Distributed Databases

Knowledge objectives

1. Enumerate the main goals of Big Data Management
2. Explain the different transparency layers in DDBMS
3. Draw a classical reference architecture for DDBMS
4. Enumerate the five main challenges in data distribution
5. Distinguish vertical and horizontal fragmentation
6. Recognize the complexity and benefits of data allocation
7. Distinguish the four replication synchronization strategies
8. Explain the main obstacles to achieve linear scalability according to the Universal Scalability Law
9. Explain the benefits and difficulties of a distributed catalog
10. Enumerate the phases of distributed query processing
11. Explain the difference between data shipping and query shipping
12. Explain the different kinds of parallelism
13. Distinguish between answer time and query cost
14. Explain the CAP theorem
15. Distinguish between ACID and BASE
16. Explain the benefits of sequential read in front of random access
17. Explain five characteristics of NewSQL architectures

Understanding Objectives

1. Identify the potential consequences of a given data replication strategy
2. Given an query and a dabatase design, recognize the difficulties and opportunities behind distributed query processing

Big Data Management Goals (I)

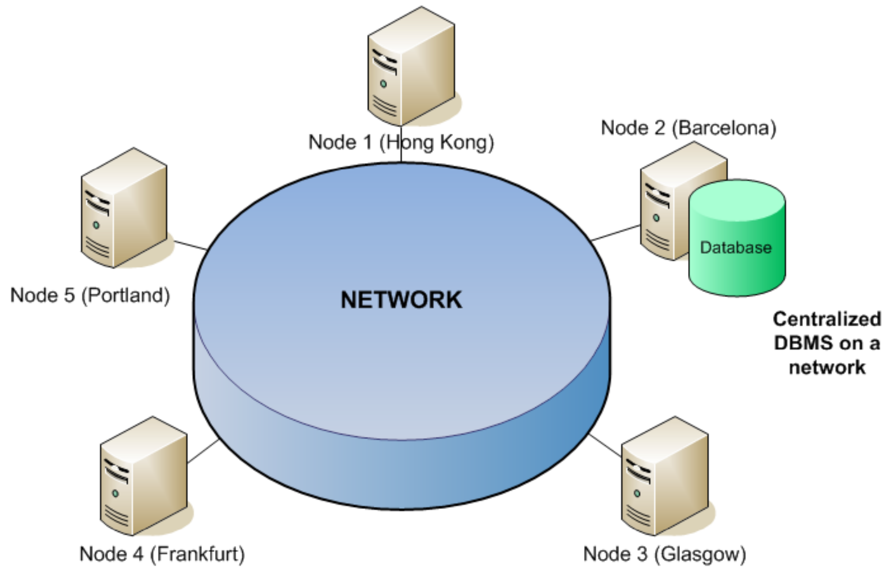
- Schemalese: No explicit schema [data structure]
- Reliability / availability: Keep delivering service even if its software or hardware components fail [distribution]
- Scalability: Continuously evolve to support a growing amount of tasks [distribution]
- Efficiency: How well the system performs, usually measured in terms of response *time* (latency) and *throughput* (bandwith) [distribution]

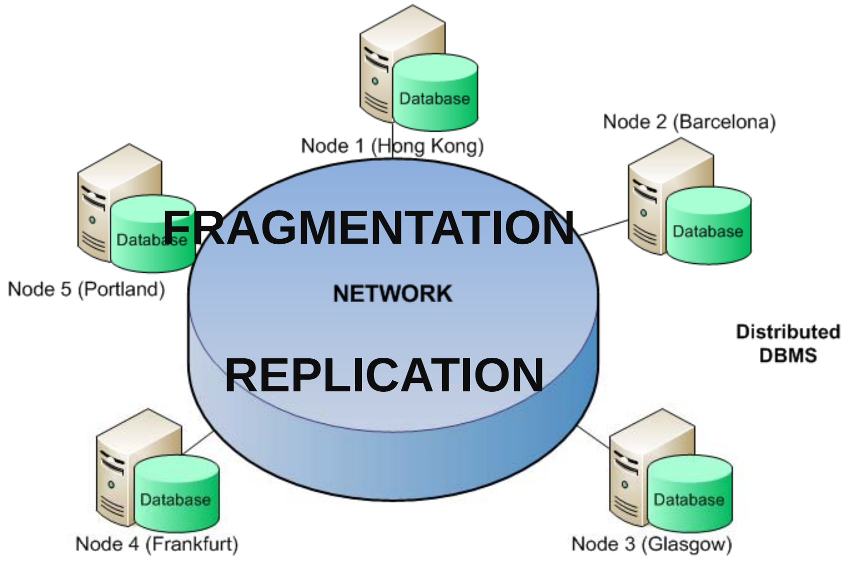
Big Data Management Goals(II)

- Fragmentation (Parallel writes)
  - At design time (static) / on-the-fly (dynamic)
  - Global catalog management
- Replication (Parallel reads)
  - Replica synchronization
- Overcome Impedance Mismatch (Polyglot)
  - Mapping between data models
    - *The problem is the overhead incurred for mapping the structures of the origin data model into the structures of the destination data model*
      - Already discussed by the OODBMS community back in the 80s-90s!
        - A single data model: the relational model

Distributed Database

- A distributed database (DDB) is a database where <u>data management</u> is distributed over several nodes in a network.
  - Each node is a database itself
    - Potential heterogeneity
  - Nodes communicate through the network

