Towards Highly Available Transactions

Peter Bailis
Alan Fekete
Ali Ghodsi
Joe Hellerstein
Ion Stoica

UC Berkeley,
U of Sydney
HAT
not
CAP
HAT
not
CAP
HAT
not
CAP
HAT
not CAP
July 2000: CAP Theorem
CONSISTENCY vs AVAILABILITY
CONSISTENCY vs AVAILABILITY

Linearizability
“Atomic”
C in CAP
CONSISTENCY vs AVAILABILITY

Linearizability
“Atomic”
C in CAP
CONSISTENCY VS AVAILABILITY

Causal

Linearizability
"Atomic"
C in CAP
CONSISTENCY vs AVAILABILITY

Eventual
Causal

Linearizability
"Atomic"
C in CAP
CONSISTENCY vs AVAILABILITY

Eventual Causal

Linearizability “Atomic” C in CAP
High Availability

System guarantees a response, even during network partitions (async network)

[Gilbert and Lynch, ACM SIGACT News 2002]
High Availability

System guarantees a response, even during network partitions (async network)

[Gilbert and Lynch, ACM SIGACT News 2002]
High Availability

System guarantees a response, even during network partitions (async network)

[Gilbert and Lynch, ACM SIGACT News 2002]
High Availability

System guarantees a response, even during network partitions (async network)

[Gilbert and Lynch, ACM SIGACT News 2002]
High Availability

System guarantees a response, even during network partitions (async network)

[Gilbert and Lynch, ACM SIGACT News 2002]
High Availability

System guarantees a response, even during network partitions (async network)

[Gilbert and Lynch, ACM SIGACT News 2002]
High Availability

System guarantees a response, even during network partitions (async network)

[Gilbert and Lynch, ACM SIGACT News 2002]
High Availability

System guarantees a response, even during network partitions (async network)

[Gilbert and Lynch, ACM SIGACT News 2002]
**High Availability**

System guarantees a response, even during network partitions (async network)

[Gilbert and Lynch, ACM SIGACT News 2002]

Corollary: low latency, especially over WAN

[“PACELC,” Abadi, IEEE Computer 2012]
NoSQL
Strong consistency is expensive; avoid whenever possible!
NoSQL

Strong consistency is expensive; avoid whenever possible!

Common misconception:
CAP implies transactions are unavailable
Brewer’s Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services

Seth Gilbert*  
Nancy Lynch*

Abstract

When designing distributed web services, there are three properties that are commonly desired: consistency, availability, and partition tolerance. It is impossible to achieve all three. In this note, we prove this conjecture in the asynchronous network model, and then discuss solutions to this dilemma in the partially synchronous model.

1 Introduction

At PODC 2000, Brewer¹, in an invited talk [2], made the following conjecture: it is impossible for a web service to provide the following three guarantees:

- Consistency
- Availability
- Partition tolerance

* indicates corresponding author
Brewer’s Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services

Seth Gilbert*       Nancy Lynch*

CAP doesn’t mention transactions

CAP is about linearizability
Are transactions achievable with high availability?
Eventual Causal Linearizability "Atomic" C in CAP
TRANSACTIONS

Eventual  Causal  Linearizability “Atomic” C in CAP
serializability
serializability
transactions behave as if executed serially against a single database
**serializability**

transactions behave as if executed serially against a single database

multiple object guarantee

no real time ordering
serializability
transactions behave as if executed serially against a single database
multiple object guarantee
no real time ordering

contrast with
linearizability
single object guarantee
strict real time ordering
serializability
transactions behave as if executed serially against a single database
Is serializability HA?

transactions behave as if executed serially against a single database
Is serializability HA?

transactions behave as if executed serially against a single database
Is serializability HA?

transactions behave as if executed serially against a single database
Is serializability HA?

transactions behave as if executed serially against a single database

T1: W(x=1)
R(y=?)
Is serializability HA?

transactions behave as if executed serially against a single database

T1: W(x=1)
R(y=?)

