How Often Do Nodes Fail?
How Often Do Nodes Fail?

Example: disk drives

Disk Failures in the Real World: What Does an MTTF of 1,000,000 Hours Mean to You? Schroeder & Gibson, USENIX FAST 2007

Typical drive replacement rate is 2–4% annually.

1 PB = 1,000 × 1 TB drives ~ 20–40 dead drives annually → A failure every couple of weeks
Replication Model

Reliable network, fail-stop nodes

Relation

Node 1

Node 2

Node 3

Tuple
Replication Model

Assume single fragment (for now)
→ Data replication increases availability
Outline

Basic replication algorithms

Improved algorithms

Multiple fragments
Basic Replication

Simple concurrency control solution
Treat each copy as an independent data item

Example
Object X has 3 copies $X_1$, $X_2$, and $X_3$
Basic Replication

read(X)
Get shared $X_1$ lock
Get shared $X_2$ lock
Get shared $X_3$ lock
Read one of $X_1, X_2, X_3$
At the end of the transaction, release $X_1, X_2, X_3$ locks
Basic Replication

write($X$)
Get exclusive $X_1$ lock
Get exclusive $X_2$ lock
Get exclusive $X_3$ lock
Write new value into $X_1, X_2, X_3$
At the end of the transaction, release $X_1, X_2, X_3$ locks

→ Read lock all, write lock all replication: **RAWA**
Basic Replication

Correctness OK
2PL $\Rightarrow$ serializability
2PC $\Rightarrow$ atomic transactions

Problem
Low availability

Node is down

X is not accessible
Basic Replication

**Improved solution**
Readers lock and access a single copy
Writers lock and update all copies

→ Read lock one, write lock all replication: **ROWA**

Good availability for reads
Poor availability for writes
Basic Replication

Reminder
Using standard 2PL
Using standard commit protocols
Primary Copy Replication

Select primary node for $X$ (static)
Readers lock and access primary copy
Writers lock primary copy and update all copies

→ Read lock primary, write lock primary: RPWP
Primary Copy Replication

Local commit (RPWP-LC)

write(X)
Get exclusive $X_1$ (primary) lock
Write new value into $X_1$
Commit at primary, get sequence number for transaction
Perform $X_2$ and $X_3$ updates in sequence number order
Primary Copy Replication

Example

\[
\begin{array}{c|c|c|c|c|c}
  & t = 0 & & & & \\
\hline
X_1 & 0 & & & & \\
Y_1 & 0 & & & & \\
Z_1 & 0 & & & & \\
\hline
X_2 & 0 & & & & \\
Y_2 & 0 & & & & \\
Z_2 & 0 & & & & \\
\end{array}
\]

\[
\begin{align*}
T_1 & \quad X \leftarrow 1; Y \leftarrow 1 \\
T_2 & \quad Y \leftarrow 2 \\
T_3 & \quad Z \leftarrow 3
\end{align*}
\]
Primary Copy Replication

Example

t = 1

\[
\begin{array}{c|c}
X_1 & 1 \\
Y_1 & 1 \\
Z_1 & 3 \\
\end{array}
\quad
\begin{array}{c|c}
X_2 & 0 \\
Y_2 & 0 \\
Z_2 & 0 \\
\end{array}
\]

T_1 \quad X \leftarrow 1; Y \leftarrow 1 \quad \text{ Active at node 1 }

T_2 \quad Y \leftarrow 2 \quad \text{ Waiting for lock at node 1 }

T_3 \quad Z \leftarrow 3 \quad \text{ Active at node 1 }
Primary Copy Replication

Example

t = 2

X_1 1
Y_1 2
Z_1 3

X_2 1
Y_2 1
Z_2 3

T_1 X \gets 1; Y \gets 1 \rightarrow Committed
T_2 Y \gets 2 \rightarrow Active at node 1
T_3 Z \gets 3 \rightarrow Committed

