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

UC Berkeley
PBS

Probabilistically Bounded Staleness
1. Fast
2. Scalable
3. Available
solution: replicate for
1. request capacity
2. reliability
solution:
replicate for

1. request capacity
2. reliability
solution: replicate for

1. request capacity
2. reliability
keep replicas in sync
keep replicas in sync
keep replicas in sync
keep replicas in sync
keep replicas in sync
keep replicas in sync
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
keep replicas in sync

slow

alternative: sync later
keep replicas in sync

slow

alternative: sync later

inconsistent
keep replicas in sync

slow

alternative: sync later

inconsistent
keep replicas in sync
slow
alternative: sync later
inconsistent
keep replicas in sync

slow

alternative: sync later

inconsistent
Increasing consistency, increasing latency:
- contact more replicas,
- read more recent data.

Decreasing consistency, decreasing latency:
- contact fewer replicas,
- read less recent data.
consistency, latency
contact more replicas, read more recent data

consistency, latency
contact 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?
PBS

**Problem:** no guarantees with eventual consistency

**Solution:** consistency prediction

**Technique:** measure latencies use WARS model
Dynamo:
Amazon’s Highly Available Key-value Store
SOSP 2007

Apache, DataStax

Cassandra

basho

riak

Linkedin

Project Voldemort
$N = 3$ replicas

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

read

R$=3$

Coordinator

client
$N = 3$ replicas

$R_1 ("key", 1)$
$R_2 ("key", 1)$
$R_3 ("key", 1)$

read

$R=3$

Coordinator

read("key")

client
$N = 3$ replicas

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

read("key")

Coordinator

read

$R = 3$
$N = 3$ replicas

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

$R=3$
$N = 3$ replicas

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

read

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

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

**read**

**Coordinator**

- read("key")

**client**
$N = 3$ replicas

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

read("key")

Coordinator

read

$R = 3$
$N = 3$ replicas

$R = 3$

(read)

client

Coordinator

$R_1$ ("key", 1)

$R_2$ ("key", 1)

$R_3$ ("key", 1)
$N = 3$ replicas

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

Client reads $R = 3$
$N = 3$ replicas

R1 ("key", 1) \rightarrow ("key", 1) \rightarrow Coordinator

R2 ("key", 1) \rightarrow ("key", 1) \rightarrow Coordinator

R3 ("key", 1) \rightarrow ("key", 1) \rightarrow Coordinator

read $R=3$

client
$N = 3$ replicas

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

("key", 1) — ("key", 1) — ("key", 1)

Coordinator

("key", 1)

read

$R = 3$
$N = 3$ replicas

- $R_1 (\text{"key"}, 1)$
- $R_2 (\text{"key"}, 1)$
- $R_3 (\text{"key"}, 1)$

Client

Coordinator

$R = 1$
$N = 3$ replicas

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

read

$R = 1$

Coordinator

read("key")

client
$N = 3$ replicas

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

Coordinator

read("key")

client

send read to all

R = 1
\[ N = 3 \text{ replicas} \]

- \( R_1 (\text{“key”, 1}) \)
- \( R_2 (\text{“key”, 1}) \)
- \( R_3 (\text{“key”, 1}) \)

Coordinator

(client)

send read to all

\( R = 1 \)

read
$N = 3$ replicas

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

("key", 1)

Coordinator

("key", 1)

client

send read to all

read $R = 1$
N = 3 replicas

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

(send read to all)

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)

client

read $R=1$  send read to all
\textbf{N} replicas/key
read: wait for \textbf{R} replies
write: wait for \textbf{W} acks
R1 ("key", 1)  R2 ("key", 1)  R3 ("key", 1)

Coordinator $W=1$
Coordinator

W = 1

write("key", 2)
Coordinator

\[ \text{write("key", 2)} \]

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

W = 1
ack("key", 2)
R1 ("key", 2)  R2 ("key", 1)  R3 ("key", 1)

 Coordinator

W = 1

ack("key", 2)

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

Coordinator

W = 1

ack("key", 2)

Coordinator

R = 1

read("key")
Coordinator

R1 ("key", 2)

R2 ("key", 1)

R3 ("key", 1)

Coordinator

W = 1

read("key")

Coordinator

R = 1

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

Coordinator

W = 1

Coordinator

R = 1

ack("key", 2)
Coordinator

ack("key", 2)

