Probabilistically Bounded Staleness

How Eventual is Eventual Consistency?

Peter Bailis, Shivaram Venkataraman,
Michael J. Franklin, Joseph M. Hellerstein, Ion Stoica (UC Berkeley)
BashoChats 002, 28 February 2012
PBS

**what:** consistency prediction

**why:** weak consistency is fast

**how:** measure latencies use **WARS** model
1. Fast
2. Scalable
3. Available
solution:
replicate for
1. capacity
2. fault-tolerance
keep replicas in sync
keep replicas in sync
slow
keep replicas in sync

slow

alternative: sync later
keep replicas in sync
slow
alternative: sync later
inconsistent
↑ consistency, ↑ latency
contact more replicas,
read more recent data

↓ consistency, ↓ latency
contact fewer replicas,
read less recent data
Dynamo:
Amazon’s Highly Available Key-value Store
*SOSP 2007*

Dynamo was followed by:
- Basho
- Riak
- Project Voldemort
- LinkedIn
- Apache, DataStax
- Cassandra
$N = 3$ replicas

- R1 ("key", 1)
- R2 ("key", 1)
- R3 ("key", 1)

read

$R = 3$

Coordinator

client
$N = 3$ replicas

R1 ("key", 1)  R2 ("key", 1)  R3 ("key", 1)

read $R=3$

Coordinator

read("key")

client
Client

\[
N = 3 \text{ replicas}
\]

\[
R_1 ("key", 1) \quad R_2 ("key", 1) \quad R_3 ("key", 1)
\]

read("key")

read

\[
R = 3
\]
$N = 3$ replicas

R1 ("key", 1)  R2 ("key", 1)  R3 ("key", 1)

("key", 1)  ("key", 1)  ("key", 1)

Coordinator

read

$R = 3$
$N = 3$ replicas

R1 ("key", 1)  

R2 ("key", 1)  

R3 ("key", 1)  

Coordinator  

("key", 1)  

("key", 1)  

("key", 1)  

Client  

read  

$R=3$
$N = 3$ replicas

R1 ("key", 1)  R2 ("key", 1)  R3 ("key", 1)

read

$R = 3$
$N = 3$ replicas

$R_1 (\text{"key"}, 1)$

$R_2 (\text{"key"}, 1)$

$R_3 (\text{"key"}, 1)$

$\text{read (\text{"key"})}$

Client
$N = 3$ replicas

R1 ("key", 1) \quad R2 ("key", 1) \quad R3 ("key", 1)

read("key")

Coordinator

read

$R = 3$

client
$N = 3$ replicas

R1 ("key", 1)  R2 ("key", 1)  R3 ("key", 1)

(read)

R=3

Coordinator

client
\( N = 3 \) replicas

- \( R1 ("key", 1) \)
- \( R2 ("key", 1) \)
- \( R3 ("key", 1) \)

Coordinator

\( R = 3 \)

Client

read
$N = 3$ replicas

R1 ("key", 1)  R2 ("key", 1)  R3 ("key", 1)

("key", 1)  ("key", 1)  ("key", 1)

Coordinator

read

$R = 3$
\(N = 3\) replicas

R1 ("key", 1)  R2 ("key", 1)  R3 ("key", 1)

("key", 1)  ("key", 1)  ("key", 1)

Coordinator

("key", 1)

Client

read

\(R=3\)
read

\[ R = 1 \]
$N = 3$ replicas

\begin{align*}
R1 & \text{ ("key", 1)} \\
R2 & \text{ ("key", 1)} \\
R3 & \text{ ("key", 1)}
\end{align*}

read

$R = 1$

Client

Coordinator

read("key")
\[ N = 3 \text{ replicas} \]

- \( R_1 ("key", 1) \)
- \( R_2 ("key", 1) \)
- \( R_3 ("key", 1) \)

Coordinator

\( \text{read("key")} \)

send read to all

\( R = 1 \)

client
$N = 3$ replicas

Coordinator

R1 ("key", 1)  R2 ("key", 1)  R3 ("key", 1)

("key", 1)

Client

read

$R = 1$

send read to all
$N = 3$ replicas

R1 ("key", 1)  R2 ("key", 1)  R3 ("key", 1)

(read
R = 1

Coordinator

("key", 1)

