if-still-positive is quite different from decrement-always)

Modify

- Operations may modify the state based on current state, or return differently based on current state
- Thus they act as both read and write
  - Unlike pure key/value, for which put is obliterating
- User rule for resolving conflicting modification?
  - Or use last-writer-wins?

Behavior of updates

- Eventual consistency:
  - Each observation returns a value from some collection of previous modifications
  - If modification ceases to occur, eventually all observations will return the same value
  - Question: is this eventual value showing the effect of *all* modifications?

Hybrid systems

- Different applications, even different activities in one application, may want different consistency choices
- Many systems allow users to choose strong or weak semantics per operation
  - Eg parameter to set quorum size (R or W)
  - Strong operations may be unavailable/slow, but weak ones should be available/faster
- Pileus (Terry et al, SOSP’13) offers a wide range of consistency options

Quantified consistency

- Γ-consistency (Golab et al, ICDCS'14):
  - expand each operation by Γ time units (shift start earlier by Γ/2, shift end later by Γ/2);
  - expanded-operations sequence is then linearizable
- Related definitions can be given for staleness measured by operation counts, etc

Road Map

- Definitions
- Measurement of consistency
  - Benchmarking
  - Trace analysis
  - Protocol analysis
- Exploitation of weak consistency
- Transactions

Benchmarking

- Apply carefully chosen workload; capture measurements during run
  - Consumer-view: can be done without knowledge or control of platform, without depending on vendor-provided monitoring reports
  - Can be designed to stress the platform (eg read very shortly after write) but can also be configured to avoid platform restrictions (eg rate-limits)

Example Benchmarking

- Wada et al, CIDR’11

- A writer updates current time into item, once each 3 secs, for 5 mins
- A reader(s) reads it 100 times/sec
  - Check if the data is stale by comparing value seen to the most recent value written (known from clock)
  - Plot probability of seeing stale value, against time of read since most recent write occurred
  - Reader location can be varied (same thread, same VM, etc)

