Replicated Data Consistency Explained Through Baseball

slides by
Landon Cox
with some others from elsewhere
prepended and appended
(cultural history lesson)
• K-V stores are a common data tier for mega-services.
• They evolved during the Web era, starting with DDS.
• Today they often feature geographic replication.
  – Multiple replicas in different data centers.
  – Geo-replication offers better scale and reliability/availability.
• But updates are slower to propagate, and network partitions may interfere.
  – A read might not see the “latest” write.
• So we have to think carefully about what consistency properties we need: “BASE” might be “good enough”.
  – FLP and CAP tell us that there are fundamental limits on what we can guarantee….but many recent innovations in this space.
Key-value stores

- Many mega-services are built on key-value stores.
  - Store variable-length content objects: think “tiny files” (value)
  - Each object is named by a “key”, usually fixed-size.
  - Key is also called a token: not to be confused with a crypto key! Although it may be a content hash (SHAx or MD5).
  - Simple put/get interface with no offsets or transactions (yet).
  - Goes back to literature on Distributed Data Structures [Gribble 2000] and Distributed Hash Tables (DHTs).

[Image from Sean Rhea, opendht.org, 2004]
ACID vs. BASE

Jim Gray
ACM Turing Award 1998

Eric Brewer
ACM SIGOPS Mark Weiser Award 2009
ACID vs. BASE

**ACID**
- Strong consistency
- Isolation
- Focus on “commit”
- Nested transactions
- Availability?
- Conservative (pessimistic)
- Difficult evolution (e.g. schema)
- “small” Invariant Boundary
- The “inside”

**BASE**
- Weak consistency
  - Stale data OK
- Availability first
- Best effort
- Approximate answers OK
- Aggressive (optimistic)
- “Simpler” and faster
- Easier evolution (XML)
- “wide” Invariant Boundary
- Outside consistency boundary

but it’s a *spectrum*

HPTS Keynote, October 2001
Dr. Werner Vogels is Vice President & Chief Technology Officer at Amazon.com.

Prior to joining Amazon, he worked as a researcher at Cornell University.

Building reliable distributed systems at a worldwide scale demands trade-offs between consistency and availability.

BY WERNER VOELGES
Vogels on consistency

The scenario: A updates a “data object” in a “storage system”.

Consistency “has to do with how observers see these updates”.

Strong consistency: “After the update completes, any subsequent access will return the updated value.”

Eventual consistency: “If no new updates are made to the object, eventually all accesses will return the last updated value.”
PNUTS: Yahoo!’s Hosted Data Serving Platform

Brian F. Cooper, Raghu Ramakrishnan, Utkarsh Srivastava, Adam Silberstein, Philip Bohannon, Hans-Arno Jacobsen, Nick Puz, Daniel Weaver and Ramana Yerneni

Yahoo! Research
Example: social network updates

What are my friends up to?

Sonja:

Brandon:
Example: social network updates

![Image of Flower](www.flickr.com)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jimi</td>
<td>ph...</td>
</tr>
<tr>
<td>Mary</td>
<td>re...</td>
</tr>
<tr>
<td>Sonja</td>
<td>ph...</td>
</tr>
<tr>
<td>Brandon</td>
<td>po...</td>
</tr>
<tr>
<td>Mike</td>
<td>ph...</td>
</tr>
<tr>
<td>Bob</td>
<td>re...</td>
</tr>
</tbody>
</table>
Asynchronous replication
Consistency model

- Goal: make it easier for applications to reason about updates and cope with asynchrony

- What happens to a record with primary key “Brian”? 
Consistency model

- Current version
- Stale version
- Stale version
- Read

Time

v. 1 to v. 8

Generation 1
Consistency model

- Time
  - v. 1
  - v. 2
  - v. 3
  - v. 4
  - v. 5
  - v. 6
  - v. 7
  - v. 8

- Read up-to-date
- Stale version
- Stale version
- Current version

Generation 1
Consistency model

Read-critical (required version):

- Read ≥ v.6
- Stale version
- Stale version
- Current version

Time
Consistency model

Test-and-set-write(required version)