(send read
to all

client)
$N = 3$ replicas

R1 ("key", 1)  R2 ("key", 1)  R3 ("key", 1)

("key", 1)  ("key", 1)

Coordinator

("key", 1)

client

read

$R = 1$

send read to all
$N = 3$ replicas

R1 ("key", 1)  R2 ("key", 1)  R3 ("key", 1)

("key", 1)  ("key", 1)  ("key", 1)

Coordinator

("key", 1)

read $R=1$

send read to all
N replicas/key
read: wait for R replies
write: wait for W acks
Coordinator \( W = 1 \)

\[ \begin{align*}
R1 & ("key", 1) \\
R2 & ("key", 1) \\
R3 & ("key", 1)
\end{align*} \]
Coordinator

\[ W = 1 \]

write("key", 2)
Coordinator

R1 ("key", 1)  R2 ("key", 1)  R3 ("key", 1)

write("key", 2)

Coordinator  W=1
Coordinator

R1 ("key", 2) \rightarrow ack("key", 2) \rightarrow R2 ("key", 1) \rightarrow R3 ("key", 1) \rightarrow Coordinator

W = 1
Coordinator

R1 ("key", 2)  R2 ("key", 1)  R3 ("key", 1)  

ack("key", 2)  

Coordinator  \[ W = 1 \]  

ack("key", 2)
R1 ("key", 2)  \hspace{10mm} R2 ("key", 1)  \hspace{10mm} R3 ("key", 1)

Coordinator

ack("key", 2)

W = 1

Coordinator

read("key")

R = 1
R1 ("key", 2) → Coordinator
R2 ("key", 1) → Coordinator
R3 ("key", 1) → Coordinator

read("key")

W = 1

Coordinator

ack("key", 2)

R = 1
R1 ("key", 2)  R2 ("key", 1)  R3 ("key", 1)

Coordinator

W = 1

Coordinator

R = 1

ack("key", 2)
Coordinator

W = 1

ack(“key”, 2)

Coordinator

R = 1

(“key”, 1)
Coordinator

R1("key", 2)

W = 1

Coordinator

R2("key", 1)

Coordinator

R3("key", 1)

ack("key", 2)

R = 1

("key", 1)
Coordinator

W = 1

ack("key", 2)

R1 ("key", 2) → Coordinator

Coordinator

R2 ("key", 2)

R3 ("key", 1)

R = 1

ack("key", 2)

("key", 1)
Coordinator

R1 ("key", 2)  R2 ("key", 2)  R3 ("key", 2)

ack("key", 2)  ack("key", 2)

W = 1

Coordinator

R = 1

("key", 1)
Coordinator

ack("key", 2)

W = 1

Coordinator

("key", 1)

R = 1
R1 ("key", 2)  R2 ("key", 2)  R3 ("key", 2)

("key", 2)

W = 1

Coordinator

ack("key", 2)

R = 1

Coordinator

("key", 1)
Coordinator

ack("key", 2)

R1 ("key", 2)  R2 ("key", 2)  R3 ("key", 2)

("key", 2)  ("key", 2)

W = 1

Coordinator

R = 1

("key", 1)
Coordinator

\text{W} = 1

\text{ack(“key”, 2)}

\text{Coordinator}

\text{R} = 1

(“key”, 1)
if: $W > \lceil \frac{N}{2} \rceil$
then: $R + W > N$
else: read (at least) last committed version
else: eventual consistency
eventual consistency

“If no new updates are made to the object, eventually all accesses will return the last updated value”

W. Vogels, CACM 2008
How eventual?

How long do I have to wait?
How consistent?

What happens if I don’t wait?
Riak Defaults

$N=3$

$R=2$

$W=2$
Riak Defaults

\[ N=3 \]
\[ R=2 \]
\[ W=2 \]

\[ 2+2 > 3 \]
Riak Defaults

\[
\begin{align*}
N &= 3 \\
R &= 2 \\
W &= 2 \\
2 + 2 &> 3
\end{align*}
\]

Phew, I’m safe!
Riak Defaults

$N=3$
$R=2$
$W=2$

$2+2 > 3$

Phew, I’m safe!

...but what’s my latency cost?
Riak Defaults

\[
\begin{align*}
N &= 3 \\
R &= 2 \\
W &= 2
\end{align*}
\]

Phew, I’m safe!