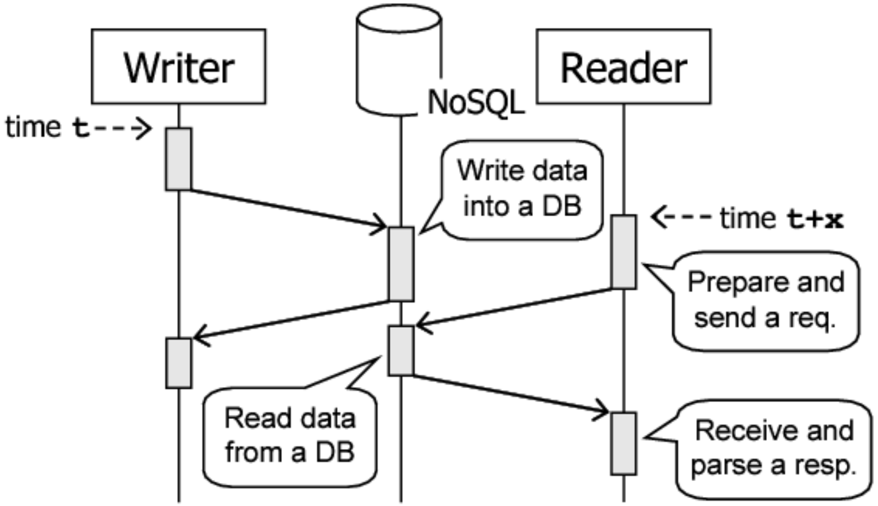
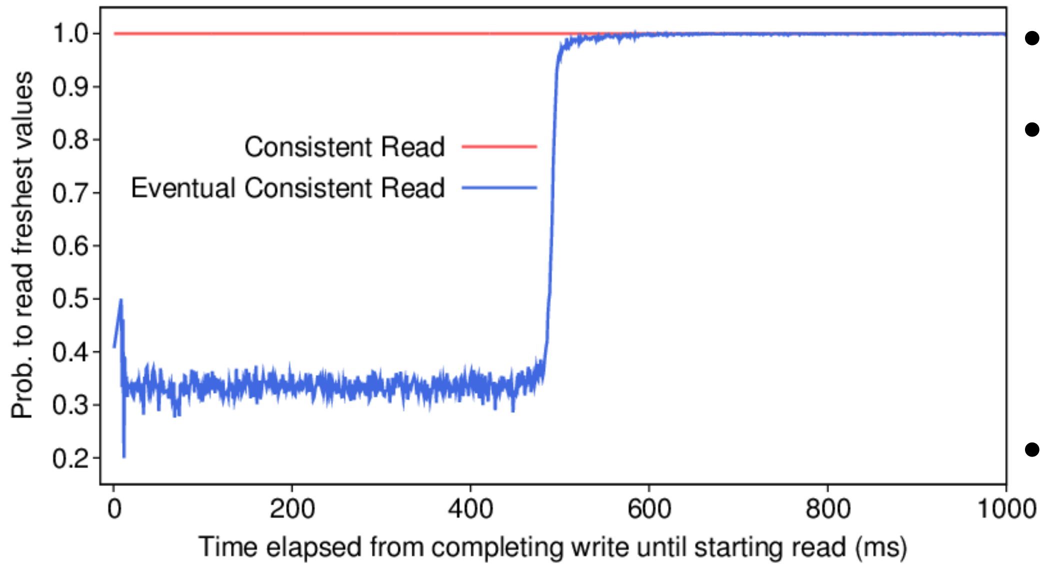
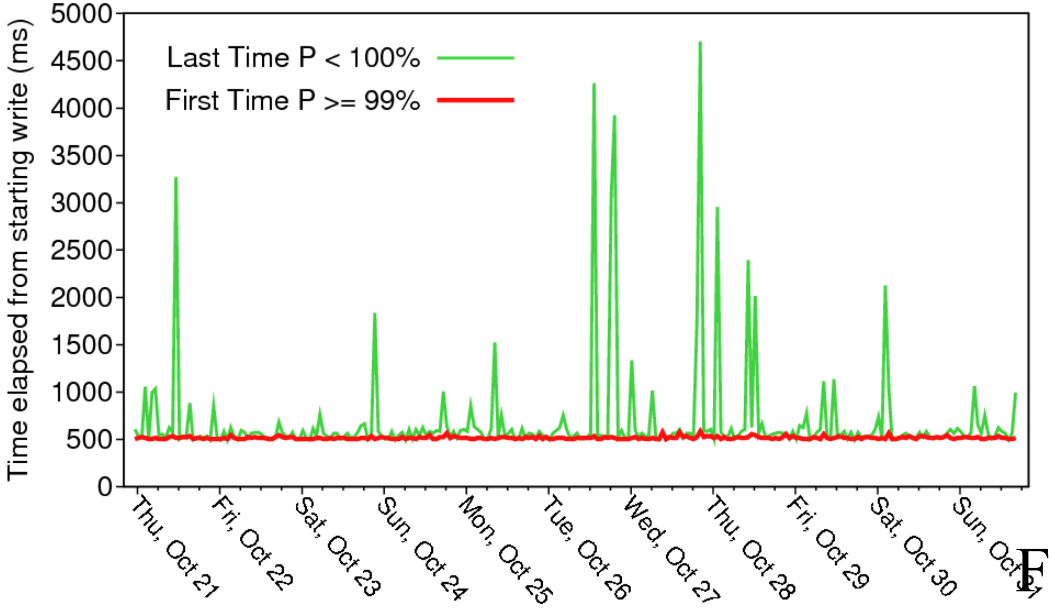


Figure from Wada et al CIDR’11

Benchmarking SimpleDB (in 2010)



