Data Replication for Fault Tolerance

- Identical copies of data held at multiple nodes in a distributed system
- Improved performance and fault-tolerance
- Data replicates must be kept consistent despite failures in the system
- **Example** - five copies in five nodes:
  - If A is disconnected and a write updates the copy in A - the rest no longer consistent with A
  - Any read of their data will result in stale data
  - How many copies should we read (write)?
Simple Voting Scheme - Non Hierarchical

♦ Assign $v_i$ votes to copy $i$ of the data
♦ $S$ - set of all nodes with copies of the data
♦ $V$ - sum of all votes - $V = \sum_{i \in S} V_i$
♦ $r, w$ - variables such that $r + w > V ; w > V/2$
♦ $V(X)$ - total number of votes assigned to copies in set $X$ - $V(X) = \sum_{i \in X} V_i$

♦ Strategy ensuring that all reads use the latest data
♦ To complete a read - read nodes of a set $R \subseteq S$ such that $V(R) \geq r$
♦ To complete a write - write on every node of a set $W \subseteq S$ such that $V(W) \geq w$

Procedure Justification

♦ A set $R$ such that $V(R) \geq r$ is called a read quorum
♦ A set $W$ such that $V(W) \geq w$ is called a write quorum
♦ For any sets $R$ and $W$ such that $V(R) \geq r$ and $V(W) \geq w$ and $R \cap W \neq \emptyset$ (since $r + w > V$)
♦ Any read operation is guaranteed to read the value of at least one copy which has been updated by the latest write
♦ Furthermore - for any two sets $W_1, W_2$ such that $V(W_1), V(W_2) \geq w$
  $W_1 \cap W_2 \neq \emptyset$
Example

♦ One vote to each node
♦ The sum of all votes - \( V = 5 \)
♦ \( w > \frac{5}{2} \); \( r > 5 - w \)
♦ Permissible combinations for \((r,w)\)
  (1, 5), (2,5), (3,5), (4, 5), (5,5),
  (2, 4), (3,4), (4,4), (5,4), (3,3)

Example - Cont.

♦ Consider \((r,w) = (1, 5)\) - a read is possible from any one of the five copies; a write must update every one of the five copies
♦ Every read operation gets the most up-to-date data
♦ If \( w=5 \), selecting \( r>1 \) slows down the
♦ If node A gets disconnected - we can still read from each node but not update all nodes
♦ Consider \((r,w) = (3,3)\) - less copies to write (only 3), but read takes longer than if \((r,w) = (1, 5)\)
♦ If node A gets disconnected - read or write into A is impossible, but the remaining four nodes can continue to read and write as usual
Performance vs. Availability

♦ Selected values of $r$ and $w$ affect the system performance and availability

♦ If there are many more reads than writes - we choose a low $r$ to speed up the read operations

♦ $r=1$ requires $w=5$ - write can not be done if even one node is disconnected

♦ Selecting $r=2$ allows $w=4$ and write can still be done if four out of the five nodes are connected

♦ Trade-off between performance and availability

Reliability and Availability

Markov Models

♦ Assumptions:
  * Failures occur at each node according to a Poisson process with rate $\lambda$ (links do not fail)
  * When a node fails, it is repaired and up-to-date data is loaded
  * Repair time is an exponentially distributed random variable with mean $1/\mu$

♦ Example: $(r,w) = (3,3)$ - both read and write operations can take place if at least three of the five nodes are up

♦ To compute reliability and availability, we use Markov Chain models
Reliability and Long-Term Availability for \((r,w)=(3,3)\)

- **Markov chain for reliability:**
  - **State** - number of nodes down;
  - **F** - the failure state
  - **Reliability at time** \(t\)
    \[ R(t) = 1 - P_F(t) \]

- **Markov chain for availability:**
  - **State** - number of nodes down
  - **Long-Term Availability** = \(P_0 + P_1 + P_2\)
  - **Complete analysis in Exercises**

Vote Assignment for Maximizing Availability

- In general, nodes have different reliabilities and availabilities
- Links can fail as well
- **Point Availability** - the probability that at time \(t\) the system is up - read and write quorums exist
- **Problem**: Assigning votes to nodes to maximize point availability
- **Optimal assignment difficult** - heuristics necessary
- **Notations**: For some fixed point in time \(t\) (\(t\) is omitted for simplicity)
  - **Point Availability of node** \(i\) - \(an(i)\)
  - **Point Availability of link** \(j\) - \(al(j)\)
  - **L(i)** - set of links incident on node \(i\)
Heuristics for Vote Assignment

♦ Heuristic 1: set \( V(i) = an(i) \sum_{j \in L(i)} al(j) \) (rounded to the nearest integer)
If sum of votes is even, give an extra vote to one of the nodes with the maximum number of votes

