Distributed Databases and Replication

Harvard Extension School

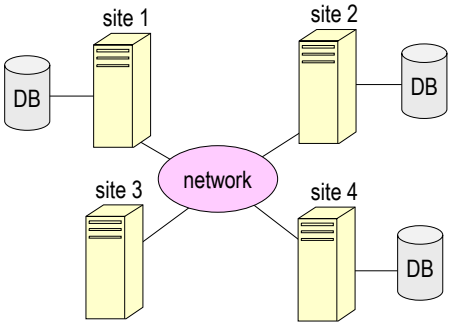
Cody Doucette, Ph.D.

Lecture designed by David G. Sullivan

What Is a Distributed Database?

- One in which data is:
  - *partitioned / fragmented* among multiple machines
    and/or
  - *replicated* – copies of the same data are made available on multiple machines
- It is managed by a *distributed DBMS (DDBMS)* – processes on two or more machines that jointly provide access to a single logical database.
- The machines in question may be:
  - at different locations (e.g., different branches of a bank)
  - at the same location (e.g., a cluster of machines)
- In the remaining slides, we will use the term *site* to mean one of the machines involved in a DDBMS.
  - may or may not be at the same location

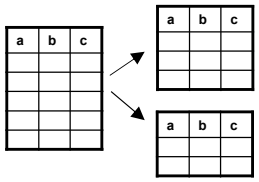
What Is a Distributed Database? (cont.)



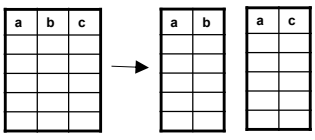
- A given site may have a local copy of all, part, or none of a particular database.
  - makes requests of other sites as needed

Fragmentation / Sharding

- Divides up a database's records among several sites
  - the resulting "pieces" are known as *fragments/shards*
- Let R be a collection of records of the same type (e.g., a relation).
- *Horizontal fragmentation* divides up the "rows" of R.
  - R(a, b, c) → R1(a, b, c), R2(a, b, c), …
  - R = R1 ∪ R2 ∪ …



- *Vertical fragmentation* divides up the "columns" of R.
  - R(a, b, c) → R1(a, b), R2(a, c), … (a is the primary key)
  - R = R1 ⋈ R2 ⋈ …



Fragmentation / Sharding (cont.)

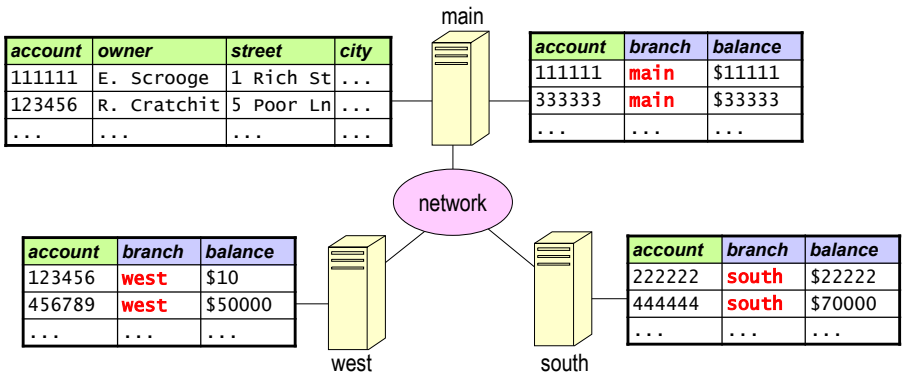
- Another version of vertical fragmentation:
  divide up the tables (or other collections of records).
  - e.g., site 1 gets tables A and B
    site 2 gets tables C and D

Example of Fragmentation

- Here's a relation from a centralized bank database:

account	owner	street	city	branch	balance
111111	E. Scrooge	1 Rich St	...	main	$11111
123456	R. Cratchit	5 Poor Ln	...	west	$10
...	...	...	...	...	...

- Here's one way of fragmenting it:



account	owner	street	city
111111	E. Scrooge	1 Rich St	...
123456	R. Cratchit	5 Poor Ln	...
...	...	...	...

