Thus far, we have assumed that there is only a single copy of each data item.

This copy is placed at one of the sites, which is responsible for concurrency control and recovery for that data item.

However, for a data item that is accessed often from different sites, this could lead to a significant amount of communication.

Moreover, when a site fails, all data residing on that site becomes unavailable.
Replication

- To increase availability of data, and to reduce communication for remote data, data can be replicated.
- From the user’s point of view, replication (like distribution, physical and logical organization of data), should be transparent.
- I.e. the user should not be aware that some (or all) data items are replicated, and should see no difference in performance.
- The user can be a programmer or an end user.

1 Copy Serializability

- The correctness definition for replicated databases is therefore that it should behave as though all transactions are executed in a serial manner on a single copy database.
- This is the notion of one copy serializability, i.e. 1SR.
- The user must be given a one copy view of the database.
- How is this achieved?
- Read-only is easy. For writes we must manage carefully!
Write-All approach

• This is the obvious first solution:
  – Reads can be satisfied by any copy in the system,
  – Writes must all modify every copy of the data item being written.
• This is a very effective solution – it completely eliminates the problem of multiple copies, and gives each txn the correct view. HOWEVER
• It is very poor in terms of performance and progress:
  – Failures have a crippling effect on transactions!

Write-All-Available

• Allow a txn to proceed even though failures make it impossible to write all copies of the data.
• Allow the txn to simply write to every site that is available. Those that are down can be ignored.
• Thus some copies of the data may be out of sync, i.e. may not contain the latest updates.
Example

- Consider the following execution. Note that multiple copies are marked using the upper case subscripts.
  \[ w_0[x_A] w_0[x_B] w_0[y_C] c_0 \quad r_1[y_C] w_1[x_A] c_1 \quad r_2[x_B] w_2[y_C] c_2 \]
  
- \( T_2 \) reads copy \( x_B \) from \( T_0 \), even though it should have read from \( T_1 \).
  
- Thus the above history is not equivalent to \( T_0 T_1 T_2 \).

- Is it equivalent to some other serial one-copy history?
  
- \( \text{NO!} \) \( w_0[y_C] < r_1[y_C] < w_2[y_C] \), there is no other equivalent serial execution.

- This is interesting, because the execution actually seems to be a serial execution of the transactions!!!

Example (contd.)

- So what has gone wrong?
  
- The problem is that the write by \( T_1 \) into \( x \), did not update all copies of \( x - x_B \) in particular.

- This could only mean that site B must have been down when \( T_1 \) wrote \( x \), and must have recovered before \( T_2 \) read \( x \).

- I.e. the failures must have been as such:
  \[ w_0[x_A] w_0[x_B] w_0[y_C] c_0 \quad \text{fail}_B \quad w_1[x_A] c_1 \quad \text{Recover}_B \quad r_2[x_B] w_2[y_C] c_2 \]

- Thus the problem is that \( T_2 \) read a copy at a site that had failed and upon recovery did not re-sync with the other sites! Fixing this is still not enough!!
Assumptions

- Again, we will assume the same model for the database.
- The TM now maps all reads onto a read of some copy, and all writes onto a write on all (available) copies. It uses directories of copies to determine where copies are stored.
- Failures are assumed to be fail stop.
- We begin by ignoring communication failures.
- Thus a copy $x_A$ at a site $A$ is available to site $B$ if $A$ correctly executes each read/write of $x_A$ from site $B$, and $B$ receives the acknowledgement.
Assumptions.

• Therefore site failures are detectable.
• The timing of updating multiple copies can vary:
  – Immediate: as soon as the write is received.
  – Deferred: could delay the updating of copies. Update copies only upon commitment or abortion. Intentions lists can be piggybacked with VOTE_REQ msgs.
• Delayed updating results in
  – Fewer messages
  – Cheaper aborts
  – Delayed commitment
  – Delayed detection of conflicting operations. Can be solved by using a primary copy approach.

Replicated Data History.