T2: W(y=1)
R(x=?)
Is *serializability* HA?

Transactions behave as if executed serially against a single database.

\[ T_1: W(x=1) \]
\[ R(y=?) \]

\[ T_2: W(y=1) \]
\[ R(x=?) \]
Is serializability HA?

Transactions behave as if executed serially against a single database

T1: W(x=1)
R(y=null)

T2: W(y=1)
R(x=?)

Is HA?
Is serializability HA?

transactions behave as if executed serially against a single database

T1: W(x=1)  
    R(y=null)

T2: W(y=1)  
    R(x=null)
Is **serializability** HA?

transactions behave as if executed serially against a single database

T1: W(x=1)  
R(y=null)

T2: W(y=1)  
R(x=null)
Was the NoSQL movement right?
Was the NoSQL movement right?

Are all transactions unavailable?
do not support serializability
8/18 databases surveyed did not support serializability.

15/18 used weak models by default.
Granularity of Locks and Degrees of Consistency in a Shared Data Base

J. N. Gray
R. A. Lorie
G. R. Putzolu
I. L. Traiger

IBM Research Laboratory
San Jose, California

ABSTRACT: In the first part of the paper the problem of choosing the granularity (size) of lockable objects is introduced and the related tradeoff between concurrency and overhead is discussed. A locking protocol which allows simultaneous locking at various granularities by different transactions is presented. It is based on the introduction of additional lock modes besides the conventional share mode and exclusive mode. A proof is given of the equivalence of this protocol to a conventional one.

In the second part of the paper the issue of consistency in a shared environment is analyzed. This discussion is motivated by the realization that some existing data base systems use automatic lock protocols which insure protection only from certain types of inconsistencies (for instance those arising from transaction backup), thereby automatically providing a limited degree of consistency. Four degrees of consistency are introduced. They can be roughly characterized as follows: degree 0 protects others from your updates, degree 1 additionally provides protection from losing updates, degree 2 additionally provides protection from
Serializability is expensive

Use weaker models instead
serializability
serializability
serializability
snapshot isolation
read committed
update
serializability
cursor stability
monotonic view
read uncommitted
repeatable read
read
serializability

snapshot isolation

read committed

update

serializability
cursor stability

monotonic view

read uncommitted

repeatable read
Highly Available Transactions

HA?

HA?

HA?

HA?
serializability
snapshot isolation
read committed
repeatable read
cursor stability
read uncommitted
monotonic view
HATs
HA?
HA?
HA?
HA?
Read Committed (RC)
Read Committed (RC)

ANSI Repeatable Read (RR)
Read Committed (RC)
Don’t read aborted transactions’ writes
Don’t read transactions’ non-final writes
[Adya PL-1, Berenson P1/A1, ANSI SQL Spec P1]

Default in 8/18 databases

ANSI Repeatable Read (RR)
Read Committed (RC)
Don’t read aborted transactions’ writes
Don’t read transactions’ non-final writes
[Adya PL-1, Berenson P1/A1, ANSI SQL Spec P1]

Default in 8/18 databases

ANSI Repeatable Read (RR)
Each transaction reads from a non-changing snapshot of the database
[ANSI SQL Spec P2]

Default in 3/18 databases
Read Committed (RC)
Read Committed (RC)

Transactions buffer writes until commit time

Replicas never serve dirty or non-final writes
Read Committed (RC)
Transactions buffer writes until commit time
Replicas never serve dirty or non-final writes
Read Committed (RC)
Transactions buffer writes until commit time
Replicas never serve dirty or non-final writes

ANSI Repeatable Read (RR)
Read Committed (RC)
Transactions buffer writes until commit time
Replicas never serve dirty or non-final writes

ANSI Repeatable Read (RR)
Transactions buffer reads from replicas
Transactions read from a snapshot of DB
Read Committed (RC)
Transactions buffer writes until commit time
Replicas never serve dirty or non-final writes