...but what’s my latency cost?

\[
2 + 2 > 3
\]

Should I change?
strong consistency
strong consistency

low latency
Cassandra:

R=W=1, N=3

by default

\((1+1 \not\geq 3)\)
"In the general case, we typically use [Cassandra’s] consistency level of [R=W=1], which provides maximum performance. Nice!"

--D. Williams,
“HBase vs Cassandra: why we moved”
February 2010
We have a memcached (not memcachedb) in front of it which gives us the atomic operations that we need, so it can take as long as it needs to replicate behind the scenes. If we didn't, we'd use CL-ONE reads/writes for most things except the operations that needed to be atomic, where we'd do CL-QUORUM. But most of our data doesn't need atomic reads/writes.
We have a memcached (not memcachedb) in front of it which gives us the atomic operations that we need, so it can take as long as it needs to replicate behind the scenes. If we didn't, we'd use CL-ONE reads/writes for most things except the operations that needed to be atomic, where we'd do CL-QUORUM. But most of our data doesn't need atomic reads/writes.
Low Value Data

n = 2, r = 1, w = 1
Low Value Data

\[ n = 2, r = 1, w = 1 \]
Mission Critical Data

n = 5, r = 1, w = 5, dw = 5

Consistency or Bust: Breaking a Riak Cluster

http://www.slideshare.net/Jkirkell/breaking-a-riak-cluster
Mission Critical Data

n = 5, r = 1, w = 5, dw = 5

http://www.slideshare.net/Jkirkell/breaking-a-riak-cluster
Voldemort @ LinkedIn

“very low latency and high availability”:

\[ R = W = 1, \ N = 3 \]

N=3 not required, “some consistency”:

\[ R = W = 1, \ N = 2 \]

@strlen, personal communication
Anecdotally, EC "worthwhile" for many kinds of data
Anecdotally, EC “worthwhile” for many kinds of data. How eventual? How consistent?
Anecdotally, EC “worthwhile” for many kinds of data. How eventual? How consistent? “eventual and consistent enough”
Can we do better?
Can we do better?

Probabilistically Bounded Staleness can’t make promises can give expectations
PBS is:
a way to quantify
latency-consistency trade-offs

what’s the latency cost of consistency?
what’s the consistency cost of latency?
PBS is:
a way to quantify
latency-consistency trade-offs
what’s the latency cost of consistency?
what’s the consistency cost of latency?
a SLA for consistency
How eventual?

t-visibility: consistent reads with probability $p$ after $t$ seconds

(e.g., 99.9% of reads will be consistent after 10ms)
Coordinator once per replica Replica
Coordinator \textit{once per replica} Replica

write
Coordinator \textit{once per replica} \hspace{3cm} Replica

write

ack
Coordinator once per replica Replica

write

wait for $W$ responses

ack
Coordinator: Once per replica

Replica

write

wait for $W$ responses

ack

$t$ seconds elapse
Coordinator \textit{once per replica} Replica

- **write**
- **ack**
- **wait for** $W$ **responses**
- **$t$ seconds elapse**
- **read**
Coordinator

once per replica

Replica

write

wait for $W$ responses

t seconds elapse

ack

read

response
Coordinator  once per replica  Replica

write

wait for W responses

t seconds elapse

ack

read

wait for R responses

response
Coordinator **once per replica** Replica

- **write**
- **ack**

wait for **W** responses

$t$ seconds elapse

- **read**
- **response**

wait for **R** responses

response is stale if read arrives before write
Coordinator *once per replica*

Replica

write

wait for $W$ responses

$\begin{array}{c} \text{wait for } R \\ \text{responses} \end{array}$

$t$ seconds elapse

read

response

response is stale if read arrives before write
Coordinator \textit{once per replica} \hfill Replica

wait for $W$ responses

$t$ seconds elapse

wait for $R$ responses

write

ack

read

response

response is stale if read arrives before write
Coordinator \textit{once per replica} Replica

wait for $W$ responses

$t$ seconds elapse

wait for $R$ responses

response is stale if read arrives before write

write

ack

read

response
write

write

$N=2$
W = 1

N = 2

write

ack

write

ack
$W=1$  
$N=2$
W = 1

N = 2

write

ack

read

response

write

ack

read

response
\( W = 1 \)