Write if = v.7

ERROR

Stale version

Stale version

Current version

Generation 1

v. 1 v. 2 v. 3 v. 4 v. 5 v. 6 v. 7 v. 8
Don’t Settle for Eventual: Scalable Causal Consistency for Wide-Area Storage with COPS

Wyatt Lloyd*
Michael J. Freedman*
Michael Kaminsky†
David G. Andersen‡

*Princeton, †Intel Labs, ‡CMU
Wide-Area Storage

Stores:
Status Updates
Likes
Comments
Photos
Friends List

Stores:
Tweets
Favorites
Following List

Stores:
Posts
+1s
Comments
Photos
Circles
Wide-Area Storage
Serves Requests Quickly
Inside the Datacenter

Web Tier

Storage Tier

A-F

G-L

M-R

S-Z

Remote DC

Replication
Desired Properties: ALPS

- **Availability**
- **Low Latency**
- **Partition Tolerance**
- **Scalability**

\[\text{“Always On”}\]
Scalability
Increase capacity and throughput in each datacenter
Desired Property: Consistency

- Restricts order/timing of operations

- Stronger consistency:
  - Makes programming easier
  - Makes user experience better
Consistency with ALPS

**Strong**
Impossible [Brewer00, GilbertLynch02]

**Sequential**
Impossible [LiptonSandberg88, AttiyaWelch94]

**Causal**
COPS

**Eventual**
Amazon Dynamo, LinkedIn Voldemort, Facebook/Apache Cassandra
<table>
<thead>
<tr>
<th>System</th>
<th>A</th>
<th>L</th>
<th>P</th>
<th>S</th>
<th>Consistency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scatter</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>Strong</td>
</tr>
<tr>
<td>Walter</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>?</td>
<td>PSI + Txn</td>
</tr>
<tr>
<td><strong>COPS</strong></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Causal+</td>
</tr>
<tr>
<td>Bayou</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>Causal+</td>
</tr>
<tr>
<td>PNUTS</td>
<td>✓</td>
<td>✓</td>
<td>?</td>
<td>✓</td>
<td>Per-Key Seq.</td>
</tr>
<tr>
<td>Dynamo</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Eventual</td>
</tr>
</tbody>
</table>
Replicated-data consistency

• A set of invariants on each read operation
  • Which writes are guaranteed to be reflected?
  • What write orders are guaranteed?

• Consistency is an application-level concern
  • When consistency is too weak, applications break
  • Example: auction site must not tell two people they won

• What are consequences of too-strong consistency?
  • Worse performance (for reads and writes)
  • Worse availability (for reads and writes)
• The following are slides on the Doug Terry paper by Landon Cox.
• We went through these pretty fast in class, but you should understand these models and why we might use them.
Assumptions for our discussion

1. Clients perform reads and writes
2. Data is replicated among a set of servers
3. Writes are serialized (logically, one writer)
   1. Performed in the same order at all servers
   2. Write order consistent with write-request order
4. Reads result of one or more past writes
Consistency models

1. **Strong consistency**
   - Reader sees effect of all prior writes

2. **Eventual consistency**
   - Reader sees effect of subset of prior writes

3. **Consistent prefix**
   - Reader sees effect of initial sequence of writes

4. **Bounded staleness**
   - Reader sees effect of all “old” writes

5. **Monotonic reads**
   - Reader sees effect of increasing subset of writes

6. **Read my writes**
   - Reader sees effect of all writes performed by reader
Setting: baseball game

Write ("visitors", 0);
Write ("home", 0);
for inning = 1..9
  outs = 0;
  while outs < 3
    visiting player bats;
    for each run scored
      score = Read ("visitors");
      Write ("visitors", score + 1);
    outs = 0;
  while outs < 3
    home player bats;
    for each run scored
      score = Read ("home");
      Write ("home", score + 1);
end game;

Primary game thread. Only thread that issues writes.
Visitors’ score

Visitor scores:
- S1: V
- S2: V
- S3: V

Home score

Home scores:
- S4: H
- S5: H
- S6: H

Reader (also reads)

