

The University of Sydney
REPUBLICAN

Replication and transactions: Fekete

Replica Management and Transactions

Alan Fekete
 (University of Sydney)
 fekete@it.usyd.edu.au
 April 2009

Replication and transactions: Fekete

Road Map

- The key principle (R any, W all)
- Strong Consistency definitions
- The main system design choices
- Serializable systems with lazy propagation
- Using SI in replication
- Weak Consistency
- Limited divergence

2

Replication and transactions: Fekete

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

3

Replication and transactions: Fekete

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

4

Replication and transactions: Fekete

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

5

Replication and transactions: Fekete

Road Map

- The key principle (R any, W all)
- Strong Consistency definitions
- The main system design choices
- Serializable systems with lazy propagation
- Using SI in replication
- Weak Consistency
- Limited divergence

6

Replication and transactions: Fekete

Consistency properties

- For users of a data-storage system, it is vital that they know what properties they get from the data
- In a distributed/replicated system, clients may be exposed to strange effects they would never see with a single database
 - Data that suddenly forgets what it previously knew
 - Data items that are not properly related to one another
- Be explicit about what might happen
 - Guide the developer on how to cope

7

Replication and transactions: Fekete

Strong consistency

- The replicas of an item are “always” consistent
 - At least, whenever values can be read, they are the same in all replicas of an item
- Of course, while a transaction is in flight, there will be delays in reaching all the replicas
 - Simultaneous activity not really possible across a network

8

Replication and transactions: Fekete

Client-view definition

- The system: clients request operations and get results
- Abstract away internals
 - Property of allowed sequence of events at the boundary

Sometimes, each client is bound to one site

9

Replication and transactions: Fekete

Strong consistency definition

- Clients see exactly what they would see interacting with a single unreplicated dbms using proper concurrency control to support transactions
 - “Transparent” replication
- Formal definition for “1 copy serializable” (abbreviated as 1-SR)
 - Defined in BG’85
- Some systems propose “1-copy SI”
 - As if interacting with a single unreplicated dbms using SI as concurrency control
 - Defined in PA’04

10

Replication and transactions: Fekete

Session guarantees

- Clients want “session consistent”
 - Each transaction is serialized after any previous transactions from the same client
 - Since every single-site unreplicated dbms has this
 - Client sees the effects of activities s/he knows have been performed
- Even more: “externally consistent”
 - Serialization order is compatible with partial order of transactions in real-time
 - If T1 finishes before T2 starts, T1 is serialized first
 - T2 sees effects of anything it could find out about, through out-of-band information flow

11

Replication and transactions: Fekete

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

12

Replication and transactions: Fekete

Global serializability

- How to get serializable behavior (1-SR)?
 - 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
 - See BG'85
 - If you're not sure each db uses 2PL, and you want global serializability, you need to do more work
 - keep global serialization graph
 - or introduce conflicts at every site through "ticket" updates (See GRS'94)

13

Replication and transactions: Fekete

Road Map

- The key principle (R any, W all)
- Strong Consistency definitions
- **The main system design choices**
- Serializable systems with lazy propagation
- Using SI in replication
- Weak Consistency
- Limited divergence

14

Replication and transactions: Fekete

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.

15

Replication and transactions: Fekete

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

16

Replication and transactions: Fekete

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

17

Replication and transactions: Fekete

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

18

Replication and transactions: Fekete

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

19

Replication and transactions: Fekete

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

20

Replication and transactions: Fekete

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

21

Replication and transactions: Fekete

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

22

Replication and transactions: Fekete

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
- Seminal paper: GHOS’96

23

Replication and transactions: Fekete

Road Map

- The key principle (R any, W all)
- Strong Consistency definitions
- The main system design choices
- **Serializable systems with lazy propagation**
- Using SI in replication
- Weak Consistency
- Limited divergence

24

Replication and transactions: Fekete

Isolation and lazy propagation?

- For now, assume primary copy
- Can we get strong consistency with a lazy propagation system design?
- Without restrictions on data and applications, reads can see old data
 - If a txn's reads are not all at same site, it might see inconsistently old data

25

Replication and transactions: Fekete

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

26

Replication and transactions: Fekete

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: $w_3[X_B] c_3 r_2[X_B] r_2[Y_B] w_2[Y_B] c_2$
- At C: $w_6[Y_C] c_6 r_3[X_C] r_3[Y_C] c_3 w_4[X_C] c_5$
- T2 sees T1, T3 sees T2 on Y (hence knows about T1) but does not see T1 on X