W = 1

Coordinator

("key", 1)

R = 1

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

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

W = 1
Coordinator

ack("key", 2)

Coordinator

R = 1

("key", 1)
Coordinator

ack("key", 2)

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

W = 1

Coordinator

R = 1

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

W = 1

 Coordinator

Coordinator

R = 1

ack("key", 2)
R1 ("key", 2) \rightarrow \text{Coordinator} \rightarrow \text{W} = 1 \\
R2 ("key", 2) \rightarrow \text{Coordinator} \\
R3 ("key", 1) \rightarrow \text{Coordinator} \rightarrow \text{R} = 1 \\
ack("key", 2) \rightarrow \text{Coordinator} \\
ack("key", 2) \rightarrow W = 1 \\
R = 1
R1 ("key", 2)  R2 ("key", 2)  R3 ("key", 2)

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

Coordinator

W = 1

Coordinator  R = 1

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

Coordinator

W = 1

Coordinator

ack("key", 2)

R = 1

("key", 1)
Coordinator

R1 ("key", 2)

R2 ("key", 2)

R3 ("key", 2)

("key", 2)

W = 1

R = 1

Coordinator

ack("key", 2)
Coordinator

W = 1

ack("key", 2)

Coordinator

R = 1

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

Coordinator

ack("key", 2)

W = 1

Coordinator

R = 1

("key", 1)
if: \[ R + W > N \] then: “strong” consistency

else: eventual consistency
strong consistency
strong consistency
lower latency
strong consistency
lower latency
strong consistency
lower latency
Cassandra:

R=W=1, N=3

by default

(1+1 \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

you are viewing a single comment's thread.
view the rest of the comments →

[-] ketralnis  [S] 13 points 1 year ago

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.
NoSQL Primer
Low Value Data

\[ n = 2, r = 1, w = 1 \]
Low Value Data

n = 2, r = 1, w = 1
Mission Critical Data

\[ n = 5, r = 1, w = 5, dw = 5 \]

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

\[ n = 5, r = 1, w = 5, dw = 5 \]
“very low latency and high availability”: \( R=W=1, N=3 \)

\( N=3 \) not required, “some consistency”: \( R=W=1, N=2 \)

Alex Feinberg, 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?
an “SLA” for consistency
How eventual?

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

(e.g., 99.9% of reads will be consistent after 10ms)
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
focus on
**steady state**
with failures: unavailable or sloppy
Coordinator  \textit{once per replica}  Replica

write
Coordinator \textit{once per replica} Replica

\begin{itemize}
  \item write
  \item ack
\end{itemize}
Coordinator \textit{once per replica} Replica

write

wait for $W$ responses

ack
Coordinator: once per replica

Replica

write

wait for $W$ responses

t seconds elapse

ack
Coordinator: Once per replica

- Write
- Wait for W responses
- \( t \) seconds elapse
- Read

Replica:

- Ack
Coordinator \textit{once per replica} Replica

write

wait for $W$ responses

$\text{t seconds elapse}$

ack

read

response

Time
Coordinator \textit{once per replica} Replica

write

wait for $W$ responses

ack

t seconds elapse

read

wait for $R$ responses

response
Coordinator \textit{once per replica} \hspace{1cm} \textbf{Replica}

\begin{align*}
\text{write} & \quad \text{write} \\
\text{wait for W responses} & \quad \text{ack} \\
& \quad t \text{ seconds elapse} \\
\text{read} & \quad \text{response is stale if read arrives before write} \\
\text{wait for R responses} & \quad \text{response}
\end{align*}
Coordinator \textit{once per replica} Replica

- write
- ack
- read
- response
- response is stale if read arrives before write
- wait for \( W \) responses
- \( t \) seconds elapse
- wait for \( R \) responses

Diagram:
- Write to replica
- Acknowledgment from replica
- Read from replica
- Timer for response
- Response is stale if read arrives before write
Coordinator

wait for $W$ responses

$t$ seconds elapse

wait for $R$ responses

once per replica

write

ack

read

response

Replica

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

write

ack

read

response

response is stale if read arrives before write
\[ N = 2 \]
write

\[ W = 1 \]

ack

write

ack

\[ N = 2 \]
$W = 1$  

$N = 2$
write

ack

write

ack

W=1

read

read

N=2
\[ W = 1 \]

\[ N = 2 \]
W = 1
write
ack
read
response

R = 1
write
ack
read
response

good

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

$N=2$  

write  

ack  

write  

ack

Time
\( W = 1 \)