• Let $h(\ )$ be a function that maps
  – $r_i[x] \rightarrow r_i[x_A]$ for some copy $x_A$ of $x$.
  – $w_i[x] \rightarrow w_i[x_{A1}], \ldots, w_i[x_{Am}]$, for some copies of $x$
  – $c_i \rightarrow c_i$
  – $a_i \rightarrow a_i$
Replicated Data History

• A complete replicated data (RD) history $H$ over $T$={$T_0$, $\ldots$, $T_n$} is a partial order with ordering relation $<$ where:
  – $H=h(U_{i=0..n} T_i)$ for some translation function $h$;
  – For each $T_i$ and all operations $p_i, q_i$ in $T_i$, if $p_i < q_i$, then every operation in $h(p_i)$ is related by $<$ to every operation in $h(q_i)$.
  – For every $r_j[x_A]$ there is at least one $w_i[x_A] < r_j[x_A]$.
  – All pairs of conflicting operations are related by $<$, where two operations conflict if they operate on the same copy and at least one is a write; and
  – If $w_i[x] < r_i[x]$ and $h(r_i[x]) = r_j[x_A]$ then $w_i[x_A]$ must be in $h(w_i[x])$.

Given txns {$T_0$, $T_1$, $T_2$, $T_3$}:

$T_0 = w_0[x] \rightarrow c_0$
$w_0[y] \rightarrow c_0$

$T_1 = r_1[x] \rightarrow w_1[x] \rightarrow c_1$

$T_2 = w_2[x] \rightarrow r_2[x] \rightarrow w_2[y] \rightarrow c_2$

$T_3 = r_3[x] \rightarrow r_3[y] \rightarrow c_3$

The following is an example of an RD history:
Reads-From Relationship

• Let $H$ be an RD history.
• Txn $T_j$ reads-$x$-from $T_i$ in $H$ if for some copy $x_A$, $T_j$ reads-$x_A$-from $T_i$, that is, if $w_i[x_A] < r_j[x_A]$ and no $w_k[x_A]$ $(k <> i)$ falls between these operations.
• Since reads-from are unique on copies, and a txn reads only one copy, then reads-from relationships on data items are unique too.

Serialization Graph

• Consider only complete histories with committed transactions only.
• I.e. we assume recoverable execution.
• What does that mean for replicated data?
• An RD history $H$, is recoverable if whenever $T_i$ reads (any copy) from $T_j$ in $H$ and $c_i$ is in $H$, then $c_j$ is in $H$ and $c_j < c_i$.
• The Serialization graph is generated as before, except that conflicting operations are now defined on copies rather than data items.
Serialization Graph

- Let $H$ be an RD history involving transaction $T_i$. If $SG(H)$ is acyclic and for some $x$, $w_i[x] < r_i[x]$, then $T_i$ reads-x-from $T_i$ in $H$.

- **Proof:**
  - From conditions (2) and (5) on RD histories, $w_i[x] < r_i[x]$ implies that for some copy $x_A$ of $x$, $w_i[x_A] < r_i[x_A]$.
  - Suppose, $T_i$ didn’t read $x$ from $T_i$ in $H$. Then there must exist some $w_k[x_A]$ ($k<>i$) in $H$ such that $w_i[x_A] < w_k[x_A] < r_i[x_A]$.
  - But then $SG(H)$ is acyclic.

Serializability

- **Acyclicity** of the serialization graph does NOT guarantee serializability for RD histories.
- A history is serializable if it is equivalent to a 1C history.
- The same order for conflicting operations does not work since the conflicting operation in the RD history and the 1C history are not the same.
- View equivalence is more natural for RD histories since the reads-from-relationships and final writes behave similarly in both types of histories.
RD history equivalence

- Given an RD history $H$, define $w_i[x_A]$ to be a final write for $x_A$ in $H$ if $a_i$ is not in $H$ and for all $w_j[x_A]$ in $H$ ($j <> i$), either $a_j$ is in $H$, or $w_j[x_A] < w_i[x_A]$.

- Two RD histories are equivalent if they are view equivalent, that is, they have the same reads-from relationships and final writes.

RD history equivalence

- An RD history $H$ over $T$ is equivalent to a 1C history $H_{1C}$ over $T$ if
  1. $H$ and $H_{1C}$ have the same reads-from relationships on data items (i.e., $T_j.reads-x-from T_i$ in $H$ iff the same holds in $H_{1C}$), and
  2. For each final write $w_i[x]$ in $H_{1C}$, $w_i[x_A]$ is a final write in $H$ for some copy $x_A$ of $x$.