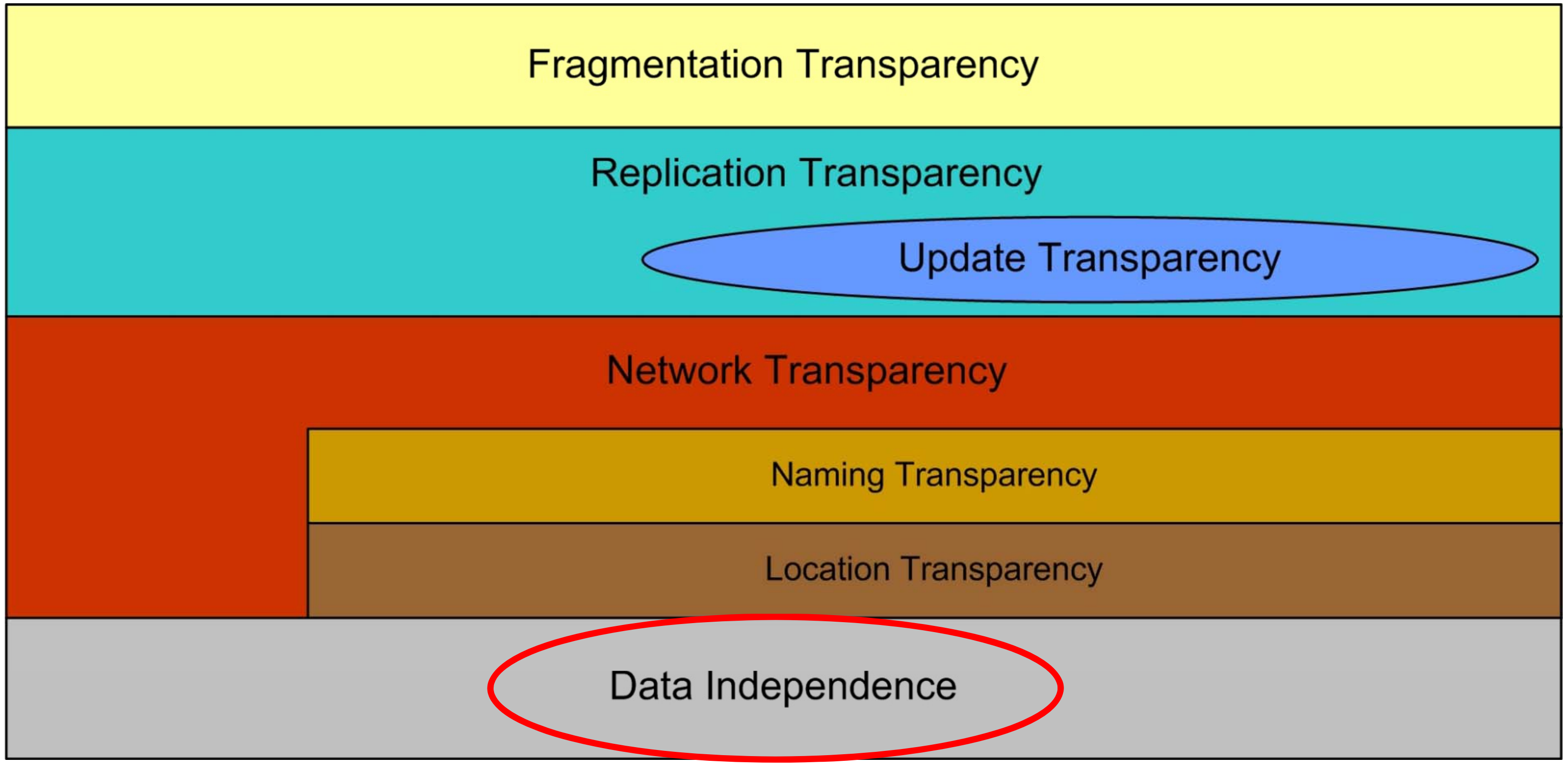




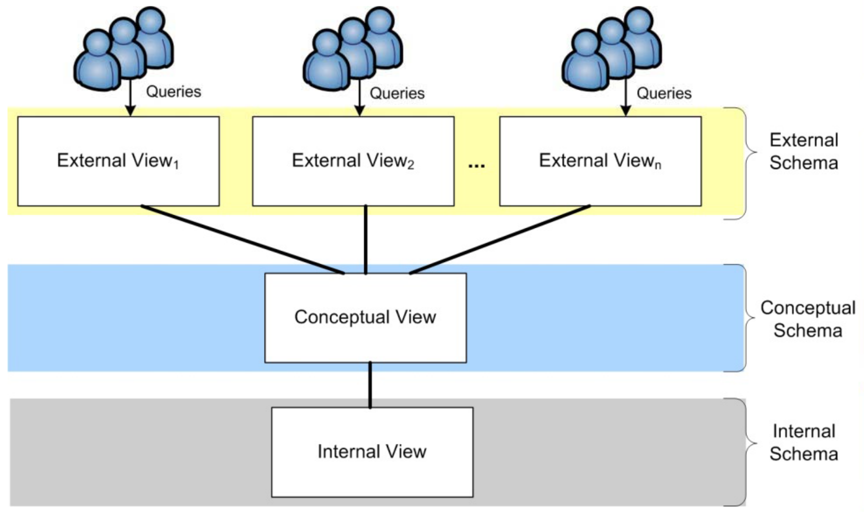
Distributed Architectures

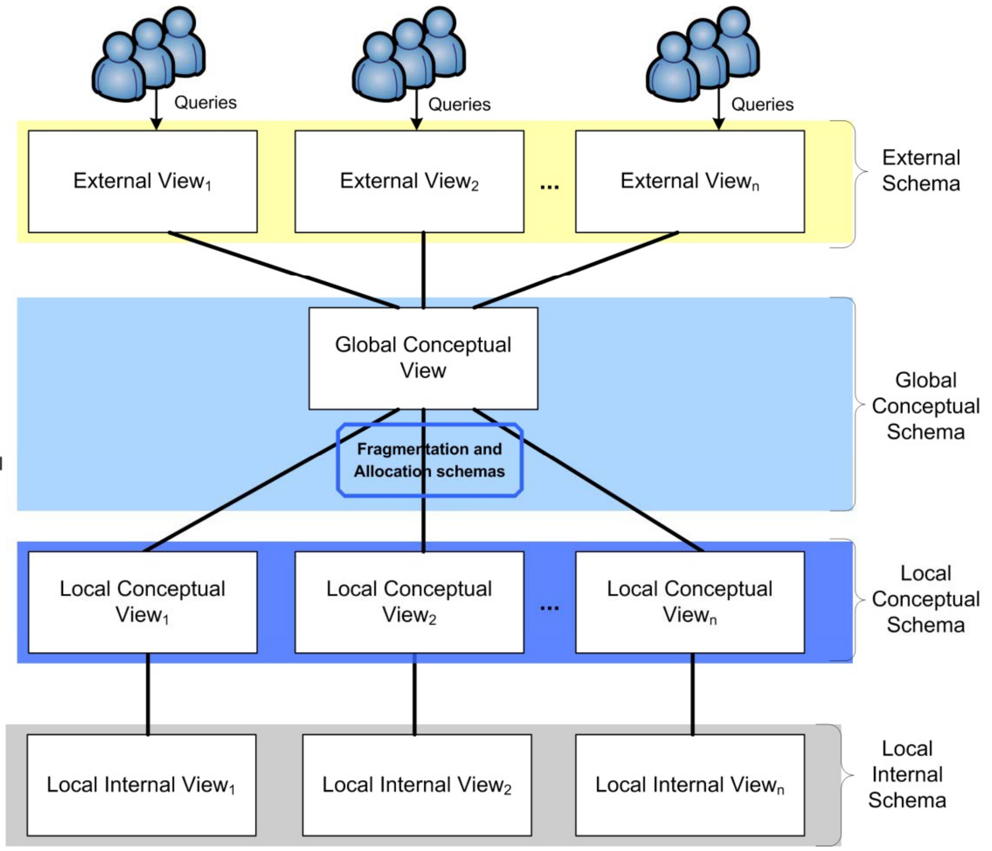
- Main objective: hide implementation (i.e., physical) details to the users
  - Data independency at the logical and physical level must be guaranteed
    - Inherited from centralized DBs (ANSI SPARC)
  - Network transparency
    - Data access must be independent regardless where data is stored
    - Each data object must have a unique name
  - Replication transparency
    - The user must not be aware of the existing replicas
  - Fragmentation transparency
    - The user must not be aware of partitioning

