Probabilistically Bounded Staleness for Practical Partial Quorums

Peter Bailis, Shivaram Venkataraman, Mike Franklin, Joe Hellerstein, Ion Stoica

UC Berkeley
NOTE TO READER

watch a recording (of an earlier talk) at:
http://vimeo.com/37758648
NOTE TO READER
play with a live demo at:
http://pbs.cs.cs.berkeley.edu/#demo
strong consistency: higher latency

eventual consistency: lower latency
consistency is a binary choice

strong eventual
our focus:
latency vs. consistency informed by practice

not in this talk:
availability, partitions, failures
our contributions

quantify eventual consistency:
  wall-clock time ("how eventual?")
  versions ("how consistent?")

analyze real-world systems:
  EC is often strongly consistent
  describe when and why
intro
system model
practice
metrics
insights
integration
Dynamo:
Amazon’s Highly Available Key-value Store
*SOSP 2007*

- Apache, DataStax
- LinkedIn
- Cassandra
- Project Voldemort
- Basho
- Riak
$N$ replicas/key
read: wait for $R$ replies
write: wait for $W$ acks

$N=3$
$R=2$
if: $R + W > N$  
then: “strong” consistency  
else: eventual consistency
“strong” consistency $\equiv$ reads return the last acknowledged write or an in-flight write \((\text{per-key})\)

\[ R + W > N \]
<table>
<thead>
<tr>
<th></th>
<th>99th</th>
<th>99.9th</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1x</td>
<td>1x</td>
</tr>
<tr>
<td><strong>R</strong></td>
<td>1.59x</td>
<td>2.35x</td>
</tr>
<tr>
<td></td>
<td>4.8x</td>
<td>6.13x</td>
</tr>
<tr>
<td><strong>W</strong></td>
<td>2.01x</td>
<td>1.9x</td>
</tr>
<tr>
<td></td>
<td>4.96x</td>
<td>14.96x</td>
</tr>
</tbody>
</table>
consistency, \[\uparrow\] latency
wait for more replicas,
read more recent data

\[\downarrow\] consistency, \[\downarrow\] latency
wait for fewer replicas,
read less recent data
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?
strong consistency → higher latency
eventual consistency → lower latency
Cassandra:

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

by default

\((1+1 \not\geq 3)\)
eventual consistency in the wild

“maximum performance” okay for “most data”

“very low latency” “general case”
anecdotally, EC
“good enough” for
many kinds of data

How eventual?

How consistent?

“eventual and consistent enough”
Can we do better?

Can’t make promises
can give expectations

Probabilistically
Bounded Staleness
intro
system model
practice
metrics
insights
integration
How eventual?

How long do I have to wait?
How eventual?

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

(e.g., 10ms after write, 99.9% of reads consistent)
$t$-visibility depends on messaging and processing delays.
Coordinator \textit{once per replica} Replica

\textbf{write} \quad \textbf{ack} \quad \textbf{read}

wait for \textit{W} responses

\textit{t} seconds elapse

\textbf{read}

wait for \textit{R} responses

response is stale if read arrives before write
Alice

write

R1

ack

Bob

read

response

inconsistent

W = 1

R = 1

N = 2

R2
wait for $W$ responses

$t$ seconds elapse

wait for $R$ responses

response is stale if read arrives before write

Coordinator

write

(W)

ack

response

wait for responses

once per replica

Replica

read

(R)

(S)

response

write

(W)

(A)
solving WARS: order statistics dependent variables

instead: Monte Carlo methods
to use WARS:

gather latency data

\[
\begin{array}{cccc}
W & A & R & S \\
53.2 & 10.3 & 15.3 & 9.6 \\
44.5 & 8.2 & 22.4 & 14.2 \\
101.1 & 11.3 & 19.8 & 6.7 \\
\ldots & \ldots & \ldots & \ldots \\
\end{array}
\]

run simulation

Monte Carlo, sampling
WARS accuracy
real Cassandra cluster varying latencies:
t-visibility RMSE: 0.28%
latency N-RMSE: 0.48%
How eventual?