\( R = 1 \)

\( N = 2 \)
$W = 1$

$R = 1$

$N = 2$

Good
$N=2$
$W = 1$

$N = 2$
\(W = 1\)
\(N = 2\)
write

ack

W=1

read

read

response

R=1

response

N=2

bad

ack
W = 1
R = 1
N = 2

write

read

ack

read

response

response

bad

ack
write

Coordinator once per replica

wait for $W$ responses

$\text{wait for } R \text{ responses}$

$t$ seconds elapse

$\text{read}$

$\text{response}$

ack

$\text{response is stale if read arrives before write}$

Replica
R1 ("key", 2)  R2 ("key", 1)  R3 ("key", 1)

Coordinator

W = 1

Coordinator

R = 1

ack("key", 2)
Coordinator
ack("key", 2)
R1 ("key", 2)  W = 1

Coordinator
R2 ("key", 1)
R3 ("key", 1)
R = 1

Coordinator
("key", 1)
R3 replied before last write arrived!

write("key", 2)

Coordinator

W = 1

ack("key", 2)

Coordinator

R = 1

R1 ("key", 2)  R2 ("key", 1)  R3 ("key", 1)
write

Coordinator \textit{once per replica} Replica

wait for \( W \) responses

\( t \) seconds elapse

wait for \( R \) responses

response is stale if read arrives before write
Coordinator \textit{once per replica} Replica

\begin{itemize}
  \item \textbf{write} \( W \)
  \item wait for \( W \) responses
  \item \( t \) seconds elapse
  \item \textbf{read}
  \item wait for \( R \) responses
  \item response
  \item response is stale if read arrives before write
\end{itemize}
write
(W)

ack
(A)

wait for \( W \) responses

t seconds elapse

wait for \( R \) responses

response

response is stale if read arrives before write

Coordinator \textit{once per replica} Replica
Coordinator once per replica

write

(W)

(A)

ack

wait for W responses

t seconds elapse

read

(R)

Response is stale if read arrives before write

wait for R responses

response
Coordinator \( \text{once per replica} \) Replica

- **write** \((W)\)
- **(A)** ack
- **read** \((R)\)
- \((S)\) response

wait for \(W\) responses

\(t\) seconds elapse

- wait for \(R\) responses

response is stale if read arrives before write
Solving WARS: hard
Monte Carlo methods: easy
To use WARS:

- gather latency data
- run simulation

Cassandra implementation validated simulations; available on Github
How eventual?

**t-visibility**: consistent reads with probability $p$ after $t$ seconds

**key**: WARS model

**need**: latencies
How consistent?

What happens if I don’t wait?
Probability of reading later older than $k$ versions is \textbf{exponentially reduced} by $k$.

Pr(reading latest write) = 99\%
Pr(reading one of last two writes) = 99.9\%
Pr(reading one of last three writes) = 99.99\%
Riak Defaults

\[ N=3 \]
\[ R=2 \]
\[ W=2 \]

Phew, I’m safe!

...but what’s my latency cost?

2+2 > 3

Should I change?
LinkedIn
150M+ users
built and uses Voldemort

Yammer
100K+ companies
uses Riak

Thanks to @strlen and @coda: production latencies
99.9% consistent reads: 
R=2, W=1

\[ t = 13.6 \text{ ms} \]

Latency: 12.53 ms

100% consistent reads: 
R=3, W=1

Latency: 15.01 ms

Latency is combined read and write latency at 99.9th percentile
16.5% faster

LNKD-DISK

99.9% consistent reads: R=2, W=1

\[ t = 13.6 \text{ ms} \]

Latency: 12.53 ms

100% consistent reads: R=3, W=1

Latency: 15.01 ms

Latency is combined read and write latency at 99.9th percentile
99.9% consistent reads: 
R=2, W=1

\[ t = 13.6 \text{ ms} \]

Latency: 12.53 ms

100% consistent reads: 
R=3, W=1

Latency: 15.01 ms

Latency is combined read and write latency at 99.9th percentile

16.5% faster

worthwhile?