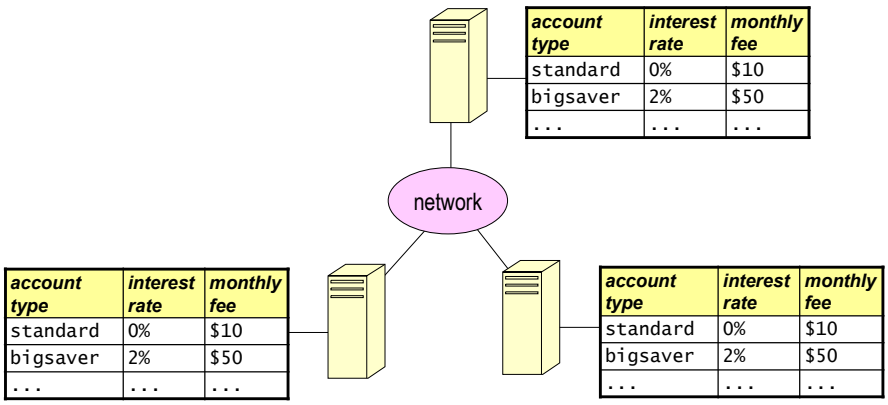
account	branch	balance
111111	main	$11111
333333	main	$33333
...	...	...

account	branch	balance
123456	west	$10
456789	west	$50000
...	...	...

account	branch	balance
222222	south	$22222
444444	south	$70000
...	...	...

Replication

- Replication involves putting copies of the same collection of records at different sites.



Reasons for Using a DDBMS

- to improve performance
  - how does distribution do this?
- to provide *high availability*
  - replication allows a database to remain available in the event of a failure at one site
- to allow for modular growth
  - add sites as demand increases
  - adapt to changes in organizational structure
- to integrate data from two or more existing systems
  - without needing to combine them
  - allows for the continued use of legacy systems
  - gives users a unified view of data maintained by different organizations

Challenges of Using a DDBMS (partial list)

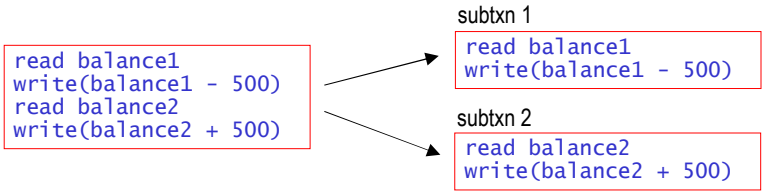
- determining the best way to distribute the data
  - when should we use vertical/horizontal fragmentation?
  - what should be replicated, and how many copies do we need?
- determining the best way to execute a query
  - need to factor in communication costs
- maintaining integrity constraints (primary key, foreign key, etc.)
- ensuring that copies of replicated data remain consistent
- managing *distributed txns*: ones that involve data at multiple sites
  - atomicity and isolation can be harder to guarantee

Failures in a DDBMS

- In addition to the failures that can occur in a centralized system, there are additional types of failures for a DDBMS.
- These include:
  - the loss or corruption of messages
    - TCP/IP handles this type of error
  - the failure of a site
  - the failure of a communication link
    - can often be dealt with by rerouting the messages
  - *network partition:* failures prevent communication between two subgroups of the sites

Distributed Transactions

- A *distributed transaction* involves data stored at multiple sites.
- One of the sites serves as the *coordinator* of the transaction.
  - one option: the site on which the txn originated
- The coordinator divides a distributed transaction into *subtransactions*, each of which executes on one of the sites.



Types of Replication

- In *synchronous replication*, transactions are guaranteed to see the most up-to-date value of an item.
- In *asynchronous replication*, transactions are *not* guaranteed to see the most up-to-date value.

Synchronous Replication I: Read-Any, Write-All

- *Read-Any:* when reading an item, access *any* of the replicas.
- *Write-All:* when writing an item, must update *all* of the replicas.
- Works well when reads are much more frequent than writes.
- Drawback: writes are very expensive.