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

key: WARS model

need: latencies
LinkedIn
175M+ users
built and uses Voldemort

Yammer
100K+ companies
uses Riak

production latencies
fit gaussian mixtures
The graph illustrates the consistency probability $P(\text{consistency})$ as a function of time $t$ (in milliseconds) for different configurations of $R$ and $W$ for the LNKD-DISK system. The configurations tested are:

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

The graph shows that with increasing time, the probability of consistency increases for all configurations. Notably, the graph indicates that with $N=3$, the time to achieve a consistency probability of 0.9 is approximately 10 ms.
Latency is combined read and write latency at 99.9th percentile.

**LNKD-DISK**

**N=3**

16.5% faster worthwhile?

- **R=2, W=1,** \( t = 13.6 \text{ ms} \)
  - 99.9% consistent:
    - Latency: 12.53 ms

- **R=3, W=1**
  - 100% consistent:
    - Latency: 15.01 ms
Latency is combined read and write latency at 99.9th percentile

N=3

LNKD-SSD

R=1, W=1, \( t = 1.85 \text{ ms} \)

99.9% consistent:
Latency: 1.32 ms

R=3, W=1

100% consistent:
Latency: 4.20 ms

59.5% faster

Latency is combined read and write latency at 99.9th percentile
N=3

CDF

Write Latency (ms)

W=3

LNKD-SSD

LNKD-DISK

YMMR

WAN
Coordinator | once per replica | Replica
---|---|---
write \((W)\) | ack \((A)\) | SSDs reduce variance compared to disks!
wait for \(W\) responses | | 
\(t\) seconds elapse | | 
read \((R)\) | response is stale if read arrives before write | 
wait for \(R\) responses | response \((S)\) |
Yammer

N=3

latency
↓ 81.1% (187ms)

Yammer

99.9th t-visibility
202 ms
How consistent?

$k$-staleness (versions)

monotonic reads

quorum load

in the paper
in the paper

\(\langle k, t \rangle\)-staleness: versions and time
in the paper

latency distributions

WAN model

varying quorum sizes

staleness detection
intro
system model
practice
metrics
insights
integration
Integration

1. Tracing
2. Simulation
3. Tune N,R,W

Cassandra  Project Voldemort
Introduction

Cassandra supports a variety of replication configurations: ReplicationFactor is set per-ColumnFamily and ConsistencyLevel is set per-request. Setting ConsistencyLevel to QUORUM for reads and writes ensures strong consistency, but QUORUM is often slower than ONE, TWO, or THREE. What should users choose?

This patch provides a latency-consistency analysis within nodetool. Users can accurately predict Cassandra's behavior in their production environments without interfering with performance.
ubuntu@ip-10-46-87-156:/cassandra-pbs$ bin/nodetool -h ec2-23-20-168-89.compute-1.amazonaws.com predictconsistency 3 75 1
75ms after a given write, with maximum version staleness of k=1
N=3, R=1, W=1
Probability of consistent reads: 0.716500
Average read latency: 31.170300ms (99.900th %ile 193ms)
Average write latency: 42.873798ms (99.900th %ile 192ms)

N=3, R=1, W=2
Probability of consistent reads: 0.902400
Average read latency: 30.958000ms (99.900th %ile 189ms)
Average write latency: 106.877098ms (99.900th %ile 240ms)

N=3, R=1, W=3
Probability of consistent reads: 1.000000
Average read latency: 30.104000ms (99.900th %ile 192ms)
Average write latency: 171.652298ms (99.900th %ile 341ms)

N=3, R=2, W=1
Probability of consistent reads: 0.934200
Average read latency: 84.446602ms (99.900th %ile 231ms)
Average write latency: 42.800301ms (99.900th %ile 194ms)

N=3, R=2, W=2
Probability of consistent reads: 1.000000
Average read latency: 82.663902ms (99.900th %ile 238ms)
Average write latency: 106.141296ms (99.900th %ile 236ms)
How Eventual is Eventual Consistency?