27

Replication and transactions: Fekete

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

28

Replication and transactions: Fekete

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

29

Replication and transactions: Fekete

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

30

Replication and transactions: Fekete

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

31

Replication and transactions: Fekete

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

32

Replication and transactions: Fekete

Road Map

- The key principle (R any, W all)
- Strong Consistency definitions
- The main system design choices
- Serializable systems with lazy propagation
- **Using SI in replication**
- Weak Consistency
- Limited divergence

33

Replication and transactions: Fekete

Use of SI in Replication

- Because SI is now so common (Oracle, PostgreSQL), there is recently a lot of interest in replication using SI at each node, rather than 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

34

Replication and transactions: Fekete

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

35

Replication and transactions: Fekete

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
- EZP'05, DS'06 show benefit from slightly weakening correctness condition
 - Txn T may not see effects of all others that commit before T starts
 - But each T sees effects from some consistent set of prior txns

36

Replication and transactions: Fekete

Road Map

- The key principle (R any, W all)
- Strong Consistency definitions
- The main system design choices
- Serializable systems with lazy propagation
- Using SI in replication
- **Weak Consistency**
- Limited divergence

37

Replication and transactions: Fekete

WAN Replication

- Unlike a LAN, WAN sometimes partitions
 - “you can’t get there from here, at present”
- It’s unreasonable to deny or delay requests just because some replica is unreachable
- This is even more so with mobile devices
 - Partially disconnected operation
- And at internet-scale, where there is always some node that is down
- Thus, update anywhere and lazy propagation!

38

Replication and transactions: Fekete

Conflicting updates

- With update anywhere, lazy propagation, and no other mechanism, the replicas of an item can become permanently different from one another: each node sees a different state of the world
 - “split brain” situation

39

Replication and transactions: Fekete

Split brain Example

- X has replicas at A, B
- T1 runs at A: $r[X] w[X]$
 - Later copier T3 propagates write of X to B
- T2 runs at B: $r[X] w[X]$
 - Later copier T4 propagates write of X to A
- At A: $r_1[X_A] w_1[X_A] c_1 w_4[X_A] c_4$
- At B: $r_2[X_B] w_2[X_B] c_2 w_3[X_B] c_3$
- Final state of A: effect of T2 only (T1 has been lost)
- Final state of B: effect of T1 only (T2 has been lost)

40

Replication and transactions: Fekete

Eventual consistency

- If updates cease for long enough, all replicas of an item will reach a common value
 - Defined in DGHKSSST’87
 - Importance for cloud computing noted in V’09
- Mechanisms (eg from KS’01)
 - Label values with timestamps to recognize out-of-order updates
 - If all writes are given by value: just ignore obsolete writes (leave replica with latest write’s value)
 - If updates are given by operation: rollback previous updates to fit in the missed one, then replay previous updates
 - Or report conflict to human, for resolution

41

Replication and transactions: Fekete

Client-view definition of eventual consistency

- For each transaction, there is a sequence (“the justifying view”) which
 - consists of a subset of transactions that were submitted (perhaps with different return values than actually happened) that could be followed by the given transaction with the return values it saw
- Eventual consistency:
 - There exists in each such sequence a prefix (“the agreed past”)
 - All agreed pasts are prefixes of a single sequence
 - Which merges all the requested transactions

42

Replication and transactions: Fekete

Session guarantees

- “Session consistency”
 - Each request’s justifying view includes previous requests from the same client
 - Importance identified by TDPSW’94
- Or stronger: “causal consistency”
 - Each request’s justifying view includes any previous request which could be known to the client at the time of the request
 - Communicated through any chain of messages

43

Replication and transactions: Fekete

Data Partitioning

- Eventual consistency is often defined item by item
 - But a transaction might operate on several items
- To keep performance benefits, cloud computing platforms often restrict each transaction so all its items are in the same data partition
 - They are co-located on any node, and the transaction can be done locally on such a node
 - Eg All data for a customer is one partition
- Eg DHJKLPSVV’07

44

Replication and transactions: Fekete

Road Map

- The key principle (R any, W all)
- Strong Consistency definitions
- The main system design choices
- Serializable systems with lazy propagation
- Using SI in replication
- Weak Consistency
- Limited divergence

45

Replication and transactions: Fekete

Data Divergence

- Many systems allow replicas to be slightly different from one another
- “Difference” can be expressed in value, or time gap until updates are applied, or number of updates not yet applied at a node, etc
- Such replicas are sometimes called “quasi-copies”