ANSI Repeatable Read (RR)
Transactions buffer reads from replicas
Transactions read from a snapshot of DB
Read Committed (RC)
Transactions buffer writes until commit time
Replicas never serve dirty or non-final writes

ANSI Repeatable Read (RR)
Transactions buffer reads from replicas
Transactions read from a snapshot of DB

Unavailable implementations
⇏ unavailable semantics
Impossible in HATs
Impossible in HATs

Cannot make recency guarantees on reads during partitions of unbounded duration

[corollary to CAP Theorem]
Impossible in HATs

Cannot make recency guarantees on reads during partitions of unbounded duration
[corollary to CAP Theorem]

Cannot enforce arbitrary database integrity constraints (e.g., uniqueness)
[corollary to serializability result]
Highly Available Transactions

many useful properties from databases like Read Committed and Repeatable Read achievable with HA during partitions
prevents lost update†, prevents write skew‡, requires recency guarantees⊕
Highly Available Transactions

many useful properties from databases like Read Committed and Repeatable Read achievable with HA during partitions

despite this paper is just a start...
Highly Available Transactions

many useful properties from databases like Read Committed and Repeatable Read achievable with HA during partitions

during partitions

this paper is just a start...

what about the strongest HAT semantics?

most useful HAT semantics?
Related Work

Spanner [OSDI 2012]
Serializable, fast read-only transactions, 2PL over WAN

Walter [SOSP 2011]
Parallel Snapshot Isolation with commutative updates

Gemini [OSDI 2012]
Mixing of strong and weak consistency for single items

Decades of DB Literature dating to 1970s
  e.g, Adya Ph.D. Thesis [MIT 2000], Critique of ANSI SQL [SIGMOD 1995], Gray Isolation [1976]

Eiger [NSDI 2013]
Low latency read-only, write-only transactions with causality

Swift [Under submission, INRIA]
Low latency, HA read-write transactions with causality

Bayou [PDIS 1994, SOSP 1995]
Early support for read-only, write-only txns, dep. checks
Related Work

Spanner [OSDI 2012]
Serializable, fast read-only transactions, 2PL over WAN

Walter [SOSP 2011]
Parallel Snapshot Isolation with commutative updates

Gemini [OSDI 2012]
Mixing of strong and weak consistency for single items

Decades of DB Literature dating to 1970s
e.g, Adya Ph.D. Thesis [MIT 2000], Critique of ANSI SQL [SIGMOD 1995], Gray Isolation [1976]

Eiger [NSDI 2013]
Low latency read-only, write-only transactions with causality

Swift [Under submission, INRIA]
Low latency, HA read-write transactions with causality

Bayou [PDIS 1994, SOSP 1995]
Early support for read-only, write-only txns, dep. checks
## Related Work

<table>
<thead>
<tr>
<th>System</th>
<th>Reference</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spanner</td>
<td>[OSDI 2012]</td>
<td>Serializable, fast read-only transactions, 2PL over WAN</td>
</tr>
<tr>
<td>Walter</td>
<td>[SOSP 2011]</td>
<td>Parallel Snapshot Isolation with commutative updates</td>
</tr>
<tr>
<td>Gemini</td>
<td>[OSDI 2012]</td>
<td>Mixing of strong and weak consistency for single items</td>
</tr>
<tr>
<td>Swift</td>
<td>[Under submission, INRIA]</td>
<td>Low latency, HA read-write transactions with causality</td>
</tr>
<tr>
<td>Bayou</td>
<td>[PDIS 1994, SOSP 1995]</td>
<td>Early support for read-only, write-only txns, dep. checks</td>
</tr>
</tbody>
</table>

*HA*: High Availability

Decades of DB Literature dating to 1970s
e.g, Adya Ph.D. Thesis [MIT 2000], Critique of ANSI SQL [SIGMOD 1995], Gray Isolation [1976]

Systems projects; our focus has been on semantics provided by existing databases