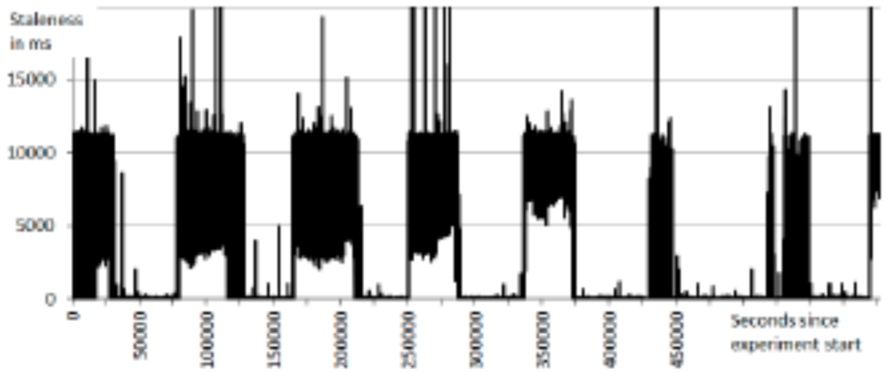


- Reader and writer in one thread
- With eventual consistent read, 33% of chance to read freshest values within 500ms
  - same when readers elsewhere
- Perhaps one master and two other replicas. Read takes value randomly from one of these?
- First time for eventual consistent read to reach 99% “fresh” is stable 500ms
- Outlier cases of stale read after 500ms, but no regular daily or weekly variation observed

Figures from Wada et al CIDR’11

Another benchmark

- Bermbach and Tai, IC2E'14
  - Measured on AWS S3
  - Multiple readers
  - Focus on maximum staleness at different times, as well as probability of staleness against time since write
  - Repeated over several years; saw considerable changes in behavior (eg periods with greater staleness)



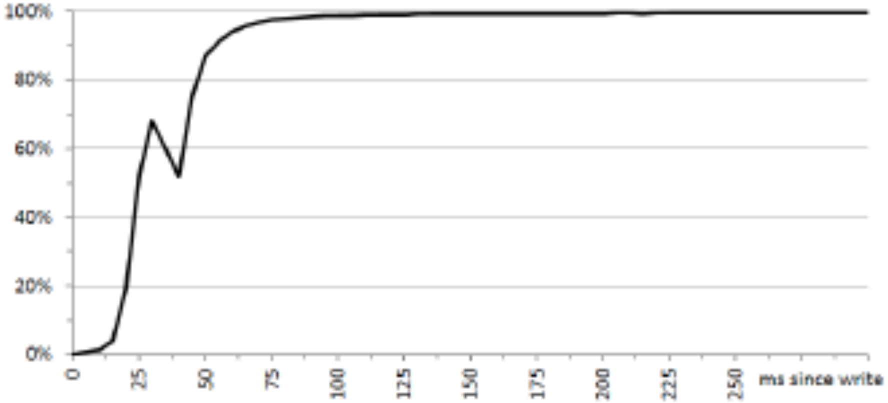


Fig. 9: Probability of Reading Fresh Data as a Function of the Time since the Last Update (Experiment 8)

Figures from Bermbach and Tai, IC2E'14

Trace analysis

- Observe trace of any workload; afterwards check whether it meets definition

- Can be used for genuine workloads, and thus measure how much actual impact from inconsistency
- Can explore how amount of inconsistency varies with properties of the workload
- Analysis of consistency can be expensive if workload doesn't have distinct values in writes