- Reader to S1: R
- Reader to S2: R
- Reader to S3: R
- Reader to S4: W
- Reader to S5: W
- Reader to S6: W

Writer

- Writer to S1: R
- Writer to S2: R
- Writer to S3: R
- Writer to S4: W
- Writer to S5: W
- Writer to S6: W
Example 1: score keeper

```plaintext
score = Read ("visitors");
Write ("visitors", score + 1);
...
score = Read ("home");
Write ("home", score + 1);
```
Example 1: score keeper

What invariant is the score keeper maintaining on the game’s score?

Both values increase monotonically.

<table>
<thead>
<tr>
<th>Write</th>
<th>“home”, 1;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>“visitors”, 1;</td>
</tr>
<tr>
<td>Write</td>
<td>“home”, 2;</td>
</tr>
<tr>
<td>Write</td>
<td>“home”, 3;</td>
</tr>
<tr>
<td>Write</td>
<td>“visitors”, 2;</td>
</tr>
<tr>
<td>Write</td>
<td>“home”, 4;</td>
</tr>
<tr>
<td>Write</td>
<td>“home”, 5;</td>
</tr>
</tbody>
</table>

Visitors = 2
Home = 5
Example 1: score keeper

What invariant must the store provide so the score keeper can ensure monotonically increasing scores?

Write ("home", 1);
Write ("visitors", 1);
Write ("home", 2);
Write ("home", 3);
Write ("visitors", 2);
Write ("home", 4);
Write ("home", 5);

Visitors = 2
Home = 5

Reads must show effect of all prior writes (strong consistency)
Example 1: score keeper

Under strong consistency, what possible scores can the score keeper read after this write completes?

```
Write ("home", 1);
Write ("visitors", 1);
Write ("home", 2);
Write ("home", 3);
Write ("visitors", 2);
Write ("home", 4);
Write ("home", 5);
```

Visitors = 2
Home = 5

2-5
Example 1: score keeper

Write ("home", 1);
Write ("visitors", 1);
Write ("home", 2);
Write ("home", 3);
Write ("visitors", 2);
Write ("home", 4);
Write ("home", 5);

Visitors = 2
Home = 5

Under read-my-writes, what possible scores can the score keeper read after this write completes?

2-5
Writer (also reads)

Visitors’ score

S1
S2
S3

Reader

Home score

S4
S5
S6
Under strong consistency, who must S3 have spoken to (directly or indirectly) to satisfy read request?

S2, S5
Visitors’ score

S1
V

S2
V’

S3
V’

Home score

S4
H

S5
H’

S6
H

When does S3 have to talk to S2 and S5? Before writes return or before read returns?

Implementation can be flexible. Guarantee is that inform-flow occurs before read completes.

Writer (also reads)

Writer (also reads)
Under read-my-writes, who must S3 have spoken to (directly or indirectly) to satisfy read request?
For baseball, why is read-my-writes equivalent to strong consistency, even though it is “weaker”? Application only has one writer. Not true in general.
Example 1: score keeper

```
Write ("home", 1);
Write ("visitors", 1);
Write ("home", 2);
Write ("home", 3);
Write ("visitors", 2);
Write ("home", 4);
Write ("home", 5);
```

Visitors = 2
Home = 5

Common theme:

(1) Consider application invariants

(2) Reason about what store must ensure to support application invariants
Example 2: umpire

```plaintext
if first half of 9th inning complete then
  vScore = Read ("visitors");
  hScore = Read ("home");
  if vScore < hScore
    end game;
```

Idea: home team doesn’t need another chance to bat if they are already ahead going into final half inning.
Example 2: umpire

```plaintext
if first half of 9th inning complete then
  vScore = Read ("visitors");
  hScore = Read ("home");
  if vScore < hScore
    end game;
end
```

What invariant must the umpire uphold?

Game should end if home team leads going into final half inning.
Example 2: umpire

```
if first half of 9^{th} inning complete then
  vScore = Read ("visitors");
  hScore = Read ("home");
  if vScore < hScore
    end game;
```

What subset of writes must be visible to the umpire to ensure game ends appropriately?