An RD history is one-copy serializable (1SR) if it is equivalent to a serial 1C history.
Examples

- Is 1SR, it is equivalent to $T_0 \ T_2 \ T_1 \ T_3$.
- But, $w_0[x_A] \ w_0[x_B] \ w_0[y_C] \ c_0 \ r_1[y_C] \ w_1[x_A] \ c_1 \ r_2[x_B] \ w_2[y_C] \ c_2$ is not.
- However, it is a serial history!!
- Thus not every serial RD history is 1SR.

Final Writes

- Let $H$ be an RD history over $T$, with acyclic SG($H$). Let $H_{1C}$ be a serial 1C history over $T$ such that the order of transactions in $H_{1C}$ is consistent with SG($H$). If $w_i[x]$ is a final write for $x$ in $H_{1C}$, then every write, $w_i[x_A]$, by $T_i$ into some copy $x_A$ of $x$ is a final write for $x_A$ in $H$.

**Proof:**
- Suppose $w_i[x]$ is a final write for $x$ in $H_{1C}$. Let $w_i[x_A]$ be any write into $x$ by $T_i$ in $H$. If $w_i[x_A]$ is not a final write, then there is some $w_j[x_A]$ ($j \neq i$) such that $a_j$ is not in $H$ and $w_i[x_A] < w_j[x_A]$.
- Thus $T_i \rightarrow T_j$ is in SG($H$), so $T_j$ precedes $T_i$ in $H_{1C}$.
- $\Rightarrow a_j$ is not in $H_{1C}$ and $w_i[x] < w_j[x]$ in $H_{1C}$, contradicting the choice of $w_i[x]$ as a final write.
Serializability

• Thus we can ignore final writes – they must be the same.
• **Theorem:** Let $H$ be an RD history. If $H$ has the same reads-from relationships as a serial 1C history $H_{1C}$, where the order of transactions in $H_{1C}$ is consistent with $SG(H)$, then $H$ is 1SR.

---

Replicated Data
20 January, 2009
Prof. Sunil Prabhakar
Serializability

• Thus we can ignore final writes – they must be the same.

• **Theorem:** Let $H$ be an RD history. If $H$ has the same reads-from relationships as a serial 1C history $H_{1C}$, where the order of transactions in $H_{1C}$ is consistent with $\text{SG}(H)$, then $H$ is 1SR.

Graphs for 1SR histories

• How can we modify the serialization graphs to identify exactly the set of 1SR histories?

• The problem arises from the failure and recovery of sites:
  – A failed site will not be updated
  – Upon recovery it has inconsistent data.

• How can we capture the effects of these failures and recoveries in the serialization graph?
Example

\[ T_0 = \begin{align*}
  w_0[x] & \rightarrow c_0 \\
  w_0[y] & \rightarrow c_0
\end{align*} \]

\[ T_1 = \begin{align*}
  r_1[x] & \rightarrow w_1[y] \\
  r_1[x] & \rightarrow c_1
\end{align*} \]

\[ T_2 = \begin{align*}
  r_2[y] & \rightarrow w_2[x] \\
  r_2[y] & \rightarrow c_2
\end{align*} \]

The following RD history can occur with 2PL on copies:

\[ w_0[x_A] \]
\[ w_0[x_B] \]
\[ w_0[y_C] \]
\[ w_0[y_D] \]
\[ r_1[x_A] \]
\[ y_D \downarrow \]
\[ w_1[y_C] \]
\[ c_1 \]
\[ w_2[x_B] \]
\[ c_2 \]

This is not a 1SR history! But SG is acyclic:

The problem

- In the example there were no recoveries, thus by ensuring that a recovering site synchronizes before it is accessed, we would still have non-1SR histories!
- We are failing to capture the conflict at the item level by considering only conflicts at the copy level.
- Note that two conflicting operations must contain a write which must write all (available) copies. Without failures the conflict is detected.
Replicated Data SG