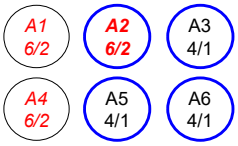
Synchronous Replication II: Voting

- When writing, update some fraction of the replicas.
  - each value has a version number that is increased when the value is updated
- When reading, read enough copies to ensure you get at least one copy of the most recent value (see next slide).
  - the copies "vote" on the value of the item
  - the copy with the highest version number is the most recent
- Drawback: reads are now more expensive

Synchronous Replication II: Voting (cont.)

- How many copies must be read?
  - let: n = the number of copies
    w = the number of copies that are written
    r = the number of copies that are read
  - need: r > n – w (i.e., at least n – w + 1)
  - example: n = 6 copies
    update w = 3 copies
    must read at least 4 copies

- Example: 6 copies of data item A, each with value = 4, version = 1.
  - txn 2 updates A1, A2, and A4 to be 6 (and their version number becomes 2)
  - txn 1 reads A2, A3, A5, and A6
  - A2 has the highest version number (2), so its value (6) is the most recent.



Which of these allow us to ensure that clients always get the most up-to-date value?

- 10 replicas – i.e., 10 copies of each item

- voting-based approach with the following requirements:

	number of copies accessed when reading	number of copies accessed when writing
A.	7	3
B.	5	5
C.	9	2
D.	4	8

(select all that work)

Distributed Concurrency Control

- To ensure the isolation of distributed transactions, need some form of distributed concurrency control.
- Extend the concurrency control schemes that we studied earlier.
  - we'll focus on extending strict 2PL
- If we just used strict 2PL at each site, we would ensure that the schedule of subtxns *at each site* is serializable.
  - why isn't this sufficient?

Distributed Concurrency Control (cont.)

- Example of why special steps are needed:
  - voting-based synchronous replication with 6 replicas
  - let's say that we configure the voting as before:
    - each write updates 3 copies
    - each read accesses 4 copies
  - can end up with schedules that are not conflict serializable
  - example:

T_1	T_2
xl(A1); xl(A2); xl(A3)	
w(A1); w(A2); w(A3)	
	xl(A4); xl(A5); xl(A6)
	w(A4); w(A5); w(A6)
	xl(B4); xl(B5); xl(B6)
	w(B4); w(B5); w(B6)
xl(B1); xl(B2); xl(B3)	
w(B1); w(B2); w(B3)	

Xi = the copy of item X at site i

T1 should come *before* T2 based on the order in which they write A.

T1 should come *after* T2 based on the order in which they write B.

What Do We Need?

- We need shared and exclusive locks for a *logical item*, not just for individual copies of that item.
  - referred to as *global locks*
  - doesn't necessarily mean locking every copy
- Requirements for global locks:
  - no two txns can hold a global exclusive lock for the same item
  - any number of txns can hold a global shared lock for an item
  - a txn cannot acquire a global exclusive lock on an item if another txn holds a global shared lock on that item, and vice versa

What Do We Need? (cont.)

- In addition, we need to ensure the correct ordering of operations *within* each distributed transaction.
  - don't want a subtxn to get ahead of where it should be in the context of the txn as a whole
  - relevant even in the absence of replication
  - one option: have the coordinator of the txn acquire the necessary locks before sending operations to a site

Option 1: Centralized Locking

- One site manages the lock requests for *all* items in the distributed database.
  - even items that don't have copies stored at that site
  - since there's only one place to acquire locks, these locks are obviously global locks!
- Problems with this approach?
  - the lock site can become a bottleneck
  - if the lock site crashes, operations at all sites are blocked

Option 2: Primary-Copy Locking

- One copy of an item is designated the *primary copy*.
- The site holding the primary copy handles all lock requests for that item.
  - acquiring a shared lock for the primary copy gives you a global shared lock for the item
  - acquiring an exclusive lock for the primary copy gives you a global exclusive lock for the item
- To prevent one site from becoming a bottleneck, distribute the primary copies among the sites.
- Problem: If a site goes down, operations are blocked on all items for which it holds the primary copy.