Reads must show effect of all prior writes (strong consistency)

Example 2: umpire

```plaintext
if first half of 9th inning complete then
  vScore = Read ("visitors");
  hScore = Read ("home");
  if vScore < hScore
    end game;
```

Would read-my-writes work as it did for the score keeper?

No, since the umpire doesn’t issue any writes.
Example 3: radio reporter

```javascript
do {
  vScore = Read ("visitors");
  hScore = Read ("home");
  report vScore, hScore;
  sleep (30 minutes);
}
```

Idea: periodically read score and broadcast it to listeners
Example 3: radio reporter

```do
{  
vScore = Read ("visitors");
hScore = Read ("home");
report vScore, hScore;
sleep (30 minutes);
}
```

What invariants must the radio reporter uphold?

Should only report scores that actually occurred, and score should monotonically increase.
Example 3: radio reporter

```
do  {
  vScore = Read ("visitors");
  hScore = Read ("home");
  report vScore, hScore;
  sleep (30 minutes);
}
```

Do we need strong consistency?

No, since listeners can accept slightly old scores.
Example 3: radio reporter

```plaintext
do {
  vScore = Read (“visitors”);
  hScore = Read (“home”);
  report vScore, hScore;
  sleep (30 minutes);
}
```

Can we get away with eventual consistency (some subset of writes is visible)?

No, eventual consistency can return scores that never occurred.
Under eventual consistency, what possible scores could the radio reporter read after this write completes?

<table>
<thead>
<tr>
<th>Write (&quot;home&quot;, 1);</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write (&quot;visitors&quot;, 1);</td>
</tr>
<tr>
<td>Write (&quot;home&quot;, 2);</td>
</tr>
<tr>
<td>Write (&quot;home&quot;, 3);</td>
</tr>
<tr>
<td>Write (&quot;visitors&quot;, 2);</td>
</tr>
<tr>
<td>Write (&quot;home&quot;, 4);</td>
</tr>
<tr>
<td>Write (&quot;home&quot;, 5);</td>
</tr>
</tbody>
</table>

Visitors = 2
Home = 5

0-0, 0-1, 0-2, 0-4, 0-5, 1-0, … 2-4, 2-5
Visitors’ score

S1
S2
S3

V
V
V

Home score

S4
S5
S6

H
H
H

W1
W2
W3

Reader
Score keeper
Radio reporter
Visitors’ score

V=0
V=1
V=0

Home score

H=1
H=2
H=0

How could reporter read a score of 1-0?
Visitors’ score

V=0
V=1
V=0

Home score

H=1
H=2
H=0

Reader
Score keeper
Radio reporter

1-0
Example 3: radio reporter

\[
\text{do } \{ \\
\text{vScore} = \text{Read} ("visitors"); \\
\text{hScore} = \text{Read} ("home"); \\
\text{report vScore, hScore; } \\
\text{sleep (30 minutes)}; \\
\}\]

How about only consistent prefix (some sequence of writes is visible)?

No. Would give us scores that occurred, but not monotonically increasing.
Visitors’ score

V=0
V=1
V=0

Home score

H=1
H=2
H=0

What prefix of writes is visible?
(initial state)

Reader

Score keeper

Radio reporter

R

0-0

0-1
Visitors’ score

S1: V=0
S2: V=1
S3: V=0

Home score

S4: H=1
S5: H=2
S6: H=0

What additional guarantee do we need?

Also need monotonic reads (see increasing subset of writes)

Reader

Score keeper

Radio reporter

0-0

0-1
Monotonic reads

- **Also called “session consistency”**
  - Reads are grouped under a “session”

- **What extra state/logic is needed for monotonic reads?**
  - System has to know which reads are related
  - Related reads have to be assigned a sequence (i.e., a total order)

- **What extra state/logic is needed for read-my-writes?**
  - System has to know which reads/writes are related
  - Related reads/writes have to be assigned a total order

- **Does read-my-writes guarantee monotonic reads?**
  - (get into groups for five minutes to discuss)
Example 3: radio reporter

do {
  vScore = Read ("visitors");
  hScore = Read ("home");
  report vScore, hScore;
  sleep (30 minutes);
}

Can we get away with bounded staleness (see all "old" writes)?