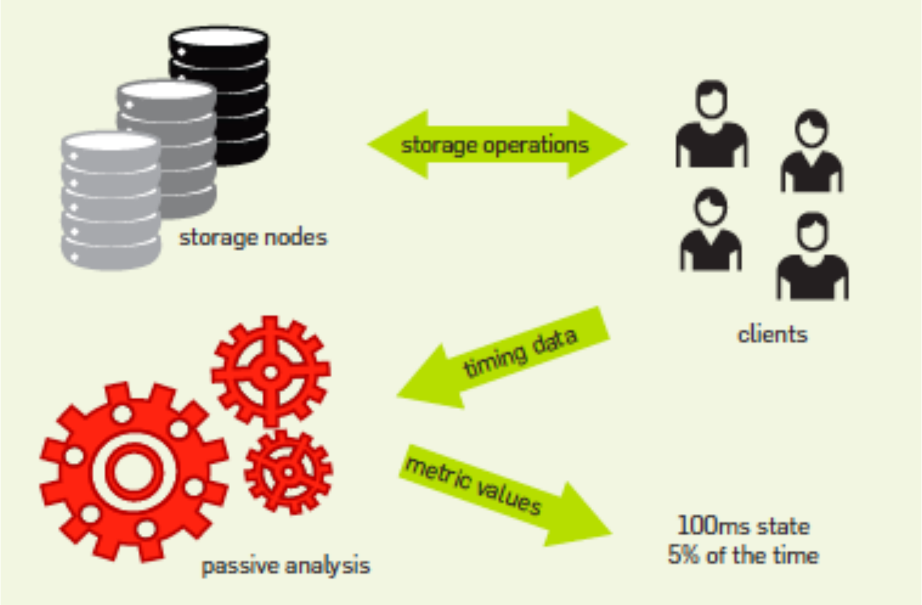
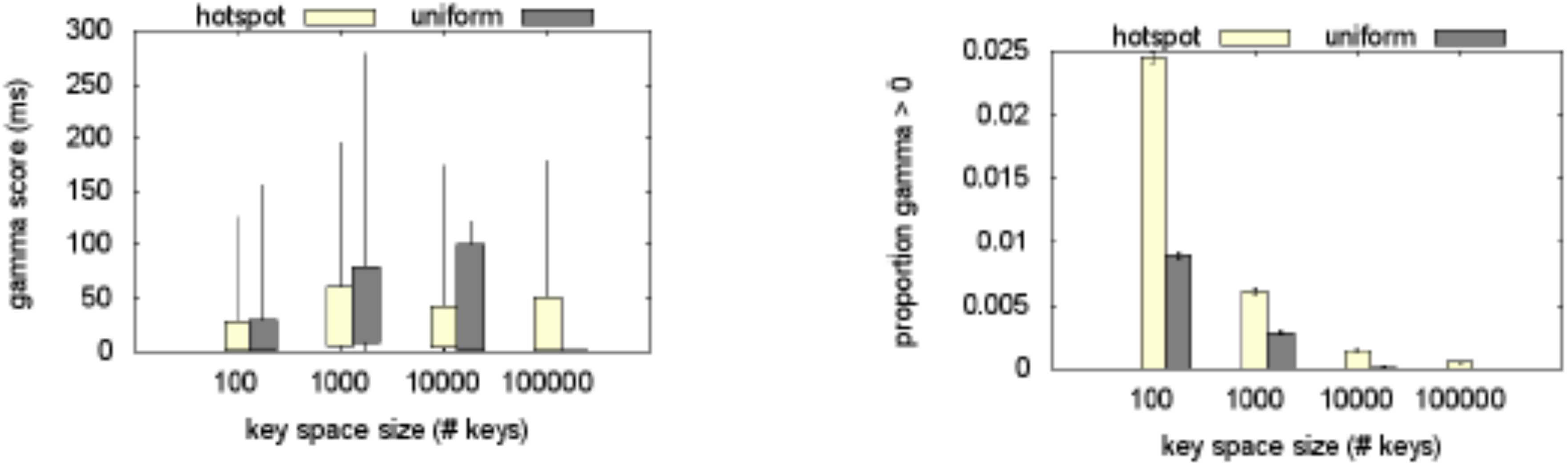


Figure from Golab et al CACM'11

Example Trace analysis

- Golab et al ICDCS’14
  - YCSB workloads against Cassandra
  - offline calculation of Γ-consistency (a separate score for each read op)



Figures from Golab et al ICDCS’14

Use of Trace analysis

- Fan et al, VLDB’15
  - YCSB-based workload on Cassandra
  - Collect logs, analyse for Γ-consistency

- plot Γ against time during run
- spikes in Γ matched to time and duration of stop-the-world garbage collection in JVM
- then modify Cassandra to detect GC and then delay reads
  - reduces spikes a lot, in exchange for increased latency (no effect on throughput)