Transparencylayers



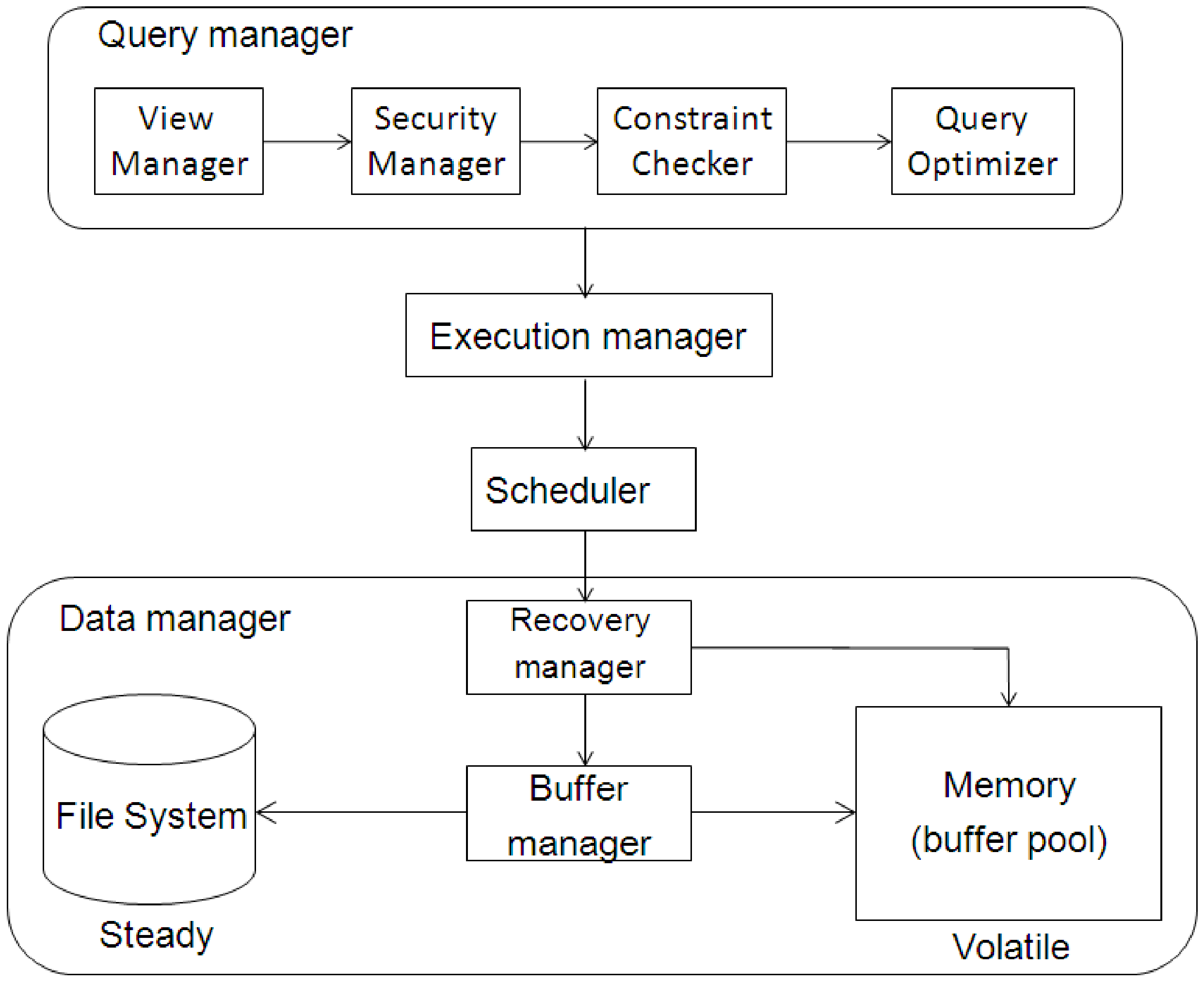
Extended ANSI-SPARC Architecture



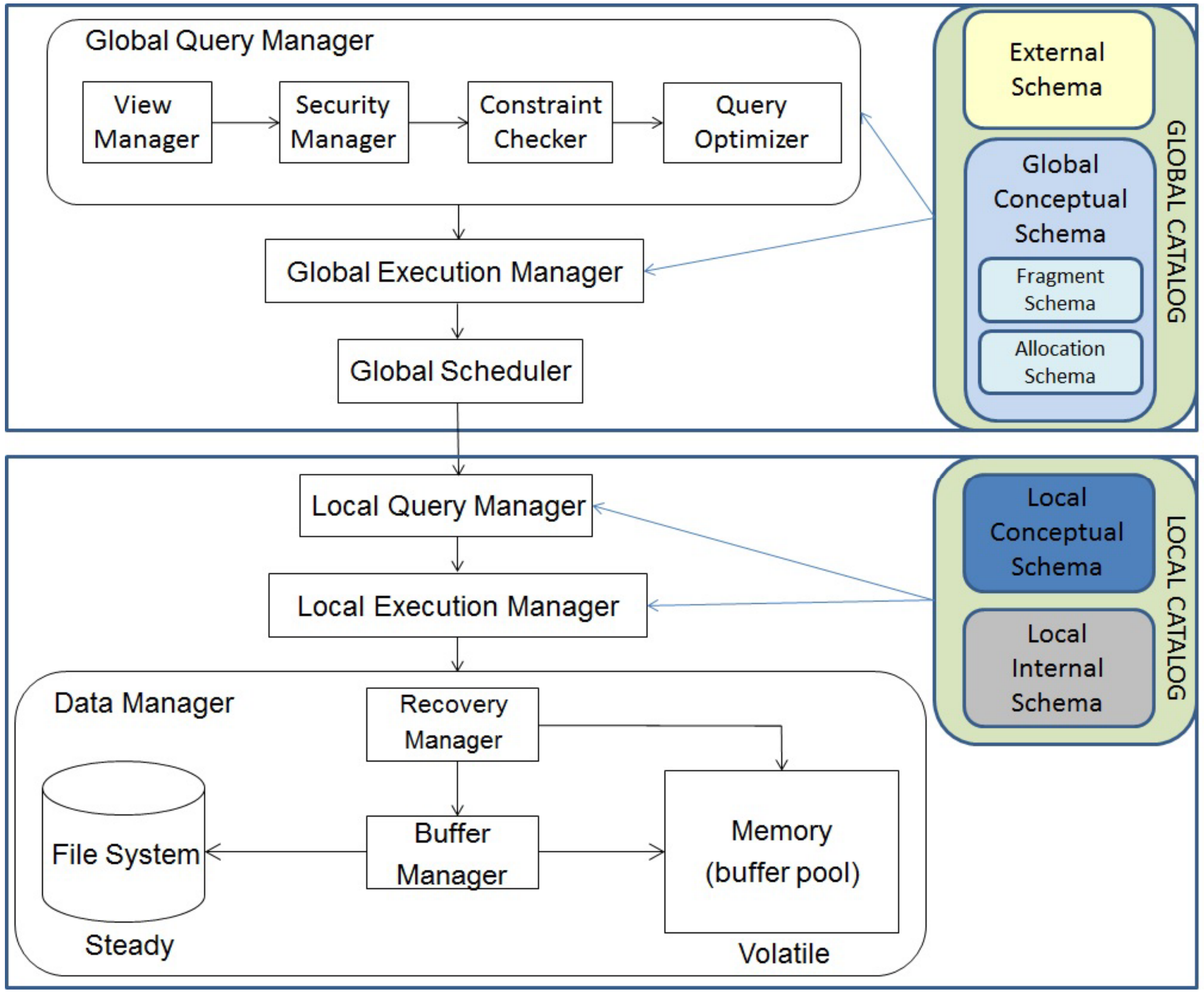


- Global catalog
  - Mappings between ESs – GCS and GCS – LCSs
- Each node has a local catalog
  - Mappings between LCS_i – IS_i

Centralized DBMS Architecture



Distributed DBMS (System-R) Architecture



- Introduce a critical reasoning on the reference architecture

Challenges in data distribution

I. Distributed DB design
   - Data fragments
   - Data replication
   - Node distribution

II. Distributed DB catalog
   - Fragmentation trade-off: Where to place the DB catalog
     - Global or local for each node
     - Centralized in a single node or distributed
     - Single-copy vs. Multi-copy