\( N = 2 \)
W = 1

R = 1

N = 2
W = 1

R = 1

N = 2

write

ack

read

response

read

response

bad

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

- write
- ack
- read
- wait for \(W\) responses
- \(t\) seconds elapse
- \(R\) responses

response is stale if read arrives before write
Coordinator

\[ \text{ack(“key”, 2)} \]

\[ \text{W} = 1 \]

\[ \text{Coordinator} \]

\[ \text{R} = 1 \]

\[ \text{R1 (“key”, 2)} \]

\[ \text{R2 (“key”, 1)} \]

\[ \text{R3 (“key”, 1)} \]
R3 replied before last write arrived!

write("key", 2)

 Coordinator

ack("key", 2)

 Coordinator

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

W = 1

R = 1

("key", 1)  

("key", 1)

("key", 1)
Coordinator \textit{once per replica} Replica

wait for $W$ responses

$t$ seconds elapse

wait for $R$ responses

write

ack

read

response is stale if read arrives before write
write (W)
write
(W)
ack
(A)
wait for W responses
t seconds elapse
read
response is stale if read arrives before write

Coordinator once per replica

wait for R responses
response

Replica

Coordinator

once per replica

Replica
write (W)

wait for W responses

wait for R responses

response is stale if read arrives before write

Coordinator once per replica

read (R)

response

t seconds elapse

wait for R responses
write
(W)
ack
(A)
read
(R)

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: easier
To use WARS:

**gather latency data**

<table>
<thead>
<tr>
<th>W</th>
<th>A</th>
<th>R</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>53.2</td>
<td>10.3</td>
<td>15.3</td>
<td>9.6</td>
</tr>
<tr>
<td>44.5</td>
<td>8.2</td>
<td>22.4</td>
<td>14.2</td>
</tr>
<tr>
<td>101.1</td>
<td>11.3</td>
<td>19.8</td>
<td>6.7</td>
</tr>
</tbody>
</table>

run simulation

Monte Carlo, sampling
How eventual?

$t\text{-visibility}$: 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 **exponentially reduced** by $k$

- $\Pr(\text{reading latest write}) = 99\%$
- $\Pr(\text{reading one of last two writes}) = 99.9\%$
- $\Pr(\text{reading one of last three writes}) = 99.99\%$
[patch] Support consistency-latency prediction in nodetool

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.

https://issues.apache.org/jira/browse/CASSANDRA-4261
Cassandra cluster, injected latencies:

WARS Simulation accuracy

t-staleness RMSE: 0.28%

latency N-RMSE: 0.48%
LinkedIn
150M+ users
built and uses Voldemort

Yammer
100K+ companies
uses Riak

production latencies
fit gaussian mixtures
LNKD-DISK

P(consistency)

N=3

t-visibility (ms)

R=1 W=1
R=1 W=2
R=2 W=1
The figure shows the consistency probability ($P(\text{consistency})$) as a function of time visibility ($t$) for different configurations of LNKD-DISK. The x-axis represents the time visibility in milliseconds (ms), with a log scale from $10^1$ to $10^2$ ms. The y-axis represents the probability of consistency. The configurations are labeled as $R=1\ W=1$, $R=1\ W=2$, and $R=2\ W=1$.

- The red triangle line represents $R=1\ W=1$.
- The green circle line represents $R=1\ W=2$.
- The blue square line represents $R=2\ W=1$.

The graph indicates that as time visibility increases, the probability of consistency also increases for all configurations. The specific data points and their corresponding consistency probabilities are shown in the graph, with a notable point at 10 ms highlighted.
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
99.9% consistent reads: 
R=2, W=1  
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

LNKD-DISK
The graph shows the relationship between $P(\text{consistency})$ and $t$-visibility (ms) for different values of $R$ and $W$. The legend indicates the following:

- Red triangle: $R=1$ and $W=1$
- Green circle: $R=1$ and $W=2$
- Blue square: $R=2$ and $W=1$

The label $N=3$ indicates the number of samples or data points used in the experiment.
N=3

LNNKD-SSD

P(consistency)

\[ \text{t-visibility (ms)} \]