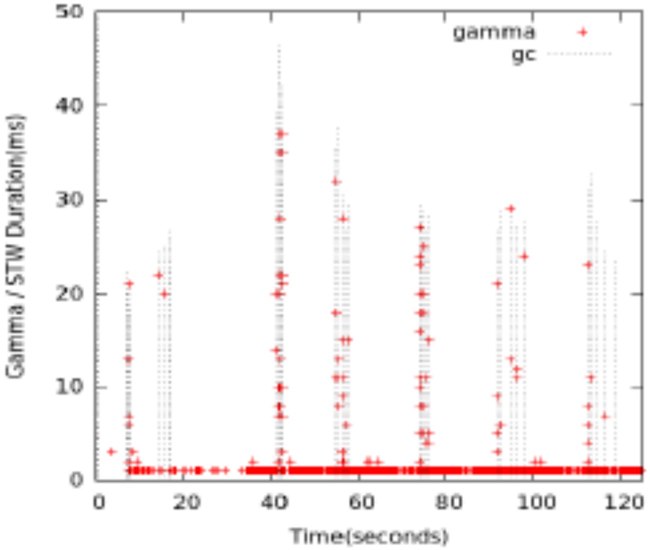
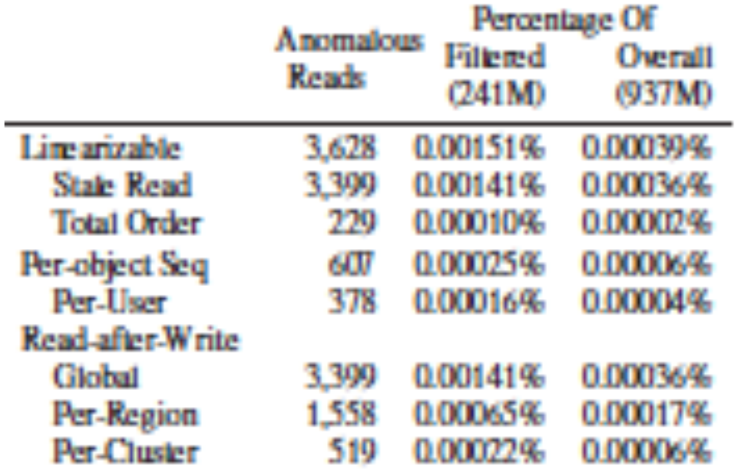


Figure from Fan et al VLDB’15

Another Trace analysis

- Real data from Facebook: Lu et al (SOSP'15)

- Log operations on a subset of items
- Check traces for several properties (linearizable, read-your-writes etc)
- Also, practical analysis done in real-time with injected reads (uses internal access to compare results from various replicas)



	Anomalous Reads	Percentage Of Filtered (241M)	Percentage Of Overall (937M)
Linearizable	3,628	0.00151%	0.00039%
Stale Read	3,399	0.00141%	0.00036%
Total Order	229	0.00010%	0.00002%
Per-object Seq	607	0.00025%	0.00006%
Per-User	378	0.00016%	0.00004%
Read-after-Write			
Global	3,399	0.00141%	0.00036%
Per-Region	1,558	0.00065%	0.00017%
Per-Cluster	519	0.00022%	0.00006%

Table from Lu et al SOSP'15

Call-me-maybe

- Jepsen, a testing framework from Kyle Kingsbury
  - focus on dealing with network partitions
  - https://aphyr.com/tags/Jepsen
- Clients each add an attribute, or insert into a set
  - While partitions etc are artificially induced
- After everything settles, look at the value, report on how many acknowledged inserts are present
  - Tool has been extended with trace analysis for linearizability
- Has found significant flaws in implementation of algorithms in deployed software
  - Also: shows the great danger of merge by latest timestamp

Protocol analysis

- Model protocol; analyse potential traces
  - Requires knowledge of protocol (perhaps not possible for cloud-vendors storage offerings)
  - Can be done with simulation, or sophisticated formal methods tools (that encompass timing)
  - Can permit sensitivity analysis and can guide system tuning

Example Protocol analysis

- PBS in Bailis et al VLDBJ'14

- Cassandra algorithm; operations sent to all replicas, and first R or W results used
- Model uses measured message latencies between each pair of sites, measured latencies at each site due to storage hardware, etc
- Failures are not considered
- Determines probability of seeing stale value in read, or probability of seeing older value than previous read, against delay from most recent write