♦ Heuristic 2:
\( k(i, j) \) - node connected to node \( i \) by link \( j \);
set \( V(i) = an(i) + \sum_{j \in L(i)} al(j) \cdot an(k(i, j)) \) (rounded to the nearest integer)
If sum is even - an extra vote is added as above

Heuristic 1 - Example
\( v(A) = \text{round}(0.7 \times 0.7) = 0 \)
\( v(B) = \text{round}(0.8 \times 1.8) = 1 \)
\( v(C) = \text{round}(0.9 \times 1.6) = 1 \)
\( v(D) = \text{round}(0.7 \times 0.9) = 1 \)

♦ Node \( A \) is unreliable compared to the rest - gets no votes
♦ Sum of votes is 3 - quorums must satisfy \( r + w > 3 \); \( w > 3/2 \) \( \Rightarrow \) \( w = 2 \) or 3
♦ If \( w=2 \) - \( r=2 \) is smallest read quorum
♦ Possible read (or write) quorums - \( BC, CD, BD \)
♦ If \( w=3 \) - \( r=1 \) is smallest read quorum
♦ Possible read quorums - \( B, C, D \)
♦ One write quorum: \( BCD \)
Heuristic 2 - Example

- Sum of votes even - B gets an extra vote
- Final vote assignment - v(A)=1, v(B)=3, v(C)=2, v(D)=1
- Sum of votes is 7 - read and write quorums must satisfy:
  - r + w > 7
  - w > 7/2
  - w = 4 or 5 or 6 or 7

Quorums for Heuristic 2

- Possible quorums for r+w=8
  - v(A)=1, v(B)=3, v(C)=2, v(D)=1

<table>
<thead>
<tr>
<th>r</th>
<th>w</th>
<th>Read Quorums</th>
<th>Write Quorums</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>4</td>
<td>AB, BC, BD, ACD</td>
<td>AB, BC, BD, ACD</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>B, AC, CD</td>
<td>BC, ABD</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>B, C, AD</td>
<td>ABC, BCD</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>A, B, C, D</td>
<td>ABCD</td>
</tr>
</tbody>
</table>

- Every (r,w) pair has an availability associated with it - the probability that at least one read and one write quorum exist despite node and/or link failures
- (r,w)=(4,4) - identical lists of read and write quorums
- Other (r,w) - lists are different
Calculating Availability for \((r,w)=(4,4)\)

- **Availability** \(A\) is the probability that at least one of the quorums \(AB, BC, BD, ACD\) can be used.
- Denote events: \(E_1, E_2, E_3, E_4\) - \(AB, BC, BD, ACD\) are up, respectively.
- Events are not mutually exclusive.
- \(A = P(E_1 \cup E_2 \cup E_3 \cup E_4)\)

Some calculations:
- \(P(E_1) = P(AB\text{ is up}) = 0.7 \times 0.7 \times 0.8 = 0.392\)
- \(P(E_2 \cap E_3) = P(BC\text{ and BD are up}) = 0.8 \times 0.9 \times 0.9 \times 0.2 \times 0.7 = 0.091\)

Exercise: Complete the calculation of the availability \(A\) for \((r,w)=(4,4)\)

Different Method for Availability Calculation

- System has 8 modules (4 nodes and 4 links) - each can be up or down.
- Total of \(2^8 = 256\) mutually exclusive states.
- Probability of each state is a product of 8 terms, either \(a(i)\) or \(1-a(i)\) or \(a(l)\) or \(1-a(l)\).
- Methodical (but long) way of computing availability - list all states and add up the probabilities of those where a quorum exists.
- For any other value of \((r,w)\) - read and write quorums are different.
- Availability - sum of probabilities of states in which both read and write quorums exist.
Dynamic Vote Assignment

- If repair is not fast enough - system can degrade
- If system degrades enough - no connected cluster with a majority of total votes exists

**Solution** - adjustable quorums instead of static ones
**Assumption** - each node has exactly one vote

For each data, version numbers are maintained - incremented with every update

This can only be executed if a write quorum can be gathered

---

Dynamic Vote Assignment - Notations

- \( VN_i \) - version number of data at node \( i \)
- \( SC_i \) - update sites cardinality at node \( i \) - number of nodes which participated in the \( VN_i \)-th update of this data

When system starts operation, \( SC_i \) is initialized to the total number of nodes in the system

- \( S_i \) - set of nodes with which node \( i \) can communicate
- \( M \) - maximum version number in \( S_i \)
- \( I \) - partial set of \( S_i \) with nodes whose version number is \( M \)
- \( N \) - maximum update sites cardinality (\( S_i \)) of nodes in \( I \)
Dynamic Vote Assignment Algorithm

1. If an update request arrives at node i, node i computes the following quantities:
   • $M = \max\{VN_j, j \in S_i\}$ where $S_i$ is the set of nodes with which node i can communicate, including itself, i.e., the maximum version number of the concerned datum, among all the nodes with which node i can communicate.
   • $I = \{j | VN_j = M, j \in S_i\}$, i.e., the set of all nodes whose version number is equal to the maximum.
   • $N = \max\{SC_j, j \in I\}$, i.e., the maximum update sites cardinality associated with all the nodes in I.

