Replication - Goals

- Performance
  - support more load than a single server
- Availability
  - one replica fails → use another
Replication - Approaches

• Hot Standby
  • use log-based recovery techniques to maintain secondary as up-to-date copy of primary

• Logical
  • multiple running DBs
  • writes on one DB are mirrored on other
    • synchronously (2PC) or
    • asynchronously (change queues)
Hot Standby

- Transactions at primary only
- Stream DB log from primary to secondary
- Apply process at secondary is like crash recovery
Redo Logging

• **Change:**
  • write NEW(xid, rid, newval) to log
  • change buffer in cache
  • mark buffer dirty, pinned

• **Commit:**
  • force COMMIT(xid) to log
  • unpin buffers for this txn
Redo Recovery

- Scan log backward computing set of all committed txns
- Scan log forward redoing writes from committed txns (only)
Undo Logging

- Change:
  - write OLD(xid, rid, oldval) to log
  - change buffer in cache
  - buffer pinned until log is forced

- Commit:
  - force buffers for this txn
  - write COMMIT(xid) to log
Undo Recovery

- Scan log forward computing set of all uncommitted txns
- Scan log backward restoring old values for uncommitted txns (only)
ARIES Algorithm

- Combine advantages of Undo and Redo
- Log contains old *and* new values for each change
- Ideal for hot standby ...
Apply Process at Secondary

- Normally reading log stream forward
  - apply all redo
  - maintain *active txn table*
- Synchronize secondary if primary fails
  - process log stream to end
  - read log backward and apply all undo for active (uncommitted) txns
• Primary must force log stream at commit to avoid lost txns
• If the cloud is a high-latency WAN this can be a serious problem
Hot Standby III

- Failover must empty *both* log queues...
- Must handle case where secondary site fails then recovers and needs to resynchronize
Routing to Current Primary

- See the analogy to load balancer?
Summary

• Advantages:
  • relatively simple
  • client sees only one DB, no replication

• Disadvantages
  • physical → secondary must be *identical* to primary
  • secondary is not usable for anything
Logical ...

- Oracle update-anywhere
- Described as triggers
  - Oracle product implements replication triggers internally
  - could even be done by processing the log
Triggers

• A common DB feature
• Register some code (usually a stored procedure in the DB) to be invoked under certain conditions
• DDL - create/drop table, etc.
• change/delete a row in a replicated table
  • trigger has access to table name and all (old and new) values for changed row
Triggers for Sync Replication

- Trigger for update at site s does the same update at each of the other sites
- distributed transaction (2PC)
- check whether this update is being done by trigger from another site!
Sync Replication

- **Advantages**
  - one-copy serializable

- **Disadvantages**
  - performance can be poor - especially if the replicas are far apart (high RTT)
  - site failure may affect every trigger at every remaining site
  - recovering a failed server requires synchronizing it with an available server.
    - *this is very difficult!*
  - nobody does it!
ASync Replication

- Trigger does not apply updates at other sites
- Instead, enqueue *update messages* in FIFO queues to be sent to other sites
- Run *apply processes* at each site
  - analogous to hot standby apply
  - logical (SQL) rather than physical (buffers)
  - check for old value before replacing with new value!
Update Message Propagation

- The queues are actually tables in the DBs
Update Anywhere

- Total number of queues is quadratic
  - only small numbers of replicas
- Weak consistency - *conflicts* are possible
An Essential Conflict

- When update messages arrive the old values are not as expected.
- *Conflict Detection* is guaranteed by at least one site.
- *Conflict Resolution* must happen consistently at all sites - difficult!
• DB0 might see *any permutation* of the change messages
Theorem

- The only consistent conflict resolution protocol is *latest timestamp wins*
  - requires timestamps in the data
  - as well as the update messages

- Semantically this may not be what you want
- But it is all you are going to get!
• Many client connections at DB1
  • high throughput
  • many simultaneously active txns
• Only one connection at DB2
  • one txn at a time
  • low throughput
Fix Propagation Performance with Parallelism?

- Can this work?
- Not without a little design work ...
Suppose T2 is a long-running txn
- Apply order at DB2 could easily be like T1; T3; T4; T2
- False conflicts!
In Oracle, each block holds the commit time of the last txn that modified the block.

Why? It’s a long story ...

Each txn T accumulates its dependency time

(latest commit time of any txn on which T depends)

This induces a partial order (DAG) on txns

Any topological sort of this DAG is a correct application order

results equivalent to original DB’s results
Oracle Parallel Prop - II

- Scheduler process
  - incremental topological sort of txns in the queue
  - assign work to apply processes
  - keep track of local commit times to determine when new txns may be scheduled
Recording Commit Times

- Gosh - that is *not* going to work!
- Solution: we put a special hook into the Oracle database engine
- I don’t know a better way :-(

```
insert into queue_table
values ( ... my_dependency_time,
        my_commit_time ... ) ;
commit ;
```
Suppose Not Oracle?

- Need solution to the “write my own commit time” problem
- Compute a dependency signature
  - vector of hash values of location (table name, PK value) of rows accessed by the txn
- Of course I haven’t actually done that!
Pros

- Secondary can be used (readonly) not just wasted
- When site fails and recovers, synchronization with current primary happens automatically
Cons

• Log based hot standby performs better than Replication, even when the parallel apply scheduler is used

• Queue is in the DB
  • cannot drain queue on failover
  • a few txns will be delayed until a failed primary recovers
  • likely to introduce false conflicts
Workaround ...

• Use Oracle Advanced Replication in update-anywhere mode
• Pretend it is hot standby