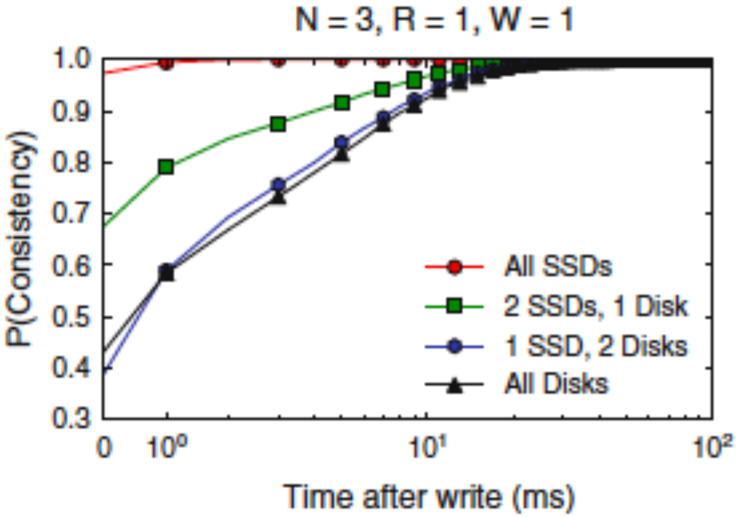


Figure from Bailis et al VLDBJ'14

Use of Protocol analysis

- From Rahman et al, arXiV:1509.02464v2, 2016
- Considers failures
- Used for system adaptation
  - predict performance and then change protocol details to ensure good SLAs
  - integrated in Cassandra and Riak

Road Map

- Definitions
- Measurement of consistency
- Exploitation of weak consistency
- Transactions

Proof techniques

- Show that some applications work correctly when run with weak isolation
- Perhaps: show executions could arise in stronger models
  - “robustness” result
- Or: show that certain invariants hold after execution

Proof rules

- One set is given in Gotsman et al POPL’16
- Requires at least causal consistency
- Commutativity of operations is important

Design guidance

- Lead programmers to produce applications that will work properly even though platform has weak consistency
- Variant: take given application, and determine where to strengthen the coordination

CRDTs

- Shapiro et al SSS’11 etc
- A very powerful paradigm for applications that can work with eventual consistency
- Application uses data types whose operations all commute
- Challenges lie in extracting information, and in composing types

Synchronisation-introduction

- Blazes proposed by Alvaro et al ICDE'14
- Starts with declarative programming approach
- Non-monotonicity is where eventual consistency is not good enough
  - so introduce coordination just there (perhaps barriers, punctuation on streams, etc)

Road Map

- Definitions
- Measurement of consistency
- Exploitation of weak consistency
- Transactions

Transactions

- Group multiple operations of a session into a fate-shared unit
  - Begin/Commit/Abort
- This can be very useful for application programmer
  - allow escape and rollback after problems detected
  - do complicated changes through a sequence of simpler (API-supported) steps

“Isolated”

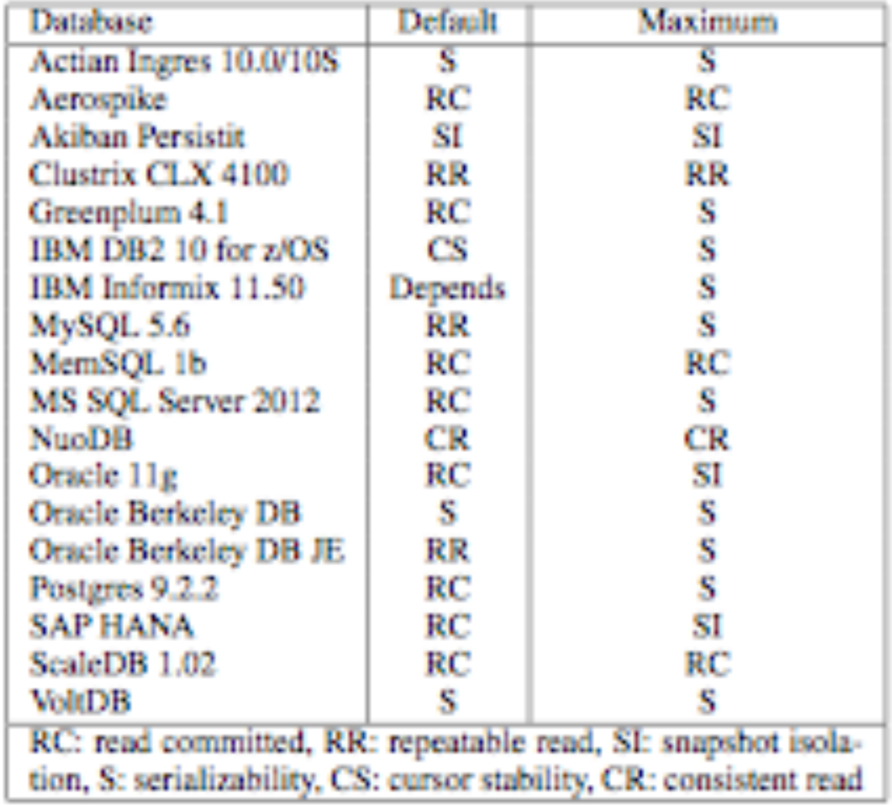
- Academic definition: Serializable
  - (ought to be default for systems, but not so in practice)
- Key property: interleaved execution is equivalent (same values returned, same final state of db) as some execution where transactions run serially (no interleaving at all)
- No dirty read, no lost update
- If each transaction (running alone) preserves some constraint I, then the whole execution preserves I
- Implemented: Traditionally done with Commit-duration locks on data and indices
  - “Two Phase Locking (2PL)”
  - Also newer multiversion implementations (eg Cahill et al, TODS’09)