PBS in action under Dynamo-style quorums

P(Consistency)

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

http://pbs.cs.berkeley.edu/#demo
Related Work

**Quorum Systems**
- probabilistic quorums [PODC ’97]
- deterministic k-quorums [DISC ’05, ’06]

**Consistency Verification**
- Golab et al. [PODC ’11]
- Bermbach and Tai [M4WSOC ’11]
- Wada et al. [CIDR ’11]
- Anderson et al. [HotDep ’10]
- Transactional consistency: Zellag and Kemme [ICDE ’11], Fekete et al. [VLDB ’09]

**Bounded Staleness Guarantees**
- TACT [OSDI ’00]
- FRACS [ICDCS ’03]
- AQuA [IEEE TPDS ’03]

**Latency-Consistency**
- Daniel Abadi [Computer ’12]
- Kraska et al. [VLDB ’09]
strong consistency
higher latency

eventual consistency
lower latency
consistency is a continuum
quantify eventual consistency
model staleness in time, versions
latency-consistency trade-offs
analyze real systems and hardware

quantify which choice is best and explain why EC is often strongly consistent
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)

\[
\# \text{ versions tolerable staleness} = \frac{\text{client's read rate}}{\text{global write rate}} \quad \text{(for a given key)}
\]
Failure?
Treat failures as 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

hide in tail of distribution OR
continuously evaluate SLA, adjust
\[ W = 3 \]

CDF

Write Latency (ms)

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

$LNKD$-$SSD$ and $LNKD$-$DISK$ identical for reads

$N=3$
<k,t>-staleness: versions and time

approximation: exponentiate

\( t \)-staleness by \( k \)
Synthetic, Exponential Distributions

$N=3, W=1, R=1$

$\lambda$: $\frac{w}{ARS}$
concurrent writes: deterministically choose

Coordinator

(“key”, 1) (“key”, 2)

R=2
Table 1: LinkedIn Voldemort single-node production latencies.

<table>
<thead>
<tr>
<th>%ile</th>
<th>Latency (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>4.85</td>
</tr>
<tr>
<td>95</td>
<td>15</td>
</tr>
<tr>
<td>99</td>
<td>25</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>%ile</th>
<th>Latency (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>0.58</td>
</tr>
<tr>
<td>95</td>
<td>1</td>
</tr>
<tr>
<td>99</td>
<td>2</td>
</tr>
<tr>
<td>%ile</td>
<td>Read Latency (ms)</td>
</tr>
<tr>
<td>-------</td>
<td>------------------</td>
</tr>
<tr>
<td>Min</td>
<td>1.55</td>
</tr>
<tr>
<td>50</td>
<td>3.75</td>
</tr>
<tr>
<td>75</td>
<td>4.17</td>
</tr>
<tr>
<td>95</td>
<td>5.2</td>
</tr>
<tr>
<td>98</td>
<td>6.045</td>
</tr>
<tr>
<td>99</td>
<td>6.59</td>
</tr>
<tr>
<td>99.9</td>
<td>32.89</td>
</tr>
<tr>
<td>Max</td>
<td>2979.85</td>
</tr>
<tr>
<td>Mean</td>
<td>9.23</td>
</tr>
<tr>
<td>Std. Dev.</td>
<td>83.93</td>
</tr>
<tr>
<td>Mean Rate</td>
<td>718.18 gets/s</td>
</tr>
</tbody>
</table>

Table 2: Yammer Riak $N=3$, $R=2$, $W=2$ production latencies.
<table>
<thead>
<tr>
<th></th>
<th>Distribution</th>
<th>Parameters</th>
<th>Percentage</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>LNKD-SSD</td>
<td>$W = A = R = S$ :</td>
<td>$W$: Pareto, $x_m = 0.235$, $\alpha = 10$</td>
<td>91.22%</td>
<td>0.55%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Exponential, $\lambda = 1.66$</td>
<td>8.78%</td>
<td></td>
</tr>
<tr>
<td>LNKD-DISK</td>
<td>$W$:</td>
<td>Pareto, $x_m = 1.05$, $\alpha = 1.51$</td>
<td>38%</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Exponential, $\lambda = 0.183$</td>
<td>62%</td>
<td>0.26%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$A = R = S$: LNKD-SSD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>YMMR</td>
<td>$W$:</td>
<td>Pareto, $x_m = 3$, $\alpha = 3.35$</td>
<td>93.9%</td>
<td>1.84%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Exponential, $\lambda = 0.0028$</td>
<td>6.1%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$A = R = S$:</td>
<td>Pareto, $x_m = 1.5$, $\alpha = 3.8$</td>
<td>98.2%</td>
<td>0.06%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Exponential, $\lambda = 0.0217$</td>
<td>1.8%</td>
<td></td>
</tr>
</tbody>
</table>

