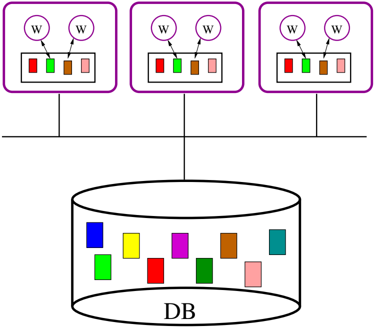
Distributed Database Systems

Why build distributed database systems?

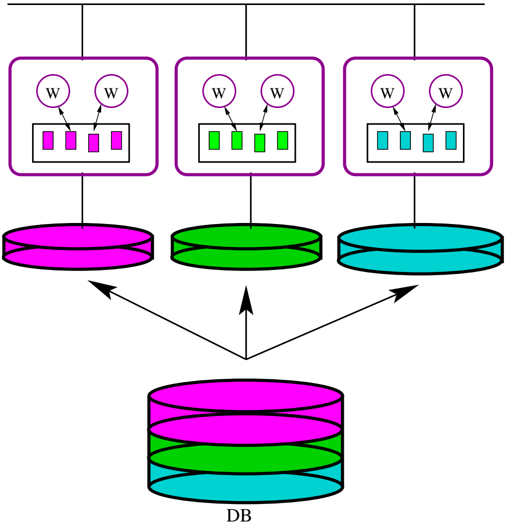
- fault tolerance: one server fails, others keep running
- scale out:
  - intra-query parallelism: make queries run faster by parallelizing execution
  - inter-query parallelism: increase throughput by distributing queries to different servers
- federation: "meta" database system that allows queries spanning multiple existing systems

Shared Storage DBMS Architectures



- one database accessed by multiple DBMS
- need to synchronize access by different DBMS to the database
- need to maintain coherence among DBMS caches
- transactions can be localized at a single DBMS

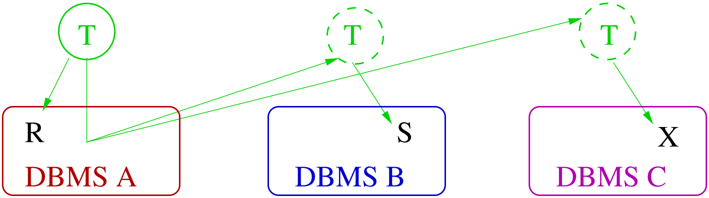
Shared Nothing DBMS Architecture: Partitioning Approach



Shared Nothing Partitioning

- each server stores and is responsible for part of the database
- transactions may be distributed
- goals:
    - improved response times via intra-query parallelism
    - improved throughput by via inter-query parallelism
- issues:
    - how to partition data to maximize parallelism or transaction localization?
    - how to enforce ACID properties of distributed transactions?

Distributed Transactions



1. `UPDATE R`
2. `UPDATE S`
3. `UPDATE X`

Global transaction consists of subtransactions at each site at which data are read or written.

Serializability in Distributed Systems

- execution history at each site describes order of execution of read and write operations of that site's subtransactions
- each site uses a local concurrency control mechanism (e.g., two-phase locking) to control the local execution order and serialize the local subtransactions
- global serializability is achieved if
  - the local execution history at every site is serializable
  - there is some total ordering of global transactions that is consistent with the local serialization order at every site

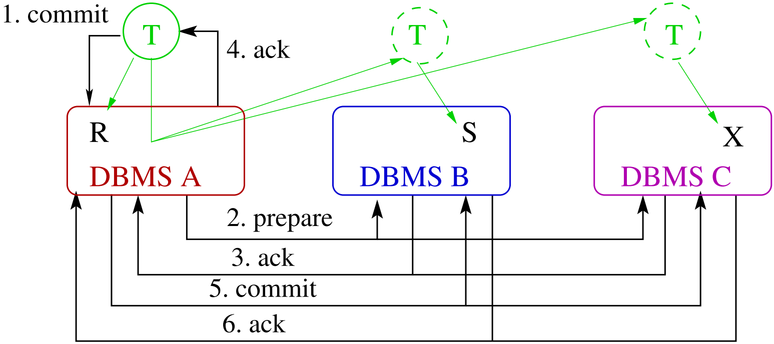
local serializability does not imply global serializability

Different sites may serialize global transactions in different orders.

Failure Atomicity in Distributed Systems

- each site can use a recovery mechanism (e.g., logging) to ensure that local subtransactions are atomic and durable
  - local subtransaction is committed when its commit record is in the local, persistent log
- partial failures are possible: some sites are down, others are up
- to ensure that a distributed transaction is atomic, we must ensure that either all of its subtransactions commit, or all of them abort

Two-Phase Commit (2PC)



The 2PC Protocol

One site acts as the coordinator.