#1: Z \gets 3
#2: X \gets 1; Y \gets 1
Primary Copy Replication

Example

t = 3

\[
\begin{array}{c|c}
X_1 & 1 \\
Y_1 & 2 \\
Z_1 & 3 \\
\end{array}
\]

\[
\begin{array}{c|c}
X_2 & 1 \\
Y_2 & 2 \\
Z_2 & 3 \\
\end{array}
\]

T_1: X ← 1; Y ← 1  \quad \text{Committed}
T_2: Y ← 2  \quad \text{Committed}
T_3: Z ← 3  \quad \text{Committed}

#3: Y ← 2
Primary Copy Replication

What good is RPWP-LC?

```
Primary | Backup | Backup
```

- Updates

Can’t read!
Primary Copy Replication

What good is RPWP-LC?

Can read *out-of-date* backup copy
→ Could be handy, more about it later

Can’t read!
Primary Copy Replication

Distributed commit (RPWP-DC)

\( \text{write}(X) \)
Get exclusive \( X_1 \) (primary) lock
Compute new value for \( X_1 \)
Prepare to write new value at all nodes
When all nodes are prepared, write new value and commit
Basic Replication

Read lock all, write lock all (RAWA)
Read lock one, write lock all (ROWA)
Read and write lock primary (RPWP)

   Local commit (LC)
   Distributed commit (DC)
## Basic Replication

### Comparison

<table>
<thead>
<tr>
<th></th>
<th>Probability that can read</th>
<th>Probability that can write</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAWA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROWA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RPWP-LC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RPWP-DC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

N – number of nodes with copies
P – probability that a node is operational
## Basic Replication

### Comparison

<table>
<thead>
<tr>
<th></th>
<th>Probability that can read</th>
<th>Probability that can write</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAWA</td>
<td>$P^N$</td>
<td>$P^N$</td>
</tr>
<tr>
<td>ROWA</td>
<td>$1 - (1 - P)^N$</td>
<td>$P^N$</td>
</tr>
<tr>
<td>RPWP-LC</td>
<td>$P$</td>
<td>$P$</td>
</tr>
<tr>
<td>RPWP-DC</td>
<td>$P$</td>
<td>$P^N$</td>
</tr>
</tbody>
</table>

$N$ – number of nodes with copies  
$P$ – probability that a node is operational
## Basic Replication

### Comparison

<table>
<thead>
<tr>
<th></th>
<th>Probability that can read</th>
<th>Probability that can write</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAWA</td>
<td>0.9510</td>
<td>0.9510</td>
</tr>
<tr>
<td>ROWA</td>
<td>~1.0000</td>
<td>0.9510</td>
</tr>
<tr>
<td>RPWP-LC</td>
<td>0.9900</td>
<td>0.9900</td>
</tr>
<tr>
<td>RPWP-DC</td>
<td>0.9900</td>
<td>0.9510</td>
</tr>
</tbody>
</table>

N = number of nodes with copies  
P = probability that a node is operational

N = 5  
P = 0.99
## Basic Replication

### Comparison

<table>
<thead>
<tr>
<th></th>
<th>Probability that can read</th>
<th>Probability that can write</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAWA</td>
<td>0.5905</td>
<td>0.5905</td>
</tr>
<tr>
<td>ROWA</td>
<td>~1.0000</td>
<td>0.5905</td>
</tr>
<tr>
<td>RPWP-LC</td>
<td>0.9000</td>
<td>0.9000</td>
</tr>
<tr>
<td>RPWP-DC</td>
<td>0.9000</td>
<td>0.5905</td>
</tr>
</tbody>
</table>

- **N** – number of nodes with copies
- **P** – probability that a node is operational

N = 5
P = 0.90
# Basic Replication

## Comparison

<table>
<thead>
<tr>
<th></th>
<th>Probability that can read</th>
<th>Probability that can write</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAWA</td>
<td>0.9510</td>
<td>0.9510</td>
</tr>
<tr>
<td>ROWA</td>
<td>~1.0000</td>
<td>0.9510</td>
</tr>
<tr>
<td>RPWP-LC</td>
<td>0.9900</td>
<td>0.9900</td>
</tr>
<tr>
<td>RPWP-DC</td>
<td>0.9900</td>
<td>0.9510</td>
</tr>
</tbody>
</table>