Serializable or available?

- Serializability can’t be implemented in partitionable system, unless some transactions are sometimes blocked
  - Davidson et al (ACM Computing Surveys 1985)

ACID Transactions with weaker I

- Serializability is the ideal for isolation of transactions but most transactions on (conventional, single site) dbms don't run serializably!
  - Read Committed is often the default level



Database	Default	Maximum
Actian Ingres 10.0/10S	S	S
Aerospike	RC	RC
Akiban Persistit	SI	SI
Clustrix CLX 4100	RR	RR
Greenplum 4.1	RC	S
IBM DB2 10 for z/OS	CS	S
IBM Informix 11.50	Depends	S
MySQL 5.6	RR	S
MemSQL 1b	RC	RC
MS SQL Server 2012	RC	S
NuoDB	CR	CR
Oracle 11g	RC	SI
Oracle Berkeley DB	S	S
Oracle Berkeley DB JE	RR	S
Postgres 9.2.2	RC	S
SAP HANA	RC	SI
ScaleDB 1.02	RC	RC
VoltDB	S	S

RC: read committed, RR: repeatable read, SI: snapshot isolation, S: serializability, CS: cursor stability, CR: consistent read

Figure from Bailis et al HotOS'13

Snapshot isolation

- Proposed by Berenson et al (SIGMOD’95)
- There is a total order on transaction start and complete events
  - All operations of T show effects of all txns that committed before T started, and its own previous ops, but of no other operations
    - No visibility of concurrent transactions
- Variants: PSI (Sovran, SOSP’11) allows different orders to be seen at other sites

Weaker Isolation Levels

- SQL standard offers several isolation levels
- Each transaction can have level set separately
- Read Uncommitted
  - Usually only for read-only code
  - Implemented: no read locks, commit-duration write locks
- Read Committed
  - No dirty reads (can’t see uncommitted, aborted or intermediate values)
  - Implemented: short duration read locks, commit-duration write locks
  - MV implementation: can return older version, while concurrent update is happening
- Repeatable Read
  - No “phantoms” (predicate evaluation that sees versions inserted concurrently)
  - Implemented: Commit-duration locks on data
    - Should be the same as Serializable for a key-value store
  - Some multiversion systems provide “snapshot reading” for this level