If we also have consistent prefix, and as long as bound is < 30 minutes.
Example 3: radio reporter

T0  Read ("home");
T1  Read ("visitors");
T2  sleep (30 minutes);
T3  Read ("home");
T4  Read ("visitors");
T5  sleep (30 minutes);
T6  Read ("visitors");
T7  Read ("home");
T8  sleep (30 minutes);
...

Under bounded staleness (bound = 15 minutes, no consistent prefix), what writes must these reads reflect?

Any write that occurred before T3 – 15 minutes
Example 3: radio reporter

```
T0 Read ("home");
T1 Read ("visitors");
T2 sleep (30 minutes);
T3 Read ("home");
T4 Read ("visitors");
T5 sleep (30 minutes);
T6 Read ("visitors");
T7 Read ("home");
T8 sleep (30 minutes);
...
```

Why isn’t unbounded staleness by itself sufficient?

Score must reflect writes that occurred before T3 – (15 minutes), could also reflect more recent writes.
Visitors’ score

S1: V=0
S2: V=0
S3: V=0

Home score

S4: H=0
S5: H=0
S6: H=0

Score keeper

Reader

Radio reporter

0-0

Sleep 30 minutes
Visitors’ score

V1 = 0
V2 = 0
V3 = 0

Home score

H4 = 0
H5 = 0
H6 = 0

Score keeper

W1

Reader

W2

W3

Radio reporter

0-0

Wake up in 10 minutes
Under bounded staleness, what writes can a reporter see?

W1, W2, and W3

Reader

Score keeper

Radio reporter

Visitors’ score

S1
V=0

S2
V=1

S3
V=0

Home score

S4
H=1

S5
H=2

S6
H=0

0-0

Wake up!
Visitors’ score

V=0
V=1
V=0

Home score

H=1
H=2
H=0

Score keeper

Reader

Radio reporter

0-1
0-2
R
R
R

Visitors’ score

S1: V=0
S2: V=1
S3: V=0

Home score

S4: H=1
S5: H=2
S6: H=0

What additional guarantee do we need?

Also need monotonic reads (see increasing subset of writes)

Reader

Score keeper

Radio reporter

R

0-2

0-1

0-2
Example 4: game-recap writer

```c
while not end of game {
  drink beer;
  smoke cigar;
}
do out to dinner;
vScore = Read ("visitors");
hScore = Read ("home");
write recap;
```

Idea: write about game several hours after it has ended
Example 4: game-recap writer

```
while not end of game {
    drink beer;
    smoke cigar;
}
do out to dinner;
vScore = Read ("visitors");
hScore = Read ("home");
write recap;
```

What invariant must the recapper uphold?

Reads must reflect all writes.
Example 4: game-recap writer

```plaintext
while not end of game {
    drink beer;
    smoke cigar;
}
do out to dinner;
vScore = Read ("visitors");
hScore = Read ("home");
write recap;
```

What consistency guarantees could she use?

Strong consistency or bounded staleness w/ bound < time to eat dinner
Example 4: game-recap writer

```c
while not end of game {
    drink beer;
    smoke cigar;
}
do out to dinner;
vScore = Read ("visitors");
hScore = Read ("home");
write recap;
```

What about eventual consistency?

Probably OK most of the time. Bounded to ensure you always get right output.
Example 5: team statistician

wait for end of game;
hScore = Read ("home");
stat = Read ("season-runs");
Write ("season-runs", stat + hScore);

What invariants must statistician uphold?

Season-runs increases monotonically by amount home team scored at the end of the game.
Example 5: team statistician

```
wait for end of game;
hScore = Read ("home");
stat = Read ("season-runs");
Write ("season-runs", stat + hScore);
```

What consistency is appropriate for this read?