- **N** – number of nodes with copies
- **P** – probability that a node is operational

\[ N = 5 \quad P = 0.99 \]
Basic Replication

Comparison

<table>
<thead>
<tr>
<th></th>
<th>Probability that can read</th>
<th>Probability that can write</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAWA</td>
<td>0.3660</td>
<td>0.3660</td>
</tr>
<tr>
<td>ROWA</td>
<td>~1.0000</td>
<td>0.3660</td>
</tr>
<tr>
<td>RPWP-LC</td>
<td>0.9900</td>
<td>0.9900</td>
</tr>
<tr>
<td>RPWP-DC</td>
<td>0.9900</td>
<td>0.3660</td>
</tr>
</tbody>
</table>

N = 100
P = 0.99

N – number of nodes with copies
P – probability that a node is operational
Outline

Basic replication algorithms ✔

Improved algorithms
  Mobile primary
  Available copies

Multiple fragments
Mobile Primary

Improvement over RPWP

General approach
1. Elect new primary
2. Ensure new primary has all previously committed transactions
3. Resolve pending transactions
4. Resume processing
Mobile Primary

1. Election

Can be tricky

**One idea**
Nodes have IDs
Largest ID wins
Mobile Primary

1. Election

Algorithm for each node
- Broadcast proposal to become primary along with own ID
- Wait long enough so anyone with larger ID can stop takeover
- If received proposal with smaller ID, kill that takeover
- After wait without seeing a larger ID declare self as new primary
Mobile Primary

1. Election

It is useful to attach an *epoch number* to messages
Avoids confusion if stale messages linger
An epoch starts with the election of a new primary

Epoch: 1  Primary: \(N_3\)
Epoch: 2  Primary: \(N_5\)

\(P[E=3, ID=4], P[E=2, ID=5], P[E=3, ID=2] \implies D[E=3, ID=4]\)

Epoch: 3  Primary: \(N_4\)
Mobile Primary

2. Ensure new primary has previously committed transactions

E.g., assume RPWP-LC

**Diagram:**
- **Primary**
  - Committed $T_1, T_2$
- **Next primary**
  - May need to get and apply $T_1, T_2$
- **Backup**
Mobile Primary

3. Resolve pending transactions

E.g., assume RPWP-DC with 3PC

- $T_3$ in $?$ state
- $T_3$ in $W$ state
- $T_3$ in $W$ state
Mobile Primary

Bad node-failure scenario

Primary

\[ X_1 \]

Commits T\(_1\)

Backup

\[ X_2 \]

Backup

\[ X_3 \]

Primary

\[ X_1 \]

\[ X_2 \]

\[ X_3 \]

Commits T\(_2\) (unaware of T\(_1\))
Mobile Primary

Bad node-failure scenario
RPWP-DC with 3PC takes care of the problem

1. Finish T
2. Send data
3. Get acks
4. Prepare
5. Get acks
6. Commit

Primary  Backup 1  Backup 2

Time
Mobile Primary

Node recovery

All transactions have commit sequence number

Active nodes save updates as long as necessary
  E.g., since last checkpoint spanning all nodes

Recovering node asks active primary for missed updates and applies them in order
Mobile Primary

Majority commit example

<table>
<thead>
<tr>
<th>State</th>
<th>X₁</th>
<th>X₂</th>
<th>X₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>T₁, T₂, T₃</td>
<td>T₁, T₂</td>
<td>T₁, T₃</td>
</tr>
<tr>
<td>PC</td>
<td></td>
<td>T₃</td>
<td>T₂</td>
</tr>
<tr>
<td>W</td>
<td>T₄</td>
<td>T₄</td>
<td>T₄</td>
</tr>
</tbody>
</table>
Mobile Primary