N=3
\[ N = 3 \]
N=3

- R=1 W=1
- R=1 W=2
- R=2 W=1

LNNKD-SSD

P(consistency)

0.975

0.970

0.980

0.985

0.990

0.995

1.000

t-visibility (ms)

2.0
LNKD-SSD

99.9% consistent reads:
R=1, W=1
\[ t = 1.85 \text{ ms} \]

Latency: 1.32 ms

100% consistent reads:
R=3, W=1

Latency: 4.20 ms

Latency is combined read and write latency at 99.9th percentile

N=3
99.9% consistent reads: $R=1, W=1$

$t = 1.85$ ms

Latency: $1.32$ ms

100% consistent reads: $R=3, W=1$

Latency: $4.20$ ms

Latency is combined read and write latency at 99.9th percentile

59.5% faster

$N=3$
LNKD-SSD

99.9% consistent reads: $R=1, W=1$
$t = 1.85 \text{ ms}$
Latency: 1.32 ms

100% consistent reads: $R=3, W=1$
Latency: 4.20 ms

Latency is combined read and write latency at 99.9th percentile

N=3

59.5% faster
better payoff!
Coordinators once per replica

- Write (W)
- Ack (A)
- Read (R)
- Response (S)

- Wait for W responses
- Wait for R responses
- t seconds elapse

- Critical factor in staleness
  - Response is stale if read arrives before write
Probability Density Function

- **Latency**
  - Low variance
Probability Density Function

- low variance
- high variance
Coordinator \( \textit{once per replica} \) Replica

- **Write** \((W)\) from Coordinator
- **Ack** \((A)\) from Replica
- **Read** \((R)\) from Replica
- **Response** \((S)\) from Response

- **Wait for** \(W\) **responses**
- **Wait for** \(R\) **responses**
- **If** \(t\) **seconds elapse**
- **Response is** **stale** if read arrives before write

SSDs reduce variance compared to disks!
99.9% consistent reads:
R = 1, W = 1
\[ t = 202.0 \text{ ms} \]
Latency: 43.3 ms

100% consistent reads:
R = 3, W = 1
Latency: 230.06 ms

Latency is combined read and write latency at 99.9th percentile
99.9% consistent reads: \( R=1, W=1 \)

\[ t = 202.0 \text{ ms} \]

Latency: 43.3 ms

100% consistent reads: \( R=3, W=1 \)

Latency: 230.06 ms

Latency is combined read and write latency at 99.9th percentile
99.9% consistent reads:
R=1, W=1
$t = 202.0 \text{ ms}$
Latency: 43.3 ms
100% consistent reads:
R=3, W=1
Latency: 230.06 ms

Latency is combined read and write latency at 99.9th percentile

81.1% faster
even better payoff!

YMMR

N=3
Riak Defaults

N=3
R=2
W=2

Phew, I’m safe!

...but what’s my latency cost?

2+2 > 3

Should I change?
Is low latency worth it?
Is low latency worth it?
PBS can tell you.
Is low latency worth it?

PBS can tell you.

(and PBS is easy)
How Eventual is Eventual Consistency? PBS in action under Dynamo-style quorums

You have at least a 74.8 percent chance of reading the last written version 0 ms after it commits.
You have at least a 92.2 percent chance of reading the last written version 10 ms after it commits.
You have at least a 99.96 percent chance of reading the last written version 100 ms after it commits.

Replica Configuration
N: 3
R: 1
W: 1

Read Latency: Median 8.43 ms, 99.9th %ile 36.97 ms
Write Latency: Median 8.38 ms, 99.9th %ile 38.28 ms

Tolerable Staleness: 1 version
Accuracy: 2500 iterations/point

Operation Latency: Exponentially Distributed CDFs
W: Write Request to Replica
A: Replica Write Ack
R: Read Request to Replica
S: Replica Read Response
Workflow

1. Metrics
2. Simulation
3. Set \( N, R, W \)
4. Profit
what: consistency prediction

why: weak consistency is fast

how: measure latencies use WARS model
strong consistency

low latency
latency vs. consistency trade-offs
fast and simple modeling
large benefits
be more

bailis.org/projects/pbs/#demo

@pbailis
VLDB 2012 early print
tinyurl.com/pbsspaper
cassandra patch
github.com/pbailis/cassandra-pbs
Extra Slides
PBS and apps
staleness requires either:

staleness-tolerant data structures
timelines, logs
cf. commutative data structures
logical monotonicity

asynchronous compensation code
detect violations after data is returned; see paper
write code to fix any errors

cf. “Building on Quicksand”
memories, guesses, apologies
asynchronous compensation

minimize:

(compensation cost) \times (\# \ of \ expected \ anomalies)
Read only newer data?