2. If $||I|| > N/2$, then node i can raise a write quorum and is allowed to carry out the update on all nodes in I; otherwise the update is not allowed. The update is carried out and the version number of each copy of that datum in I is incremented, i.e., $VN_i$ is incremented for each $i \in I$. Also, for each $i \in I$, we set $SC_i = ||I||$. This entire step must be done atomically: all these operations must be done at each node in I, or none of them can be done.

Dynamic Vote Assignment - Example

♦ Seven nodes - same data - state at time $t_0$

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>VN</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>SC</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
</tbody>
</table>

♦ Failure disconnects into two - {A,B,C,D} and {E,F,G}

♦ E receives an update request
   * $SC_E = 7$ - E must find more than 7/2 nodes (including itself)
   * can find only 3
   * update request is rejected

♦ A receives an update request
   * can be accepted
   * A,B,C,D are updated

♦ New state

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>VN</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>SC</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
</tbody>
</table>
Example – Cont.

♦ Another failure - components become {A,B,C}, {D}, {E,F,G}
♦ An update request arrives at C
  * write quorum at C is 3
  * update successful
♦ New state

\[
\begin{array}{ccccccc}
A & B & C & D & E & F & G \\
VN & 7 & 7 & 7 & 6 & 5 & 5 & 5 \\
SC & 3 & 3 & 3 & 4 & 7 & 7 & 7 \\
\end{array}
\]

Voting – Hierarchical Organization

♦ If V is large, \( r+w \) is large - data operations take a long time
♦ Possible solution - hierarchical voting scheme
♦ Construct an \( m \)-level tree
♦ All the nodes holding copies of the data are leaves at level \( m-1 \)
♦ Add virtual nodes at the higher levels up to the root at level \( 0 \) - added nodes are virtual groupings of the real nodes
♦ Each node at level \( i \) will have exactly \( L_{i+1} \) children
**Quorum Generation Algorithm**

- Assign one vote to each node in the tree
- Set Read quorum and write quorum sizes at level $i$, $r_i$ and $w_i$ so that: $r_i + w_i > L_i$; $w_i > L_i / 2$
- Following algorithm is used recursively:
  - Read-mark the root at level 0
  - At level 1 - read-mark $r_1$ nodes
  - Proceeding from level $i$ to level $i+1$
    - read-mark $r_{i+1}$ children of each of the nodes read-marked at level $i$
  - You cannot read-mark a node which does not have at least $r_{i+1}$ non-faulty children
  - Proceed until $i = m-1$
- The read-marked leaves form a read quorum
- Forming a write-quorum is similar

---

**Example - a Tree for Hierarchical Quorum Generation**

- $m = 3$
- $L_1 = L_2 = 3$
Algorithm - Example

- $w_i = 2$ for $i=1,2$, $r_i = L_i - w_i + 1 = 2$
- Starting at the root - read-mark two of its children - say X and Y
- Read-mark two children for X and Y - say A,B - for X, and D,E for Y
- Read quorum is the set of read-marked leaves - A, B, D, E

Example - cont.

- Suppose D is faulty - cannot be part of the read quorum
- We have to pick another child of Y - say F - to be in the read quorum
- If two of Y's children are faulty - we cannot read-mark Y - we have to backtrack and try read-marking Z instead
- Exercise: List read quorums generated by $r_1 = 1$, $w_1 = 3$, $r_2 = 2$, $w_2 = 2$
Hierarchical vs. Non-Hierarchical Approach

♦ Read quorum consists of just 4 copies
♦ Similarly, we can have a write quorum with 4 copies
♦ For the non-hierarchical approach with one vote per node, \( r + w > 9 ; w > 9/2 \)
♦ \( w \) is at least 5, compared to 4 in the tree approach
♦ To prove that the hierarchical approach works, we show that every possible read quorum has to intersect every possible write quorum in at least one node

Primary Backup Approach

♦ A node is designated as the primary - all accesses are through that node
♦ Other nodes are designated as backups
♦ Under normal operation - all writes to the primary are also copied to the functional backups
♦ When the primary fails - one of the backup nodes is chosen to take its place