The University of Sydney

Topics in Database Isolation
IITB, January 2006
Lecture 3: Replica Management

Alan Fekete
(University of Sydney)
fekete@it.usyd.edu.au

Road Map

- Lecture 1: Isolation levels
- Lecture 2: Safe Use of Low Isolation
- Lecture 3: Replication Management
  - The key principle (R any, W all)
  - Global concurrency control
  - The main design choices
  - Serializable systems with lazy propagation
  - Using SI in replication
  - Limited divergence

Definition

- Replication is when the value of some data item is stored in more than one place
  - Typically in different databases at different physical locations
  - Similar issues arise with cached copies
- Eg keep a copy of the part-list at each warehouse

Motivation

- Performance
  - Each reader can find a copy close-by
    - Less latency to access the data
  - More parallelism, load-sharing
    - Improved throughput
- Fault-tolerance
  - Failure of some site doesn’t halt all activities
  - Graceful degradation

Key principle

- Read any copy
  - Preferably near to the client
- For unchanging data, this is wonderful! But what if the data item value sometimes changes (i.e. some transactions write the data)?
  - Write all the copies
  - This damages performance and fault-tolerance!
  - Thus replication is best for data where reads dominate over updates

Global transaction issues

- For now, ignore replication and just think about a system with multiple databases, and transactions that access them
- How to get global atomicity?
  - Use Two-phase commit
  - But this reduces performance markedly, especially during periods where some nodes are not available

Global serializability

- How to get serializable behavior?
- It is not enough for each db to provide serializable operation locally
- If each db uses 2PL, then global execution is serializable
  - All conflicts are compatible with the Commit order
- If you're not sure each db uses 2PL, and you want global serializability, you can
  - keep global serialization graph
  - introduce conflicts at every site through "ticket" updates

The main design choices

- There are many design choices for a system with replicated data. In the next slides, we present some of these, with sketches of the trade-offs involved.

Where to replicate?

- Everywhere
  - "total replication"
  - All dbs have identical contents
  - Any read can be done locally, with no cross-network communication
- Simple system design
- Performance may suffer

- Not everywhere
  - "partial replication"
  - Need to manage information about replica locations, and choose location for reads
  - Need to make choices about placement
- Complicated system design
- Performance may be improved

If partial, what to replicate?

- Complete tables
  - Each db has some of the tables
  - Easy to decide whether local copy exists for some data
  - Easy to reuse standard dbms engine for query optimization and processing
- Relatively simpler system design

- Fragments of tables
  - Keep copy of some rows, perhaps based on values in particular columns
  - Keep copy of some columns
  - Copy can be seen as a view of underlying global table
- Complex system design

How consistent?

- "Always" consistent
  - At least, apps shouldn't observe difference from using single dbms
    - "transparent replication"
  - Formal definition for "1 copy serializable (abbreviated as 1-SR)"
  - Some systems propose "1-copy SI"

- Eventually consistent
  - "convergent"
  - If updates cease for long enough, all copies will reach a common value
- Intermediate approach: limited divergence

How to propagate writes?

- Capture SQL statements, and execute at replicas
  - Difficulties if state is not the same as when originally executed

- Capture values written/inserted, and perform at replicas
  - Use triggers to capture information
  - Or access logs kept by each dbms

When to propagate writes?

- Eager
  - Update all replicas inside the original transaction
  - Requires two-phase commit
- Good for consistency
- Bad for performance
- Hybrid approach: do some remote activity, but not the updates themselves
- Lazy
  - "asynchronous"
  - Update one copy of each item inside original txn, then apply those writes that are relevant to replicas at a given site in a separate "copier" txn
  - Original txn may be entirely local at one site
- Good for performance
- May be bad for consistency

Is there a master?

- Primary copy
  - "master-slave"
  - One replica of each item is authoritative
  - It is always updated first
  - If lazy propagation, this either restricts transaction content, or forces non-local execution
- Bad for flexibility
- Group
  - "multimaster" or "update anywhere"
  - Different txns can update replicas in different orders
  - If eager propagation, then deadlock is very common;
  - If lazy propagation, then need conflict resolution to ensure convergence
- Good for flexibility

System architecture

- Middleware
  - Applications go through a veneer that manages global issues and then passes operations to local dbs
  - Middleware may not have enough information eg internal conflicts, risk of distributed deadlocks
  - No need to modify apps *if* they use JDBC or similar API
  - No need to modify engines
- More practical in most cases
- Engine-based
  - Modify each dbms to know about replication
  - No need to modify applications
  - Need to modify engines
- Hard to do except with open-source dbms, or if you work for one of the vendors!
  - Unlikely to work with heterogenous engines

Communication platform?

- Point-to-point messages
  - Eg socket programming
- Always present on any platform
- Programmer needs to deal with failures, and with out-of-order deliveries
- Can get good raw performance
- Group communication services
  - Eg Spread, Transis, etc
  - Deliver to all members of the group
  - Sender can require guarantees on order etc
- Much easier system design
- Performance may suffer

Design space summary

- In practice, want performance and simple system design
  - lazy propagation and primary copy
- In theory, want consistency and application generality
  - eager propagation, multi-master

Isolation and lazy propagation?