Phase 1:

- The coordinator sends "prepare" to the other sites.
- Each site decides whether it wants to commit or abort the transaction and sends its vote to the coordinator
  - if abort, it writes an abort record in its log, and votes for abort
  - if commit, it writes a prepare record in its log, and votes for commit

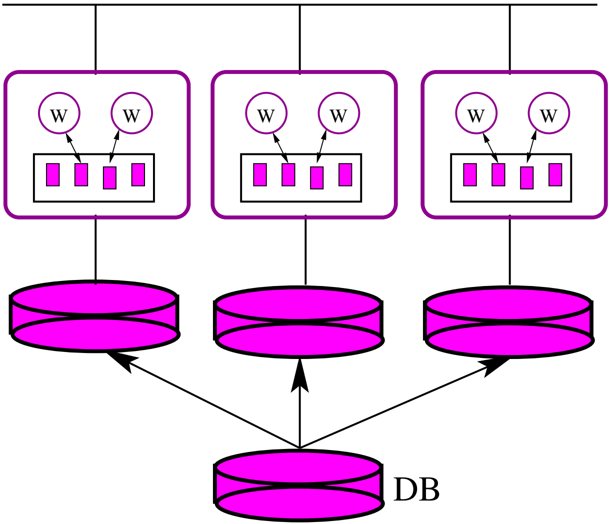
Phase 2:

- If all sites vote commit, the coordinator writes a commit record in its log, otherwise it writes an abort record. The coordinator sends its decision to all of the sites.
- Each site that voted to commit records the decision in its log and sends an acknowledgment to the coordinator.

2PC Discussion

- A distributed transaction is committed when the coordinator logs a commit record, at the end of Phase 1.
- Failure of the coordinator may result in blocking at other sites:
    - once a site sends “prepare” to the coordinator in Phase 1, the transaction is said to be in doubt at the site
    - a site may not unilaterally commit or abort a transaction that is in doubt!
    - if the coordinator does not report its commit/abort decision in Phase 2, e.g., because of a failure, other sites must wait for the coordinator to recover to learn the fate of in doubt transactions

Shared Nothing DBMS Architecture
Replication Approach

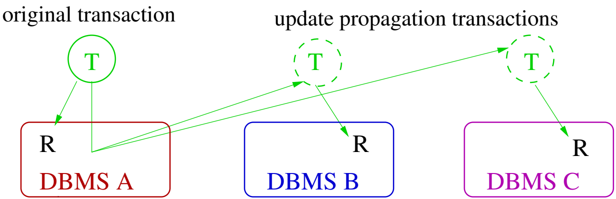


Shared Nothing Replication

- each server stores and is responsible for a copy of the database
- transactions run at a single site (except update synchronization)
- goals:
  - improved throughput by via inter-query parallelism
  - improved availability
- issues:
  - how to synchronize replicas so that ACID properties of transactions are maintained?

Update Propagation

- each transaction can run at a single site, as every site has a copy of the complete databases
- for each update transaction, the system initiates update propagation transactions at other sites
- eager propagation means that the original update transaction and its propagation transactions commit or abort as a single global distributed transaction
- lazy propagation means that the original update transaction commits first, and its propagation transactions commit separately later.



Global 1-Copy Serializability

- execution history at each site includes transactions initiated at that site, plus update propagation transactions from other sites
- each site uses a local concurrency control mechanism (e.g., two-phase locking) to control the local execution order and serialize the local transactions
- global 1-copy serializability (1SR) is achieved if it appears as if all transactions executed sequentially on a single copy of the database
- 1SR is achieved if
  - the local execution history at every site is serializable
  - there is some total ordering of update transactions that is consistent with the local serialization order at every site

Eager Read-One, Write-All (ROWA) Replication

- each transaction runs at a single site
- for update transactions, the DBMS initiates update propagation transactions at all other sites
- commit of an update transaction and its update propagation transactions is coordinated using two-phase commit
- replicas remain tightly synchronized

Correctness

Local strict two-phase locking at each site, plus two-phase commit of update propagation transactions, is sufficient to ensure global 1-copy serializability

Lazy-Master Replication

- a single master site handles all update transactions
- slave sites handle read-only transactions
- local concurrency control at the master site serializes update transactions
- updates are propagated lazily, in serialization order, from the master site to the slaves
- slaves execute and serialize update propagation transactions in propagation order

Correctness

Local serializability at each site, plus in-order propagation of updates from master to slaves, is sufficient to ensure global 1-copy serializability. However, read-only transactions may see stale data.

Lazy-Master vs. Eager ROWA

- Advantage of Eager ROWA: freshness
  - all transactions get an up-to-date view of the database
- Advantage of Lazy-Master: no two-phase commit
  - update transactions (at the master) and propagation transactions (at the slaves) commit independently
- scalability
  - lazy-master scales easily - by adding more slaves - until the master site becomes a bottleneck
  - commit coordination (2pc) limits scalability of eager replication: more sites means slower update transactions