Option 3: Fully Distributed Locking

- No one site is responsible for managing lock requests for a given item.
- A transaction acquires a global lock for an item by locking a sufficient number of the item's copies.
  - these local locks combine to form the global lock
- To acquire a global shared lock, acquire local shared locks for a sufficient number of copies (see next slide).
- To acquire a global exclusive lock, acquire local exclusive locks for a sufficient number of copies (see next slide).

Option 3: Fully Distributed Locking (cont.)

- How many copies must be locked?
  - let: n = the total number of copies
    x = the number of copies that must be locked to acquire a global exclusive lock
    s = the number of copies that must be locked to acquire a global shared lock
  - we need $x > n/2$
    - guarantees that no two txns can both acquire a global exclusive lock at the same time
  - we need $s > n - x$ (i.e., $s + x > n$)
    - if there's a global exclusive lock on an item, there aren't enough unlocked copies for a global shared lock
    - if there's a global shared lock on an item, there aren't enough unlocked copies for a global excl. lock

Option 3: Fully Distributed Locking (cont.)

- Our earlier example would no longer be possible:

T_1	T_2
xl(A1); xl(A2); xl(A3) w(A1); w(A2); w(A3)	xl(A4); xl(A5); xl(A6) w(A4); w(A5); w(A6)
xl(B1); xl(B2); xl(B3) w(B1); w(B2); w(B3)	xl(B4); xl(B5); xl(B6) w(B4); w(B5); w(B6)

T_1	T_2
xl(A1); xl(A2); xl(A3); xl(A6) w(A1); w(A2); w(A3); w(A6)	xl(A4); xl(A5); xl(A6) – ***denied*** must wait for T1

- n = 6
- need x > 6/2
- must acquire at least 4 local exclusive locks before writing

Synchronous Replication and Fully Distributed Locking

- Read-any write-all:
  - when writing an item, a txn must update all of the replicas
    - this gives it x = n exclusive locks, so x > n/2
  - when reading an item, a txn can access any of the replicas
    - this gives it s = 1 shared lock, and 1 > n – n
- Voting:
  - when writing, a txn updates *a majority of the copies* – i.e., w copies, where w > n/2.
    - this gives it x > n/2 exclusive locks as required
  - when reading, a txn reads r > n – w copies
    - this gives it s > n – x shared locks as required

Which of these would work?

- 9 replicas – i.e., 9 copies of each item
- *fully distributed* locking
- voting-based approach with the following requirements:

	number of copies read	written
A.	5	5
B.	6	4
C.	7	3
D.	4	5

(select all that work)

Which of these would work?

- 9 replicas – i.e., 9 copies of each item
- *primary-copy* locking
- voting-based approach with the following requirements:

	number of copies read	written
A.	5	5
B.	6	4
C.	7	3
D.	4	5

(select all that work)

Distributed Deadlock Handling

- Under centralized locking, we can just use one of the schemes that we studied earlier in the semester.
- Under the other two locking schemes, *deadlock detection* becomes more difficult.
  - local waits-for graphs alone will not necessarily detect a deadlock
    - example:

  - one option: periodically send local waits-for graphs to one site that checks for deadlocks
- Instead of using deadlock detection, it's often easier to use a timeout-based scheme.
  - if a txn waits too long, presume deadlock and roll it back!

Recall: Types of Replication

- In *synchronous replication*, transactions are guaranteed to see the most up-to-date value of an item.
- In *asynchronous replication*, transactions are *not* guaranteed to see the most up-to-date value.

Asynchronous Replication I: Primary Site

- In *primary-site replication*, one replica is designated the *primary* or *master* replica.
- All writes go to the primary.
  - propagated asynchronously to the other replicas (the *secondaries*)
- The secondaries can only be read.
  - no locks are acquired when accessing them
  - thus, we only use them when performing read-only txns
- Drawbacks of this approach?

Asynchronous Replication II: Peer-to-Peer

- In *peer-to-peer replication*, more than one replica can be updated.
- Problem: need to somehow resolve conflicting updates!