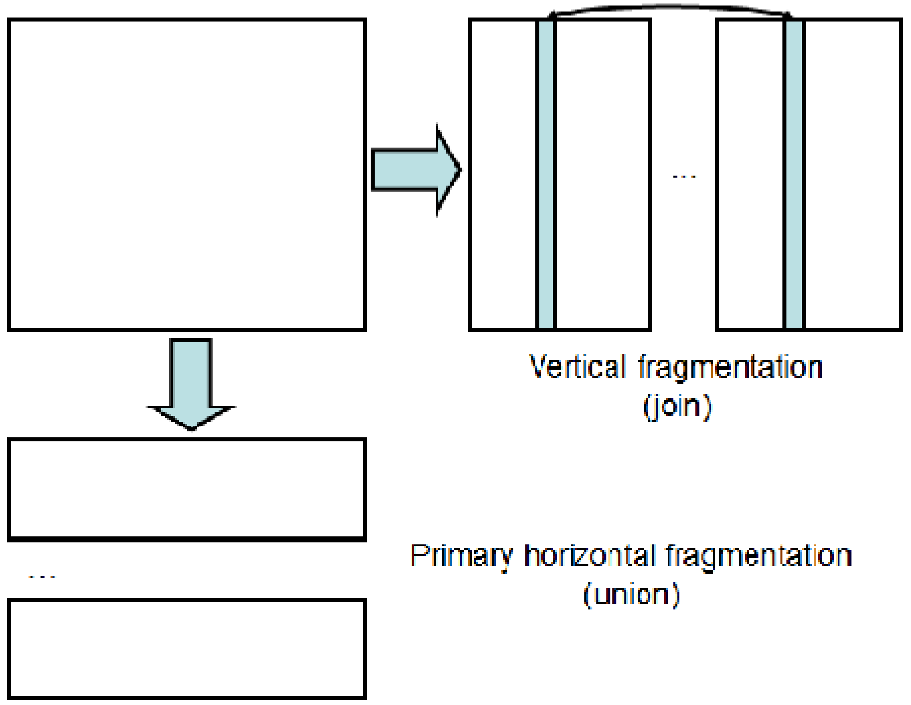
III. Distributed query processing
   - Data distribution / replication
   - Communication overhead

IV. Distributed transaction management
   - How to enforce the ACID properties
     - Replication trade-off: Queries vs. Data consistency between replicas (updates)
     - Distributed recovery system
     - Distributed concurrency control system

V. Security issues
   - Network security

Challenge I: DDB Design

- Given a DB and its workload, how should the DB be split and allocated to sites as to optimize certain objective functions
  - Minimize resource consumption for query processing
- Two main issues:
  - Data fragmentation
  - Data allocation
    - Data replication