Majority commit example

$X_1$ fails
$X_2$ new primary
$X_2$ commits $T_1$, $T_2$, $T_3$ and aborts $T_4$
$X_2$ resumes processing
$X_2$ commits $T_5$, $T_6$
$X_1$ recovers and asks $X_2$ for latest state
$X_2$ sends committed and pending transactions
$X_2$ involves $X_1$ in any future transactions
Mobile Primary

RPWP-DC guarantee
After transaction $T$ commits at current primary, any future primary will see $T$

\[ T_1, T_2, T_3 \]

\[ T_1, T_2, T_3, T_4 \]

Time
Mobile Primary

RPWP-DC performance hit

3PC is very expensive
  Many messages
  Locks held longer $\Rightarrow$ less concurrency

Could use 2PC instead
  May be blocking
  2PC is still expensive
Mobile Primary

Alternative: RPWP-LC
Commit transactions unilaterally at primary
Send updates to backups as soon as possible

(1) Finish T
(2) Commit T
(3) Send data
(4) Get acks
(5) Purge data

Primary
Backup 1
Backup 2

Time
Mobile Primary

Lost transactions
May happen with RPWP-LC
Mobile Primary

Lost transactions
Claim: the problem is tolerable
   Failures are rare
   Only a few transactions are lost
Mobile Primary

Lost transactions
Primary recovery: need to *compensate* for transaction

```
<table>
<thead>
<tr>
<th>Primary</th>
<th>Next primary</th>
<th>Backup 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_1, T_2, T_3$</td>
<td>$T_1, T_4, T_5$</td>
<td>$T_1, T_4$</td>
</tr>
<tr>
<td>Backup 3</td>
<td>Next primary</td>
<td>Backup 2</td>
</tr>
<tr>
<td>$T_1, T_2, T_3$</td>
<td>$T_1, T_4, T_5$</td>
<td>$T_1, T_4, T_5$</td>
</tr>
<tr>
<td>$T_3^{-1}, T_2^{-1}, T_4, T_5$</td>
<td>compensation</td>
<td></td>
</tr>
</tbody>
</table>
```

Time
Outline

Basic replication algorithms ✔

Improved algorithms
  Mobile primary ✔
  Available copies

Multiple fragments
Available Copies

Locks

Transactions write lock at **all available copies**
Transactions read lock at **any available** copy
Primary site (static) manages set of available copies \( \bigcup \)
Available Copies

Updates
1. Get $U$ from primary
2. Get write locks at $U$ nodes
3. Commit at $U$ nodes

$U = \{ X_1, X_2 \}$

$T_3 \oplus U = \{ X_1, X_2 \}$
Available Copies

Potential problem

\[ U = \{ X_1, X_2 \} \]

- **Primary**: \( X_1 \)
- **Recovering**: \( X_3 \)

Initiate recovery

\[ T_3 @ U = \{ X_1, X_2 \} \]
Available Copies

Potential problem

\[ U = \{ X_1, X_2, X_3 \} \]

Recover \( T_1, T_2 \)

\[ T_3 @ U = \{ X_1, X_2 \} \]
Available Copies

Solution

Initially for transaction $T$ get copy $U_T$ of $U$ from primary
Can use cached value instead

At commit of $T$, compare $U_T$ with current $U$ at primary
   If different, abort $T$
Available Copies

Solution

\[ U = \{ X_1, X_2 \} \]
\[ U = \{ X_1, X_2, X_3 \} \]

- Recover \( T_1, T_2 \)
- Initiate recovery

\[ X_1 \]
\[ \text{Reject} \]
\[ \text{Prepare} \]

\[ X_2 \]
\[ \text{Prepare} \]

\[ X_3 \]

\[ T_3 @ U = \{ X_1, X_2 \} \]
Available Copies

No primary
Let all nodes have a copy of $U$ (not just primary)
To modify $U$, run a special *atomic* transaction at all available sites
  Use commit protocol

E.g., $U_1 = \{ X_1, X_2 \} \rightarrow U_2 = \{ X_1, X_2, X_3 \}$
$X_3$ initiates transaction, but only $X_1, X_2$ participate

E.g., $U_2 = \{ X_1, X_2, X_3 \} \rightarrow U_3 = \{ X_1, X_2 \}$
Only $X_1, X_2$ participate in this transaction
Who initiates?
Available Copies

No primary
Can get tricky
What if the $U$-update transaction blocks?
How much update information must be remembered by whom?