Could use strong consistency, bounded staleness (with appropriate bound), maybe eventual consistency
Example 5: team statistician

wait for end of game;
hScore = Read (“home”);
stat = Read (“season-runs”);
Write (“season-runs”, stat + hScore);

What consistency is appropriate for this read?

Could use strong consistency, bounded staleness, or read-my-writes if statistician is only writer
• Geo-replicated stores face fundamental limits common to all distributed systems.
• **FLP result: consensus is impossible in asynchronous distributed systems.**
  - Distributed systems may “partly fail”, and the network may block or delay network traffic arbitrarily.
  - In particular, a network partition may cause a “split brain” in which parts of the system function without an ability to contact other parts of the system (see material on leases).
  - Example of consensus: what was the last value written for X?
• **Popular form of FLP: “Brewer’s conjecture” also known as “CAP theorem”**.
  - We can build systems that are CA, CP, or AP, but we cannot have all three properties at once, ever.
• **To a large extent these limits drive the consistency models.**
  - (Following slides by Chase)
"CAP theorem"

CA: available, and consistent, unless there is a partition.

AP: a reachable replica provides service even in a partition, but may be inconsistent.

CP: always consistent, even in a partition, but a reachable replica may deny service if it is unable to agree with the others (e.g., quorum).

Dr. Eric Brewer

choose two
Fischer-Lynch-Patterson (1985)

• No consensus can be guaranteed in an asynchronous system in the presence of failures.

• Intuition: a “failed” process may just be slow, and can rise from the dead at exactly the wrong time.

• Consensus may occur recognizably, rarely or often.

Network partition  Split brain
Getting precise about CAP #1

- What does consistency mean?
- **Consistency** → Ability to implement an atomic data object served by multiple nodes.
- Requires **linearizability** of ops on the object.
  - Total order for all operations, consistent with causal order, observed by all nodes
  - Also called **one-copy serializability** (1SR): object behaves as if there is only one copy, with operations executing in sequence.
  - Also called **atomic consistency** ("atomic")

Getting precise about CAP #2

• **Availability** \(\rightarrow\) Every request received by a node must result in a response.
  - Every algorithm used by the service must terminate.

• **Network partition** \(\rightarrow\) Network loses or delays arbitrary runs of messages between arbitrary pairs of nodes.
  - Asynchronous network model assumed
  - Service consists of at least two nodes

Getting precise about CAP #3

• **Theorem.** It is impossible to implement an atomic data object that is available in all executions.
  – **Proof.** Partition the network. A write on one side is not seen by a read on the other side, but the read must return a response.

• **Corollary.** Applies even if messages are delayed arbitrarily, but no message is lost.
  – **Proof.** The service cannot tell the difference.

Getting precise about CAP #4

• Atomic and partition-tolerant
  – Trivial: ignore all requests.
  – Or: pick a primary to execute all requests

• Atomic and available.
  – Multi-node case not discussed.
  – But use the primary approach.
  – Need a terminating algorithm to select the primary. Does not require a quorum if no partition can occur. Left as an exercise.

Getting precise about CAP #5

• Available and partition-tolerant
  – Trivial: ignore writes; return initial value for reads.
  – Or: make a best effort to propagate writes among the replicas; reads return any value at hand.

Quorum

- How to build a replicated store that is atomic (consistent) always, and available unless there is a partition?
  - Read and write operations complete only if they are acknowledged by some minimum number (a quorum) of replicas.
  - Set the quorum size so that any read set is guaranteed to overlap with any write set.
  - This property is sufficient to ensure that any read “sees” the value of the “latest” write.
  - So it ensures consistency, but it must deny service if “too many” replicas fail or become unreachable.
Quorum consistency

\[ n = 7 \]
\[ rv = 4 \]
\[ wv = 4 \]

\[ rv = wv = f \]
\[ n = 2f + 1 \]

[Keith Marzullo]
Weighted quorum voting

\[ n = 7 \]
\[ rv = 2 \]
\[ wv = 6 \]

\[ rv + wv = n + 1 \]

Any write quorum must intersect every other quorum.

[Keith Marzullo]