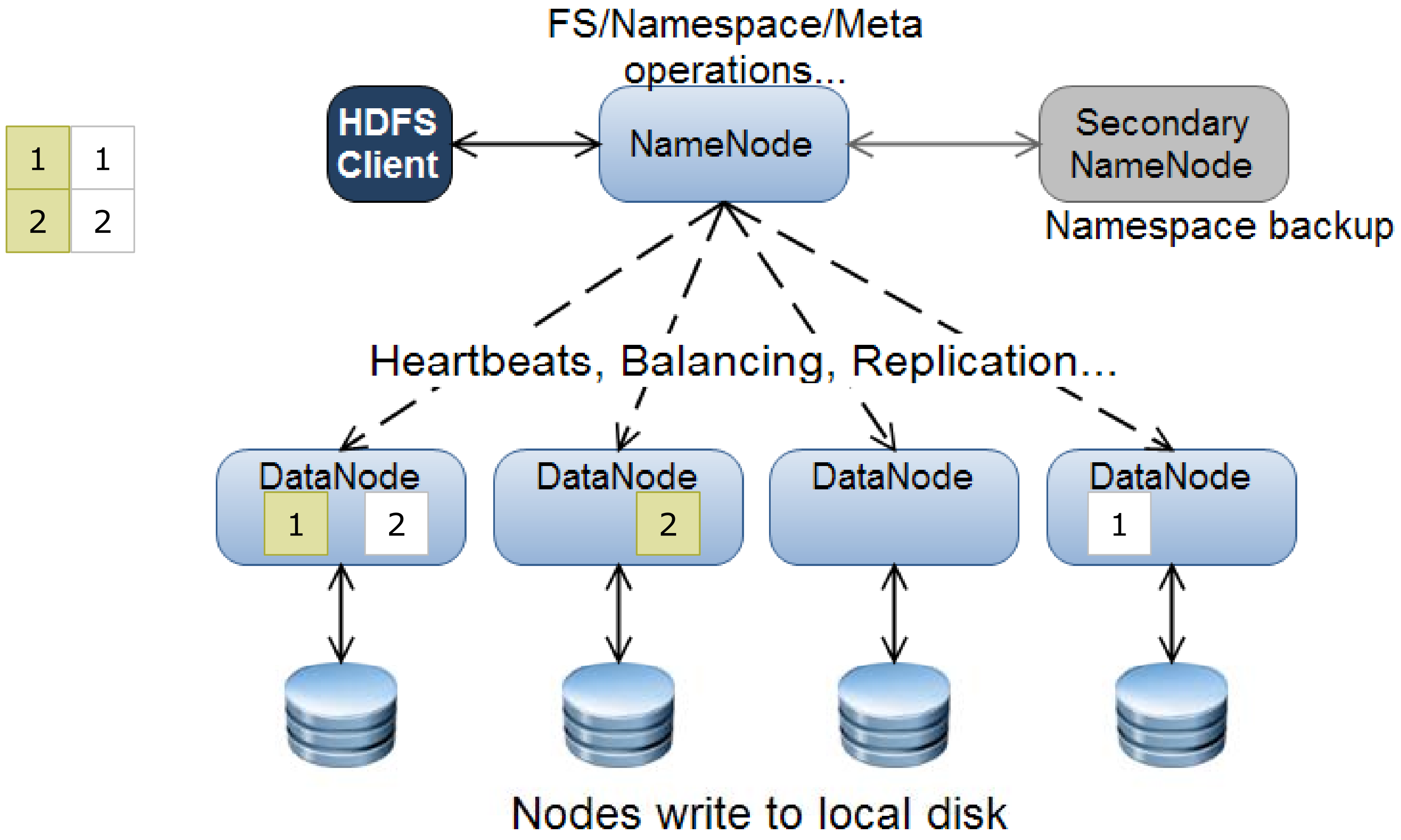
Data Fragmentation

- Fragmentation of data is useful for several reasons…
  - An application typically accesses only a subset of data
  - Different subsets are (naturally) needed at different sites
  - The degree of concurrency is enhanced
    - Facilitates parallelism
  - Fragments likely to be used jointly can be colocated to minimize communication overhead
- However…
  - May lead to poorer performance when multiple fragments need to be joined
  - It becomes more costly to enforce the dependency between attributes in different fragments

Decide Data Allocation

- Given a set of <u>fragments</u>, a set of <u>sites</u> on which a number of <u>applications</u> are running, **allocate** each fragment such that some <u>optimization criterion</u> is met (subject to certain <u>constraints</u>)
- It is known to be a NP-hard problem
  - The optimal solution depends on many factors
    - Location in which the query originates
    - The query processing strategies (e.g., join methods)
  - Furthermore, in a dynamic environment the workload and access pattern may change
- The problem is typically simplified with certain assumptions (e.g., only communication cost considered)
- Typical approaches build *cost models* and any optimization algorithm can be adapted to solve it
  - Heuristics are also available: (e.g., best-fit for non-replicated fragments)
  - Sub-optimal solutions

Hadoop Distributed File System: Example

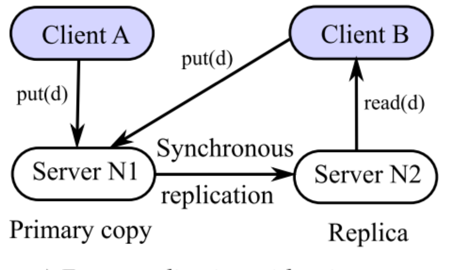

1
1
2
2
FS/Namespace/Meta operations...
HDFS Client
NameNode
Secondary NameNode
Namespace backup
Heartbeats, Balancing, Replication...
DataNode
1
2
DataNode
2
DataNode
DataNode
1
Nodes write to local disk

By Víctor Herrero. Big Data Management & Analytics (UPC School)

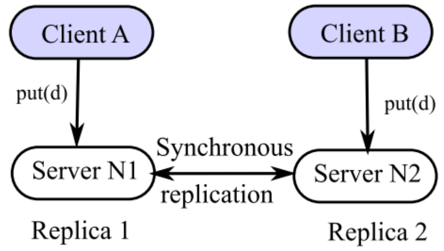
Manage Data Replication

- Replicating fragments improves the system throughput but raises some other issues:
  - Consistency
  - Update performance
- Most used replication protocols
  - Eager – Lazy replication
  - Primary – Secondary versioning

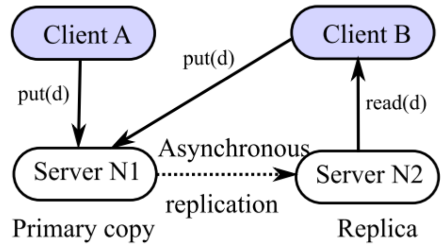
Strong Consistency



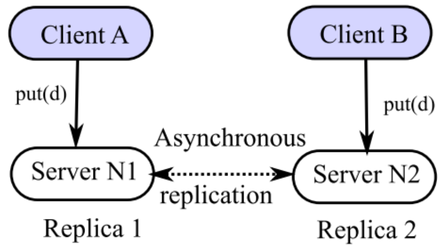
a) Eager replication with primary copy



c) Eager replication, distributed



b) Lazy replication with primary copy
(a.k.a Master-Slave replication)



d) Lazy replication, distributed
(a.k.a. Master-Master replication)

Eventually Consistent

Activity

- *Objective: Understand the consequences behind each data replication strategy*
- *Tasks:*
  1. *(10') By pairs, answer the following questions:*
     1. *Discuss the questions below with your peer*
     2. *What is the most important feature for each scenario?*
  2. *(5') Discussion*

- You are a customer using an e-commerce based on heavy replication (e.g., Amazon):
  - Show a database replication strategy (e.g., sketch it) where you buy an item, but this item does not appear in your basket.
  - You reload the page: the item appears. What happened?
  - You delete an item from your command, and add another one: the basket shows both items. What happened?
  - Will the situation change if you reload the page?

What About the System Scalability?

- How do we define “Scalability”? And “Elasticity”?