<table>
<thead>
<tr>
<th></th>
<th>Committed</th>
<th>Pending</th>
<th>Recovering</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A, B, C, D, E, F</td>
<td>G</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A, B</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A, C, B, E, D</td>
<td>F</td>
<td>G, H</td>
</tr>
</tbody>
</table>
Outline

Basic replication algorithms ✔

Improved algorithms ✔
  Mobile primary
  Available copies

Multiple fragments
Correctness

$S_1: r_1[X_1] \rightarrow r_2[X_2] \rightarrow w_1[X_1] \rightarrow w_2[X_2]$

Is this schedule serializable?
Correctness

\[ X_1 \rightarrow X_2 \]

\[ S_1: r_1[X_1] \rightarrow r_2[X_2] \rightarrow w_1[X_1] \rightarrow w_2[X_2] \]

Is this schedule serializable?

One idea: require transactions to update all copies

\[ S_1: r_1[X_1] \rightarrow r_2[X_2] \rightarrow w_1[X_1] \rightarrow w_2[X_2] \rightarrow w_1[X_2] \rightarrow w_2[X_1] \]
Correctness

$S_1: r_1[X_1] \rightarrow r_2[X_2] \rightarrow w_1[X_1] \rightarrow w_2[X_2]$

Is this schedule serializable?

One idea: require transactions to update all copies
$S_1: r_1[X_1] \rightarrow r_2[X_2] \rightarrow w_1[X_1] \rightarrow w_2[X_2] \rightarrow w_1[X_2] \rightarrow w_2[X_1]$

→ Not a good idea for high-availability replication
Correctness

\[ X_1 \quad X_2 \]

\[ S_1: r_1[ X_1 ] \rightarrow r_2[ X_2 ] \rightarrow w_1[ X_1 ] \rightarrow w_2[ X_2 ] \]

Is this schedule serializable?

Another idea: build copy-semantics into the notion of serializability
Correctness

One-copy serializability (1SR)

A schedule $S$ on replicated data is 1SR if it is equivalent to a serial history of the same transactions on a one-copy database.
Correctness

Checking for 1SR

1. Treat $r_i[X_j]$ as $r_i[X]$ and $w_i[X_j]$ as $w_i[X]$ for all $X_j$ copies of $X$
2. Compute $P(S)$
3. If $P(S)$ acyclic then $S$ is 1SR
Correctness

Example 1

\[ S_1 \quad r_1[X_1] \rightarrow r_2[X_2] \rightarrow w_1[X_1] \rightarrow w_2[X_2] \]

\[ S_1' \quad r_1[X] \rightarrow r_2[X] \rightarrow w_1[X] \rightarrow w_2[X] \]

\[ S_1 \text{ is not 1SR} \]
Correctness

Example 2

\[ S_2 \quad r_1[X_1] \rightarrow w_1[X_1] \rightarrow w_1[X_2] \]
\[ \quad r_2[X_1] \rightarrow w_2[X_1] \rightarrow w_2[X_2] \]

\[ S_2' \quad r_1[X] \rightarrow w_1[X] \rightarrow w_1[X] \]
\[ \quad r_2[X] \rightarrow w_2[X] \rightarrow w_2[X] \]

\[ P(S_2) \quad T_1 \rightarrow T_2 \]

\[ S_2 \text{ is 1SR} \]
Correctness

Example 2

\[
S_2 \quad r_1[X_1] \rightarrow w_1[X_1] \rightarrow w_1[X_2] \\
\quad \quad \downarrow \\
\quad r_2[X_1] \rightarrow w_2[X_1] \rightarrow w_2[X_2]
\]

\[
S_2' \quad r_1[X] \rightarrow w_1[X] \rightarrow w_1[X] \\
\quad \quad \downarrow \\
\quad r_2[X] \rightarrow w_2[X] \rightarrow w_2[X]
\]