*(monotonic reads session guarantee)*

\[
\text{# versions tolerable staleness} = \frac{\text{client’s read rate}}{\text{global write rate}}
\]

(for a given key)
Failure?
Treat failures as latency spikes
How long do partitions last?
what time interval?

99.9% uptime/yr
⇒ 8.76 hours downtime/yr

8.76 consecutive hours down
⇒ bad 8-hour rolling average
what time interval?

99.9% uptime/yr
⇒ 8.76 hours downtime/yr
8.76 consecutive hours down
⇒ bad 8-hour rolling average

hide in tail of distribution OR
continuously evaluate SLA, adjust
Give me (and academia) failure data!
In paper:

- Closed-form analysis
- Monotonic reads
- Staleness detection
- Varying $N$
- WAN model
- Production latency data

tinyurl.com/pbspaper
$t$-visibility depends on:

1) message delays
2) background version exchange (anti-entropy)
$t$-visibility depends on:

1) message delays

2) background version exchange (anti-entropy)

anti-entropy:

- only decreases staleness
- comes in many flavors
- hard to guarantee rate

Focus on message delays first
N=3  (LNKD-SSD and LNKD-DISK identical for reads)
\[ N=3 \]

A plot showing the Cumulative Distribution Function (CDF) of write latency for different systems. The axes are labeled as follows:

- **Y-axis**: CDF
- **X-axis**: Write Latency (ms)

The plot includes four datasets:
- LNKD-SSD (green triangles)
- LNKD-DISK (red circles)
- YMMR (black squares)
- WAN (blue inverted triangles)

Each dataset represents different performance characteristics under the condition of \( W=3 \).
$N=3$

**YMMR**

- $R=1 \ W=1$
- $R=1 \ W=2$
- $R=2 \ W=1$

**Graph:**

- **Y-axis:** $P(\text{consistency})$  
- **X-axis:** t-visibility (ms)

- Lines represent different combinations of $R$ and $W$.
$N = 3$

$P(\text{consistency})$ vs $t$-visibility (ms)

- $R = 1$, $W = 1$
- $R = 1$, $W = 2$
- $R = 2$, $W = 1$

$10^2$
$N=3$

Graph showing $P(\text{consistency})$ as a function of $t$-visibility (ms). The graph includes three curves corresponding to different conditions: $R=1, W=1$, $R=1, W=2$, and $R=2, W=1$. The $y$-axis represents $P(\text{consistency})$, ranging from 0.88 to 1.00, and the $x$-axis represents $t$-visibility (ms), ranging from $10^1$ to $10^3$. The $P(\text{consistency})$ value of 0.90 is circled, indicating a specific point of interest on the graph.
Synthetic, Exponential Distributions

$P(\text{consistency})$ vs. $t$-visibility (ms)

- $ARS_{\lambda}:W_{\lambda}$
- $1:4$
- $1:0.50$
- $1:2$
- $1:0.20$
- $1:1$
- $1:0.10$
Synthetic, Exponential Distributions

$P(\text{consistency})$ vs $t\text{-visibility (ms)}$

- $w \frac{1}{4}x$ ARS
- $w 10x$ ARS

Graph showing the relationship between $P(\text{consistency})$ and $t\text{-visibility}$ with different ARS$\lambda$:W$\lambda$ ratios.
$N = 3$ replicas

R1  R2  R3

Write to W, read from R replicas
$N = 3$ replicas

Write to $W$, read from $R$ replicas

quorum system: guaranteed intersection

$\{\{R1, R2\}, \{R2, R3\}, \{R1, R3\}\}$

$R=W=3$ replicas

$\{\{R1, R2\}, \{R2, R3\}\}$

$R=W=2$ replicas
Write to $W$, read from $R$ replicas

**quorum system:**
- guaranteed intersection
  - $R=W=3$ replicas
  - $R=W=2$ replicas
  - $R=W=1$ replicas

**partial quorum system:**
- may not intersect
  - $R=W=1$ replicas