- Try to synchronize two transactions that access a conflicting item.
- Define: $n_j \prec n_k$, i.e., $n_i \ll n_k$, in a directed graph, if there is a path from $n_i$ to $n_k$.
- A replicated data serialization graph (RDSG) for $H$ is $\text{SG}(H)$ with enough edges added such that for all data items, $x$:
  1. If $T_i$ and $T_k$ write $x$, then either $T_i \ll T_k$ or $T_k \ll T_i$.
  2. If $T_j$ reads $x$ from $T_i$, $T_k$ writes some copy of $x$ ($k \not= i$, $k \not= j$), and $T_i \ll T_k$, then $T_j \ll T_k$.

RDSG

- A graph that satisfies condition 1 induces a write order for $H$.
- If it satisfies condition 2 it induces a read order for $H$.
- Given a history $H$, the $\text{RDSG}(H)$ is not unique.
- The write order ensures that every pair of txns that write into the same item (even if they don’t write the same copy).
- Write and read order ensure that every pair of txns that read and write the same item.
Example.

- The example enforces a write order.
- However it does not enforce a read order:
  - Since $T_1$ reads-$x$-from $T_0$, $T_2$ writes $x$, and $T_0 \rightarrow T_2$, we add $T_1 \rightarrow T_2$ to RDSG($H$);
  - Since $T_2$ reads-$y$-from $T_0$, $T_1$ writes $y$, and $T_0 \rightarrow T_1$, we add $T_2 \rightarrow T_1$ to the RDSG($H$).

\[ T_0 \xrightarrow{} T_2 \xrightarrow{} T_1 \]

- Now RDSG($H$) has a cycle, as required.

1SR

- **Theorem:** Let $H$ be an RD history. If $H$ has an acyclic RDSG, then $H$ is 1SR.
- **Proof:**
  - Let $H_s = T_{i1}, \ldots, T_{in}$ be a serial 1C history where $T_{i1}, \ldots, T_{in}$ is a topological sort of RDSG($H$).
  - Since RDSG($H$) contains SG($H$), $H$ is 1SR if $H$ and $H_s$ have the same reads-from relationships.
  - Assume that $T_j$ reads-$x$-from $T_i$ in $H$. Suppose, by way of contradiction, that $T_j$ reads-$x$-from $T_k$ in $H_s$.
  - If $k=j$, then $T_j$ must read-$x$-from $T_k$ in $H$ too since SG($H$) is acyclic $\Rightarrow k <> j$. 

31

32
Proof (cont)

– Since $T_j$ reads-x-from $T_i$ in $H$, $T_i \rightarrow T_j$ is in RDSG($H$), so $T_i$ precedes $T_j$ in $H_s$.
– Since the RDSG induces both a read and write order, we have that either $T_k << T_i$ or $T_j << T_k$.
– Thus either $T_k$ precedes $T_i$ (which precedes $T_j$) or $T_k$ follows $T_j$ in $H_s$, both contradict that $T_j$ reads-x-from $T_k$ in $H_s$.
– Now assume $T_j$ reads-x-from $T_i$ in $H_s$. By the definition of RD histories and the reads-from relationship, $T_j$ reads-x-from some txn in $H$, say $T_h$. By the above, $T_j$ reads-x-from $T_h$ in $H_s$. Since the reads-from relation is unique, $T_h=T_j$.

Atomicity of Failures and Recovery

• Another alternative, is to ensure that all transactions view failures and recoveries consistently.
• Atomicity of failure:

$w_0[x_A] \rightarrow r_1[x_A] \rightarrow y_D \downarrow \rightarrow w_1[y_C] \rightarrow c_1$
$w_0[x_B] \rightarrow r_2[y_B] \rightarrow x_A \downarrow \rightarrow w_2[x_B] \rightarrow c_2$

• $T_1$ sees the failures as: $yD\downarrow \rightarrow T_1 \rightarrow xA\downarrow$ but
• $T_2$ sees the failures as: $xA\downarrow \rightarrow T_2 \rightarrow yD\downarrow$
Atomicity of Failures

- We want all transactions to agree on when the failures occurred.
- There can be no serial ordering of the failures and $T_1$, $T_2$ that is consistent with the views of $T_1$ and $T_2$.
- We want to synchronize the recognition of failures of sites with the read and write operations that are taking place.
- Certain views of failures may be troublesome and should not be allowed.