Equivalent serial schedule

\[
S_2'' \quad r_1[X] \rightarrow w_1[X] \\
\quad \quad \downarrow \\
\quad r_2[X] \rightarrow w_2[X]
\]
Correctness

Example 3

\[ S_3 \quad r_1[X_1] \rightarrow w_1[X_1] \rightarrow w_1[X_2] \]
\[ \quad \downarrow \]
\[ \quad r_2[X_1] \rightarrow w_2[X_1] \]

Is this a good schedule?
Correctness

Example 3

\[ S_3 \quad r_1[X_1] \rightarrow w_1[X_1] \rightarrow w_1[X_2] \]
\[ \quad \downarrow \]
\[ \quad r_2[X_1] \rightarrow w_2[X_1] \]

\[ S_3' \quad r_1[X] \rightarrow w_1[X] \rightarrow w_1[X] \]
\[ \quad \downarrow \]
\[ \quad r_2[X] \rightarrow w_2[X] \]

For \( S_3 \) to be a valid schedule, we need the precedence edge between \( w_1[X] \) and \( w_2[X] \).
Correctness

Example 3
We need to know how $w_2[ X_2 ]$ is resolved

$S_3 \quad r_1[ X_1 ] \rightarrow w_1[ X_1 ] \rightarrow w_1[ X_2 ]$

$\downarrow$  

$\rightarrow$  

$w_2[ X_2 ] \quad ✓$

$r_2[ X_1 ] \rightarrow w_2[ X_1 ]$

$S_3 \quad r_1[ X_1 ] \rightarrow w_1[ X_1 ] \rightarrow w_1[ X_2 ]$

$\downarrow$  

$\rightarrow$  

$w_2[ X_2 ] \quad ×$

$r_2[ X_1 ] \rightarrow w_2[ X_1 ]$
Correctness

Example 3
When $w_2[ X_2 ]$ is missing because $X_2$ is down, during recovery $X_2$ will have to perform $w_2[ X_2 ]$ in the correct order

\[
S_3 \quad r_1[X] \rightarrow w_1[X] \\
\quad \downarrow \\
\quad r_2[X] \rightarrow w_2[X]
\]
Multiple Fragments

A transaction spanning multiple fragments must
Follow locking rules for each fragment
Commit with *majority in each* fragment
Multiple Fragments

Must be careful with update transactions that read but do not modify a fragment

Read lock on $F_1$ at $X_1$

Read lock on $F_2$ at $Y_1$
Multiple Fragments

Assume $X_1, Y_1$ fail

- $F_1$
- $X_1$
- $X_2$
- $T_2$

Write to $F_1$ at $X_2$
Commit on $F_1$ at $X_2$

- $Y_1$
- $Y_2$
- $T_1$

Write to $F_2$ at $Y_2$
Commit on $F_2$ at $Y_2$

$\square$ Node down
Multiple Fragments

Equivalent schedule not serializable
\[ r_1[X] \rightarrow r_2[Y] \rightarrow w_1[Y] \rightarrow w_2[X] \]

Solution
Commit at read fragments/nodes too
Multiple Fragments

E.g., using available copies

Cannot commit on $F_2$ because $T_2 \preceq U = \{ F_2: \{ Y_1, Y_2 \}, F_1: \{ X_2 \} \}$ is out of date for $F_2$

Cannot commit on $F_1$ because $T_1 \preceq U = \{ F_1: \{ X_1, X_2 \}, F_2: \{ Y_2 \} \}$ is out of date for $F_1$

Write to $F_1$ at $X_2$

Commit on $F_1$ at $X_2$
Summary

RAWA, ROWA
Primary copy
  Static primary
  Mobile primary
Available copies
Correctness
Multiple fragments