- If multi-master, then even convergence is hard to enforce
  - Need timestamps to recognize out-of-order updates
- So, assume primary copy
- Without restrictions on data and applications, reads can see old data
  - If a txn's reads are not all at same site, it might even see inconsistently old data

Example

- X has primary copy at A, replica at B
- Y has primary copy at B, replica at A
- T1 runs at A: r[X] r[Y] w[X]
  - Later copier T3 propagates write of X to B
- T2 runs at B: r[X] r[Y] w[Y]
  - Later copier T4 propagates write of Y to A
- At A: $r_1[X_A]$ $r_1[Y_A]$ $w_1[X_A]$ c_1 $w_4[Y_A]$ c_4
- At B: $r_2[X_B]$ $r_2[Y_B]$ $w_2[Y_B]$ c_2 $w_3[X_B]$ c_3
- Neither T1 nor T2 sees the other's changes

Example II

- X has primary copy at A, replicas at B and C
- Y has primary copy at B, replica at C
- T1 runs at A: r[X] w[X]
  - Later copier T4 propagates write of X to B
  - Copier T5 propagates write of X to C
- T2 runs at B: r[X] r[Y] w[Y]
  - Later copier T6 propagates write of Y to C
- T3 runs at C: r[X]r[Y]
- At A: $r_1[X_A]$ $w_1[X_A]$ c_1
- At B: $w_5[X_B]$ c_3 $r_2[X_B]$ $r_2[Y_B]$ $w_2[Y_B]$ c_2
- At C: $w_6[Y_C]$ c_6 $r_3[X_C]$ $r_3[Y_C]$ c_3 $w_5[X_C]$ c_5
- T2 sees T1, T3 sees T2 on Y (hence knows about T1) but does not see T1 on X

Restrictions

- Most work on serialization with lazy updates assumes a restricted model of data and apps
- We limit application logic so that each original transaction can run at one site
  - It accesses data with copies at that site
  - It only updates data whose primary copy is at that site
- Call this the "data ownership" assumption
  - This is common in practice, since app is usually focused on modifying data which "belongs" to the organisation or suborg which wrote the app
  - But it may read data which belongs elsewhere

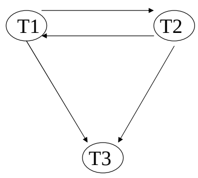
The copy graph

- Nodes are the sites where databases are located
- Edge from N_i to N_j if
  - There is an item X whose primary copy is located at N_i and which is replicated at N_j

Strongly Acyclic Copy graph

- CRR96 showed:
  - Assume data ownership model
  - Assume each db uses 2PL
  - Allow arbitrary execution of copier transactions,
  - then the overall execution is 1-copy serializable if and only if the undirected image of the copy graph has no cycles

Problem: cycle from T1 to T1



Problem: two different paths from T1 to T3

Combining OLTP and OLAP

- A special case has been widely used, where copy graph is a star
- Have one site which has the primary copy for all items (OLTP node)
- Other sites just run read-only queries (OLAP nodes)
- Eg RBSS'02, PA'04

Acyclic Copy Graph

- BKRSS99 introduced algorithms that work if *directed* copy graph has no cycle
- Key idea: ensure that copiers update nodes in a consistent order
  - Based on a tree
  - Or using timestamps
  - Could also be done with totally ordered multicast to carry each txn's copiers

Use of SI in Replication

- Because SI is now so common (Oracle, PostgreSQL), there is recently a lot of interest in replication using SI rather 2PL
- SW'00 shows how to ensure 1-SR using ticket or graph techniques
- WK'05, LKPJ'05 show how to get 1-SI
  - Without data ownership hypothesis
  - Using totally-ordered multicast

Combining local SI to 1-SI

- Assume each txn runs at a single site
- Then reading is determined by consistent snapshot
- But how to test for concurrent writes?
- Solution: deliver writeset info to other sites within the txn
  - But defer actually applying them
- Important to use db info so conflicts are checked at tuple not table granularity

Extensions

- LKPJ'05 also deals with many practical issues such as handling message failures, preventing deadlocks, detecting some conflicts early using the local SI properties
- Overall message: they get quite scalable performance

Relaxed Currency

- 1-SR allows read-only queries which run on out-of-date values
  - Some applications want limits on how old data might be
- RBSS'02 allows app to specify bound on staleness
- GLRG'04 provides SQL extension
  - And builds checks into query optimisation

Relaxed Consistency

- 1-SR and 1-SI both require all items read by txn T to come from a consistent view
  - This is easy if each txn runs at a single site
  - But it is hard if txn's reads can be spread around, for performance or because replication is not total
- Some applications may be willing to see data which were valid at slightly different times
- Much theory about controlling timing of updates to limit divergence of data values
  - Esp work on real-time databases
- GLRG'04 introduced SQL syntax to capture apps requirements
  - Focus on allocating reads to sites, rather than controlling divergence of sites

Future Work

- Replication across WANS
  - Order-enforcing group communication costs are very high here
- Limited divergence (QoS guarantees)
  - Integrating system mechanisms with application requirements

References

- Gray, Helland, O'Neil, Shasha *"The dangers of replication and a solution"* in SIGMOD'96
- Chundi, Rosenkrantz, Ravi *"Deferred updates and data placement in distributed databases"* in ICDE'96
- Breitbart, Komondoor, Rastogi, Seshadri, Silberschatz *"Update propagation protocols for replicated databases"* in SIGMOD'99

References

- Schenkel, Weikum *"Integrating snapshot isolation into transactional federations"* in CoopIS'00
- Röhm, Böhm, Schek, Schuldt *"FAS – a freshness-sensitive coordination middleware for a cluster of OLAP components"* in VLDB'02
- Plattner, Alonso *"Ganymed: scalable replication for transactional web applications"* in Middleware'04

References

- Guo, Larson, Ramakrishnan, Goldstein *"Relaxed Currency and Consistency: How to say good enough in SQL"* in SIGMOD'04
- Wu, Kemme *"Postgres-R(SI): combining replica control with concurrency control based on snapshot isolation"* in ICDE'05
- Lin, Kemme, Patino-Martinex, Jimenez-Peris *"Middleware-based data replication providing snapshot isolation"* in SIGMOD'05