The Universal Scalability Law (I)

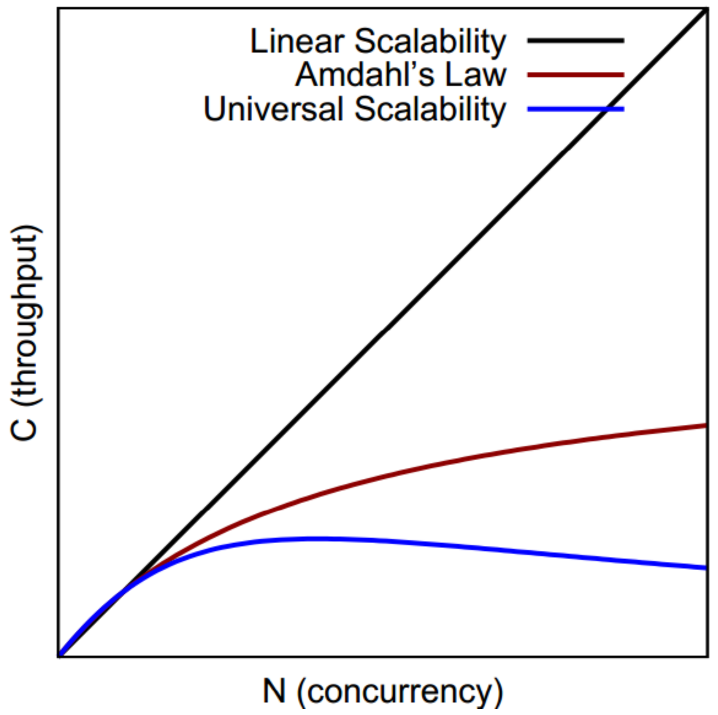
- It can model both Sw or Hw scalability
- The USL is defined as follows:

$$C(N) = \frac{N}{1 + \sigma(N-1) + \kappa N(N-1)}$$

  - C: System's capacity (i.e., throughput) improvement
    - Improvement of queries per second
  - N: System's concurrency
    - (Sw): Number of users / processes active
    - (Hw): Number of CPUs
  - σ: System's contention. Performance degradation due to serial instead of parallel processing
  - κ: System's consistency delay (aka coherency delay). Extra work needed to keep synchronized shared data (i.e., inter-process communication)

The Universal Scalability Law (II)

- If both σ = 0 and κ = 0, we obtain linear scalability
- If κ = 0, it simplifies to Amdahl's law



The USL at Work

- Method:
  - [Step 1] Empirical analysis: Compute C (throughput) for different values of N (concurrency)
  - [Step 2] Perform statistical regression against gathered data (needs some data cooking first)
  - [Step 3] Reverse the transformation to find the σ (contention) and κ (consistency delay) parameters
- How to apply this method step by step:

http://www.percona.com/files/white-papers/forecasting-mysql-scalability.pdf

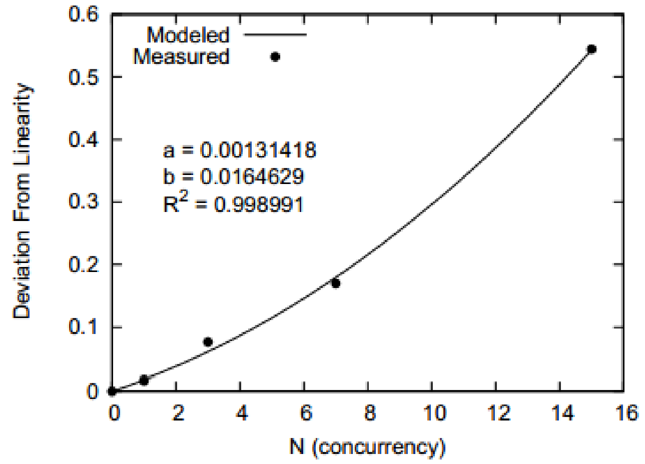
The USL at Work: Example

- System's Setting
  - Percona's MySQL Server with XtraDB
  - Cisco UCS server (2 processors, each with 6 cores and each core can run two threads: 24 threads)
  - 384GB memory

Step 1:

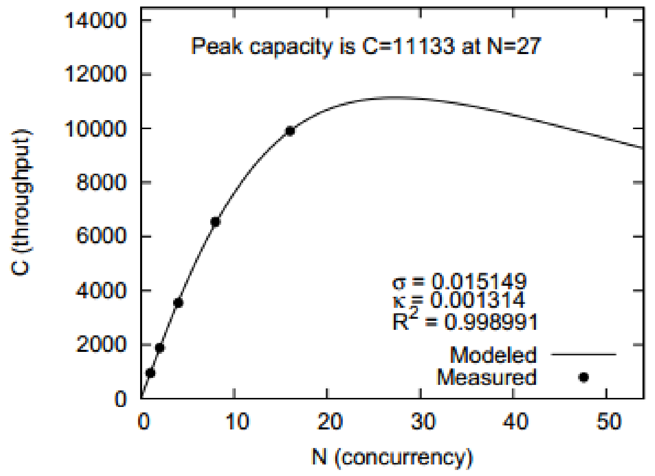
Concurrency (N)	Throughput (C)
1	955.16
2	1878.91
4	3548.68
8	6531.08
16	9897.24

Step 2:



Points are fit in a second-order polynomial: ax^2+bx+0, and a and b are computed

Step 3:



σ and κ are next computed from a and b. Next, we apply the USL formula

Measuring Scalability

- Ideally, scalability should be linear
- Scalability is normally measured in terms of speed-up and scale-up
  - Speed-up: Measures performance when adding Hw for a constant problem size
    - Linear speed-up means that N sites solve in T/N time, a problem solved in T time by 1 site
  - Scale-up: Measures performance when the problem size is altered with resources
    - Linear scale-up means that N sites solve a problem N*T times bigger in the same time 1 site solves the same problem in T time
- The USL shows that linear scalability is hardly achievable
  - σ (contention) could be avoided (i.e., σ = 0) if our code has no serial chunks (everything parallelizable)
  - κ (consistency delay) could be avoided (i.e., κ = 0) if replicas can be synchronized without sending messages