2.0
N = 3

\[ P(\text{consistency}) \]

\[ 0.975 \]

\[ 2.0 \]
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

\( N=3 \)

Latency is combined read and write latency at 99.9th percentile.
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

59.5% faster

N=3
A Coordinator writes to a Replica. The Coordinator waits for \( W \) responses from the Replica, then waits for \( R \) responses after \( t \) seconds elapse. If a read arrives before the write, the response is considered stale. The Coordinator's critical factor in staleness is once per replica.
Write Latency (ms)

CDF

W=3

N=3
N=3

CDF

Write Latency (ms)

W=3

LNKD-SSD

LNKD-DISK

YMMR
The diagram shows the cumulative distribution function (CDF) of write latency for different storage systems. The y-axis represents the CDF, while the x-axis represents the write latency in milliseconds (ms). Two sets of data are compared: LNKD-SSD (green triangles) and LNKD-DISK (red circles). The data is represented for different widths (W) of 1, 2, and 3.

For W=3, the CDF for LNKD-SSD is shown with green triangles, indicating a lower latency compared to LNKD-DISK, represented with red circles. The notation N=3 is also present in the diagram, indicating the number of devices used in the test.
**Coordinator** once per replica

- write \((W)\)
- ack \((A)\)
- wait for \(W\) responses
- \(t\) seconds elapse
- read \((R)\)
- wait for \(R\) responses

**Replica**

- SSDs reduce variance compared to disks!
- response is stale if read arrives before write
- response is stale if read arrives before write
$N = 3$

![Graph showing $P(\text{consistency})$ vs. t-visibility (ms)](image_url)
$N=3$

The graph shows the relationship between $P(\text{consistency})$ and $t$-visibility (ms). The data points are categorized by different symbols and colors for different configurations marked as $R=1$ $W=1$, $R=1$ $W=2$, and $R=2$ $W=1$. The graph indicates a trend where $P(\text{consistency})$ increases as $t$-visibility increases.
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
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

N=3
How Eventual is Eventual Consistency?

PBS in action under Dynamo-style quorums

![Graph showing P(Consistency) over time after commit.]

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

**Operation Latency:** Exponentially Distributed CDFs

**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
Workflow

1. Tracing
2. Simulation
3. Tune $N, R, W$
4. Profit
[patch] Support consistency-latency prediction in nodetool

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.

https://issues.apache.org/jira/browse/CASSANDRA-4261
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)
PBS

**problem:** no guarantees with eventual consistency

**solution:** consistency prediction

**technique:** measure latencies use WARS model
consistency is a metric to predict
strong consistency
lower latency
WHAT IF I TOLD YOU
I CAN TELL YOU WHAT TO PICK
latency vs. consistency trade-offs
simple modeling with WARS
model staleness in time, versions
latency vs. consistency trade-offs
simple modeling with WARS
model staleness in time, versions

eventual consistency
often fast
often consistent

PBS helps explain when and why
latency vs. consistency trade-offs
simple modeling with WARS
model staleness in time, versions

eventual consistency
often fast
often consistent

PBS helps explain when and why

pbs.cs.berkeley.edu/#demo
@pbailis
VLDB 2012 early print
tinyurl.com/pbsvldb
cassandra patch
tinyurl.com/pbspatch
Extra Slides
Related Work
Quorum System Theory

e.g., Probabilistic Quorums

k-quorums

Deterministic Staleness

e.g., TACT/conits

FRACS
Consistency Verification

e.g., Golab et al. (PODC ’11), Bermbach and Tai (M4WSOC ’11)
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:

\[(\text{compensation cost}) \times (\# \text{ of expected anomalies})\]
Read only newer data?

\[(\text{monotonic reads session guarantee})\]

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

(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
\( N = 3 \)  
\((\text{LNKD-SSD and LNKD-DISK identical for reads})\)
Probabilistic quorum systems

\[ p_{\text{inconsistent}} = \frac{\binom{N-W}{R} \binom{R}{N-R}}{\binom{N-W}{R} \binom{R}{N-R}} \]
How consistent?