Atomicity of Recoveries

- We require that each copy be initialized before it is read a copies txn can be used for this.
- After initialization, all txns need to be informed about the new copy so that they can write it too.
- This has to be done carefully:
Example

- The problem is that $T_2$ should have updated the new copy of $x$, $xB$.
- Since $T_1$ knew about $xB$, and executed before $T_2$.
- In terms of recoveries,
  - The view of $T_1$ is: $xB \uparrow \rightarrow T_1$
  - The view of $T_2$ is: $T_2 \rightarrow xB\uparrow$
  - Since $T_1$ executes before $T_2$, this is inconsistent!!
- We want all txns to have a consistent view of the recovery of copies.
Failure-Recovery SG

- Assume that once a copy fails, it never recovers!!
- Given an RD history $H$ over transactions $\{T_0, \ldots, T_n\}$, a failure-recovery serialization graph (FRSG) for $H$ is a directed graph with nodes $N$ and edges $E$ where:
  - $N = \{T_0, \ldots, T_n\} \cup \{create[x_A] \mid x$ is a data item, and $x_A$ is a copy of $x\} \cup \{fail[x_A]\}$
  - $E = \{T_i \rightarrow T_j \mid T_i, T_j$ is in $SG(H)\} \cup E1 \cup E2 \cup E3$, where:
    - $E1 = \{create[x_A] \rightarrow T_i \mid T_i$ reads or writes $x_A\}$;
    - $E2 = \{T_i \rightarrow fail[x_A] \mid T_i$ reads $x_A\}$;
    - $E3 = \{T_i \rightarrow create[x_A]$ or $fail[x_A] \rightarrow T_i \mid T_i$ writes some copy of $x$, but not $x_A\}$.

Example

- For the following RD history:

  - The following is a FRSG:
1SR

• **Theorem**: Let $H$ be an RD history. If $H$ has an acyclic FRSG, then $H$ is 1SR.

• **Proof**:
  - Let $H_s = T_{i1}, \ldots, T_{in}$ be a serial 1C history where $T_{i1}, \ldots, T_{in}$ is a topological sort of FRSG($H$).
  - Since FRSG($H$) contains SG($H$), $H$ is 1SR if $H$ and $H_s$ have the same reads-from relationships.
  - Assume that $T_i$ reads-$x_A$-from $T_j$ in $H$. Hence $T_i \rightarrow T_j$ is in FRSG($H$), and $T_i$ precedes $T_j$ in $H_s$.
  - Let $T_k$ be any other transaction that writes $x$.
  - If $T_k$ writes $x_A$, then since $T_j$ reads-$x_A$-from $T_i$ in $H$, either $T_k \rightarrow T_i$ or $T_j \rightarrow T_k$ must be in FRSG($H$).

**Proof (contd.)**

- If $T_k$ does not write $x_A$, by defn of FRSG, either $T_k \rightarrow create[x_A]$ or $fail[x_A] \rightarrow T_k$.
- In the former case, since $create[x_A] \rightarrow T_i$, $T_k$ precedes $T_i$ in FRSG($H$).
- In the latter case, since $T_j \rightarrow fail[x_A]$, $T_j$ precedes $T_k$ in the FRSG($H$).
- Hence, if $T_k$ writes $x$, either $T_k$ precedes $T_i$ or follows $T_j$ in the FRSG and $H_s$.
- Thus $T_j$ reads-$x$-from $T_i$ in $H_s$.
- Now, suppose $T_j$ reads-$x$-from $T_i$ in $H_s$. By the defn of RD history, $T_j$ reads-$x$-from some txn in $H$, say $T_h$. By the above, $T_j$ reads-$x$-from $T_h$ in $H_s$. Since reads from relationships are unique, $T_h = T_i$. 
Communication Failures

• Thus far, we have ignored communication failures!
• These can lead to non-serializable executions if network partitions result from the failures.
• Handled by the use of quorums – ensuring that only one of the partitions handles transactions.
• There are several alternatives for enforcing quorums.