46

Replication and transactions: Fekete

Benefits of divergence

- System can get better performance
 - Downside: applications programming may be harder, to cope with inaccuracies in reads
- Reads can use replicas where some updates haven’t arrived yet
- Some updates may never need to be propagated
 - If a later update will be propagated before divergence grows too big
- See ABG’90, and more recently OLW’01, SRS’04
 - PCVO’05 gets strong client-view consistency with divergent replicas

47

Replication and transactions: Fekete

Relaxed Consistency

- Instead of looking at limiting divergence in the data throughout the execution, consider the client’s view of each transaction
- Limit on how far from “ideal state” is the result allowed to be, in an operation
 - Measured by value, or number of operations submitted but not in justifying view, or by time since earliest update not in justifying view
 - also can bound drift between state seen in various reads of multiple items within a txn

48

Freshness

- Formal def of 1-SR allows read-only queries which run on out-of-date values
 - Theoretically legal in a single site dbms too, but never seen with any common implementation mechanism
- Each transaction might wish to insist on data that was stale by no more than Δ
 - If $\Delta=0$, then external consistency is obtained

49

Providing txn-specific relaxation

- RBSS'02 accept txn-specified staleness limit
 - Done for read-only txns, with 1-SR required
 - All reads by T happen from consistent state, but that state might be stale in real-time
 - Allocate each txn to a node which might be delayed in applying updates
- GLRG'04 also accept txn-specified drift between reads by T, and also allow value-based divergence
 - Introduced SQL syntax to capture app requirements
 - Different reads may be at different sites
- BFGRT'06 allowed divergent reads in updating transactions

50

References

- Bernstein and Goodman “*Serializability theory for replicated databases*” in JCSS, 1985
- Demers, Greene, Hauser, Irish, Larson, Shanker, Sturgis, Swinehart, Terry “*Epidemic algorithms for replicated database maintenance*” in PODC'87
- Alonso, Barbara, and Garcia-Molina “*Data caching issues in an information retrieval system*” in ACM TODS 1990

51

References

- Georgakopoulos, Rusinkiewicz, Sheth “*Using tickets to enforce the serializability of multidatabase transactions*” in IEEE TKDE, 1994
- Terry, Demers, Petersen, Spreitzer, Welch “*Session guarantees for weakly consistent replicated data*” in PDIS'94
- Chundi, Rosenkrantz, Ravi “*Deferred updates and data placement in distributed databases*” in ICDE'96

52

References

- Gray, Helland, O'Neil, Shasha “*The dangers of replication and a solution*” in SIGMOD'96
- Breitbart, Komondoor, Rastogi, Seshadri, Silberschatz “*Update propagation protocols for replicated databases*” in SIGMOD'99
- Schenkel, Weikum “*Integrating snapshot isolation into transactional federations*” in CoopIS'00

53

References

- Olston, Loo, Widom “*Adaptive precision setting for cached approximate values*” in SIGMOD'01
- Kistler, Satyanarayanan “*Disconnected operation in the CODA file system*” in SOSP'01
- Röhm, Böhm, Schek, Schudt “*FAS – a freshness-sensitive coordination middleware for a cluster of OLAP components*” in VLDB'02

54

References

- Shah, Ramamritham, Shenoy “*Resilient and coherence preserving dissemination of dynamic data using cooperating peers*” In IEEE TKDE, 2004.
- Plattner, Alonso “*Ganymed: scalable replication for transactional web applications*” in Middleware’04
- Guo, Larson, Ramakrishnan, Goldstein “*Relaxed Currency and Consistency: How to say good enough in SQL*” in SIGMOD’04

55

References

- Pacitti, Coulon, Valduriez, Ozsü “*Preventative replication in a database cluster*” In Distributed and Parallel Databases, 2005.
- Wu, Kemme “*Postgres-R(SI): combining replica control with concurrency control based on snapshot isolation*” in ICDE’05
- Lin, Kemme, Patino-Martinez, Jimenez-Peris “*Middleware-based data replication providing snapshot isolation*” in SIGMOD’05

56

References

- Elnikety, Zwaenepoel and Pedone “*Database replication using generalized Snapshot Isolation*” in SRDS’05
- Dudgee and Salem “*Lazy database replication with snapshot isolation*” in VLDB’06
- Bernstein, Fekete, Guo, Ramakrishnan, and Tamma “*Relaxed currency serializability for middle-tier caching and replication*” in SIGMOD’06

57

References

- DeCandia, Hastorun, Jampani, Kakulapati, Lakshman, Pilchin, Sivasubramanian, Vosshall, Vogels “*Dynamo: Amazon’s Highly Available Key-value store*” in SOS’07
- Vogels “*Eventually consistent*” in Comm ACM, 2009

58