Table 3: Distribution fits for production latency distributions from LinkedIn (LNKD-*) and Yammer (YMMR).
<table>
<thead>
<tr>
<th></th>
<th>LNKD-SSD</th>
<th></th>
<th>LNKD-DISK</th>
<th></th>
<th>YMMR</th>
<th></th>
<th>WAN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$L_r$</td>
<td>$L_w$</td>
<td>$t$</td>
<td>$L_r$</td>
<td>$L_w$</td>
<td>$t$</td>
<td>$L_r$</td>
</tr>
<tr>
<td>$R=1$, $W=1$</td>
<td>0.66</td>
<td>0.66</td>
<td>1.85</td>
<td>0.66</td>
<td>10.99</td>
<td>45.5</td>
<td>5.58</td>
</tr>
<tr>
<td>$R=1$, $W=2$</td>
<td>0.66</td>
<td>1.63</td>
<td>1.79</td>
<td>0.65</td>
<td>20.97</td>
<td>43.3</td>
<td>5.61</td>
</tr>
<tr>
<td>$R=2$, $W=1$</td>
<td>1.63</td>
<td>0.65</td>
<td>0</td>
<td>1.63</td>
<td>10.9</td>
<td>13.6</td>
<td>32.6</td>
</tr>
<tr>
<td>$R=2$, $W=2$</td>
<td>1.62</td>
<td>1.64</td>
<td>0</td>
<td>1.64</td>
<td>20.96</td>
<td>0</td>
<td>33.18</td>
</tr>
<tr>
<td>$R=3$, $W=1$</td>
<td>4.14</td>
<td>0.65</td>
<td>0</td>
<td>4.12</td>
<td>10.89</td>
<td>0</td>
<td>219.27</td>
</tr>
<tr>
<td>$R=1$, $W=3$</td>
<td>0.65</td>
<td>4.09</td>
<td>0</td>
<td>0.65</td>
<td>112.65</td>
<td>0</td>
<td>5.63</td>
</tr>
<tr>
<td>$R=1$, $W=3$</td>
<td>3.4</td>
<td>55.12</td>
<td>113.0</td>
<td>3.4</td>
<td>167.64</td>
<td>0</td>
<td>151.3</td>
</tr>
<tr>
<td>$R=2$, $W=2$</td>
<td>151.3</td>
<td>167.72</td>
<td>0</td>
<td>151.31</td>
<td>167.72</td>
<td>0</td>
<td>153.86</td>
</tr>
<tr>
<td>$R=1$, $W=3$</td>
<td>3.44</td>
<td>241.55</td>
<td>0</td>
<td>3.44</td>
<td>241.55</td>
<td>0</td>
<td>5.63</td>
</tr>
</tbody>
</table>
\[ N = 3 \text{ replicas} \]

\begin{align*}
R_1 \text{ ("key", 1)} & \quad R_2 \text{ ("key", 1)} & \quad R_3 \text{ ("key", 1)} \\
\text{read ("key")} & & \\
\text{client} \end{align*}
$N = 3$ replicas

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

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

read("key")

Coordinator

read("key")

read

R=3

client
N = 3 replicas

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

(read("key")

(send

read to all

R = 1

(client

read("key")

Coordinator

(read("key")

("key", 1)

("key", 1)

("key", 1)

("key", 1)
Coordinator

R1 ("key", 2)

R2 ("key", 2)

R3 ("key", 2)

W = 1

R1 = 1

Coordinator

ack("key", 2)

write("key", 2)

read("key")

ack("key", 2)

("key", 2)

("key", 2)

("key", 1)

("key", 1)
N=3