$k$-staleness: probability $p$ of reading one of last $k$ versions
How consistent?

$$1 - \left( \frac{N-W}{R_{\text{NR}}} \right)_K$$
How consistent?

\[ 1 - \left( \frac{N-W}{R} \right)^K \]
\langle k, t \rangle$-staleness: versions and time
\( \langle k, t \rangle \)-staleness: versions and time approximation: exponentiate \( t \)-staleness by \( k \)
“strong” consistency $\equiv$ reads return the last written value or newer (defined w.r.t. real time, when the read started)
Write to $W$, read from $R$ replicas
Write to $W$, read from $R$ replicas

quorum system:
guaranteed intersection

$N = 3$ replicas

$R \cap W = 3$ replicas

$R \cap W = 2$ 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
Synthetic, Exponential Distributions

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

ARS$\lambda:W\lambda$

- $1:4$
- $1:2$
- $1:1$
- $1:0.50$
- $1:0.20$
- $1:0.10$

$N=3$, $W=1$, $R=1$
Synthetic, Exponential Distributions

$\lambda:W$ 1/4x ARS

$N=3, W=1, R=1$
Synthetic, Exponential Distributions

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

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

$\lambda: W_\lambda$
- $1:4$
- $1:2$
- $1:1$
- $1:0.5$
- $1:0.2$
- $1:0.1$

$W 1/4 \times ARS$
$W 10 \times ARS$
concurrent writes: deterministically choose

Coordinator

(“key”, 1) (“key”, 2)

R=2
<table>
<thead>
<tr>
<th>%ile</th>
<th>Latency (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>15,000 RPM SAS Disk</strong></td>
<td></td>
</tr>
<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>
<tr>
<td><strong>Commodity SSD</strong></td>
<td></td>
</tr>
<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>
</tbody>
</table>

Table 1: LinkedIn Voldemort single-node production latencies.
<table>
<thead>
<tr>
<th>%ile</th>
<th>Read Latency (ms)</th>
<th>Write Latency (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>1.55</td>
<td>1.68</td>
</tr>
<tr>
<td>50</td>
<td>3.75</td>
<td>5.73</td>
</tr>
<tr>
<td>75</td>
<td>4.17</td>
<td>6.50</td>
</tr>
<tr>
<td>95</td>
<td>5.2</td>
<td>8.48</td>
</tr>
<tr>
<td>98</td>
<td>6.045</td>
<td>10.36</td>
</tr>
<tr>
<td>99</td>
<td>6.59</td>
<td>131.73</td>
</tr>
<tr>
<td>99.9</td>
<td>32.89</td>
<td>435.83</td>
</tr>
<tr>
<td>Max</td>
<td>2979.85</td>
<td>4465.28</td>
</tr>
<tr>
<td>Mean</td>
<td>9.23</td>
<td>8.62</td>
</tr>
<tr>
<td>Std. Dev.</td>
<td>83.93</td>
<td>26.10</td>
</tr>
<tr>
<td>Mean Rate</td>
<td>718.18 gets/s</td>
<td>45.65 puts/s</td>
</tr>
</tbody>
</table>

Table 2: Yammer Riak $N=3$, $R=2$, $W=2$ production latencies.
<table>
<thead>
<tr>
<th></th>
<th>Distribution Fit</th>
<th>Parameters</th>
<th>N-RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LNKD-SSD</strong></td>
<td>W = A = R = S:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>91.22%: Pareto, $x_m = 0.235$, $\alpha = 10$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>8.78%: Exponential, $\lambda = 1.66$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>N-RMSE: .55%</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>LNKD-DISK</strong></td>
<td>W:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>38%: Pareto, $x_m = 1.05$, $\alpha = 1.51$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>62%: Exponential, $\lambda = 0.183$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>N-RMSE: .26%</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>YMMR</strong></td>
<td>W:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>93.9%: Pareto, $x_m = 3$, $\alpha = 3.35$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>6.1%: Exponential, $\lambda = 0.0028$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>N-RMSE: 1.84%</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A = R = S: LNKD-SSD</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>YMMR</strong></td>
<td>A = R = S:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>98.2%: Pareto, $x_m = 1.5$, $\alpha = 3.8$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.8%: Exponential, $\lambda = 0.0217$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>N-RMSE: .06%</td>
<td></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></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></td>
<td>151.31</td>
<td>167.72</td>
<td>0</td>
<td>153.86</td>
<td>55.19</td>
<td>0</td>
<td>3.44</td>
</tr>
</tbody>
</table>