Isolation levels for HAT

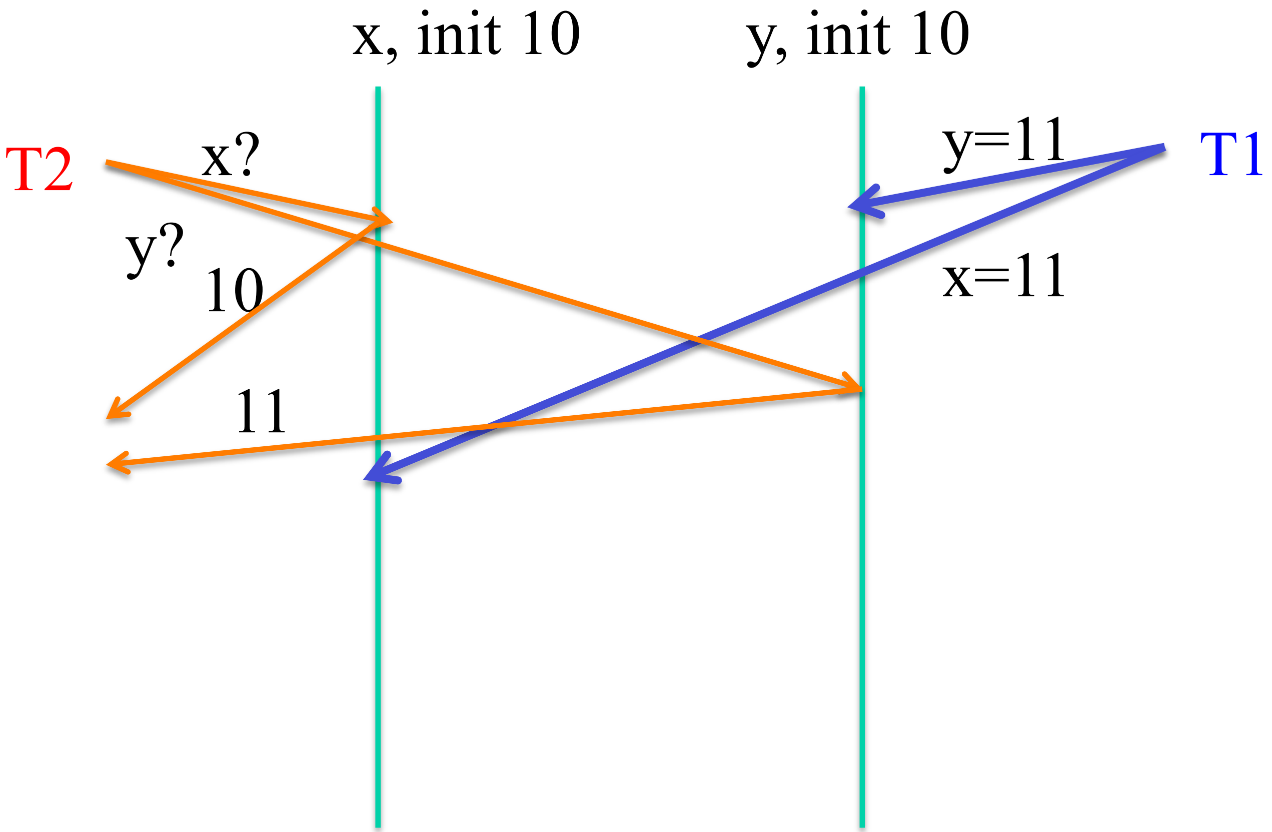
- Bailis et al (VLDB14) shows that you can offer available transactions that have
  - All-or-nothing atomicity
  - Isolation level like (the definitions of) read committed and repeatable read*
    - But where reads may not always see the most recent committed changes
    - And you don’t get all the extra properties of conventional locking implementation (eg timeline view)
  - Causal consistency (including RYW, monotonic reads, write follows reads) {as long as client is sticky to a partition}

*in absence of predicate reads [which is not an issue for key-value store]

Read Atomic

- A new propsed isolation level (Bailis et al, SIGMOD'14)
- Read committed, PLUS "No fractured reads"
  - Avoid the following:
    - T1 writes x, y
    - T2 reads x (seeing T1 or later), y (not seeing T1)
- RA does not always guarantee transaction consistent snapshot: Transitive information flow may be fractured

Anomaly Prevented by RA



time
increasing
down the page

Caveat

- RA does not always guarantee transaction consistent snapshot
  - Transitive information flow may be fractured
  - However, many common coding idioms are supported effectively
    - Eg maintain both ends of bidirectional associations consistently
      - Contrast with Facebook TAO, LinkedIn Espresso etc
    - Eg maintain secondary index consistent with data
    - Eg maintain referential integrity

Exploitation of weak txn models

- Theory to check whether set of txns acts serializably on SI (Fekete et al, TODS'05)
- Ways to re-code cases so theory does apply
  - without changing application meaning (just eliminating anomalous concurrent executions)

Exploitation of weak txn models

- Theory to check whether set of txns preserves invariants under RA (Bailis, Berkeley PhD thesis 2015)
- Many common coding idioms are supported
  - Eg maintain both ends of bidirectional associations consistently
  - Eg maintain secondary index consistent with data
  - Eg maintain referential integrity

Call to arms?

- For most systems, we aren’t told clearly what they do
  - Need SLAs including the quantitative aspects (eg how stale, when stale)
- For most weak consistency definitions, we don’t know enough about how to use it properly
  - Need “design patterns” justified by proof rules