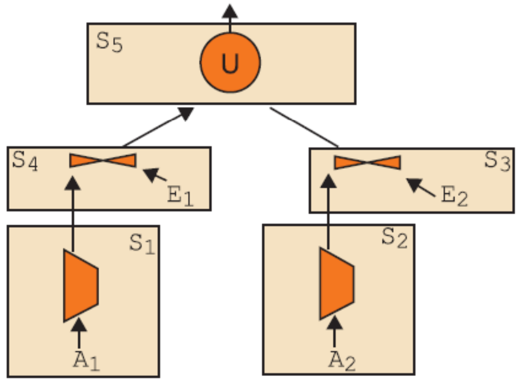
Challenge II: Global Catalog

- Centralized version (@master)
  - Accessing it is a bottleneck
    - Single-point failure
      - May add a mirror
    - Poorer performance
- Distributed version (several *masters*)
  - Replica synchronization
    - Potential inconsistencies

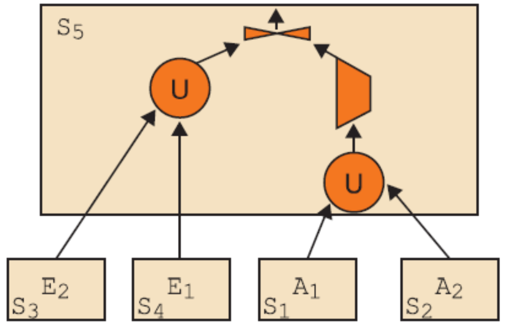
Activity

- Objective: Recognize the difficulties and opportunities behind distributed query processing
- Tasks:
  1. (10') By pairs, answer the following questions:
     I. What are the main differences between these two distributed access plans?
     II. Under which assumptions is one or the other better?
     III. List the new tasks a distributed query optimizer must consider with regard to a centralized version
  2. (5') Discussion

```
SELECT *
FROM employee e, assignedTo a
WHARE e.#emp=a.#emp AND
a.responsability= 'manager';
```



Access Plan A



Acces Plan B

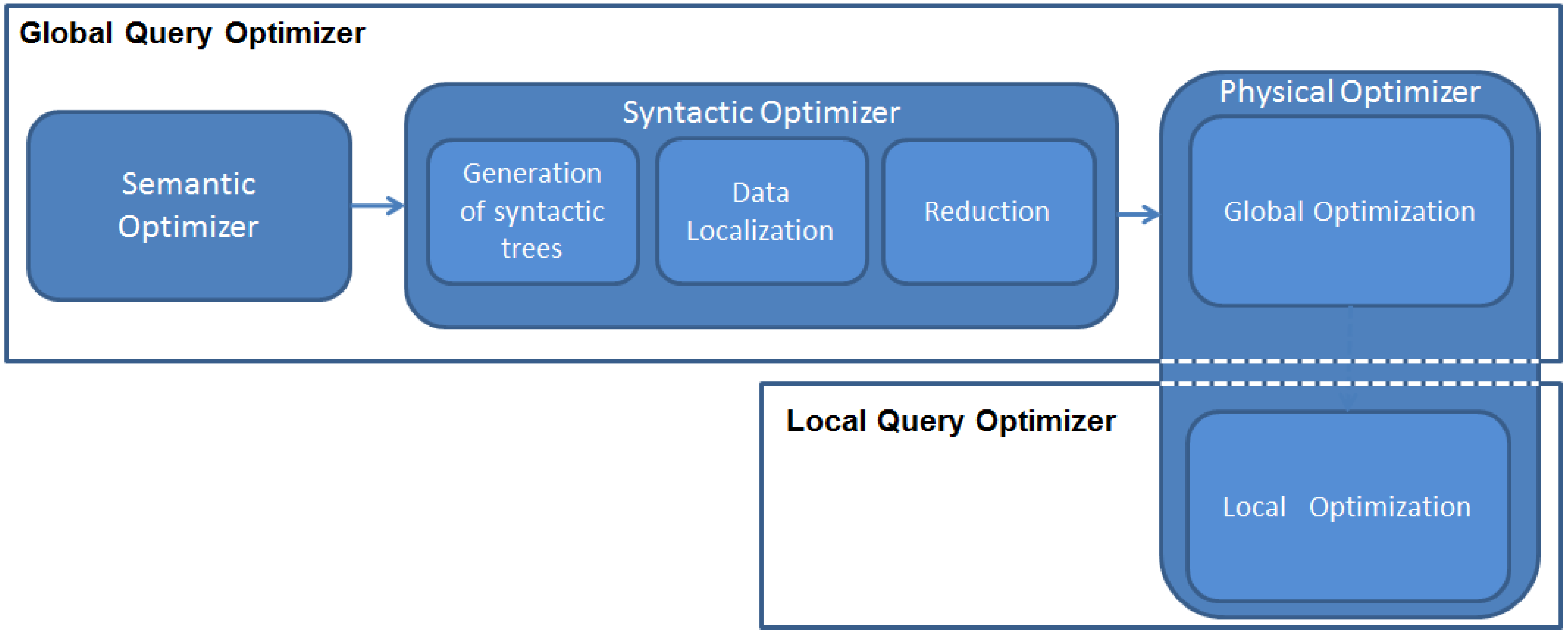
AssignedTo (#emp,#proj, responsibility, fullTime)
- S_1: A_1 = AssignedTo(#emp≤'E3')
- S_2: A_2 = AssignedTo(#emp>'E3')

Employee (#emp, empName, degree)
- S_3: E_2 = Employee(#emp>'E3')
- S_4: E_1 = Employee(#emp≤'E3')

Challenge III: Distributed Query Processing

- Communication cost (data shipping)
  - Not that critical for LAN networks
    - Assuming high enough I/O cost
- Fragmentation / Replication
  - Metadata and statistics about fragments (and replicas) in the global catalog
- Join Optimization
  - Joins order
  - Semijoin strategy
- How to decide the execution plan
  - Who executes what
  - Exploit parallelism (!)

Phases of Distributed Query Processing

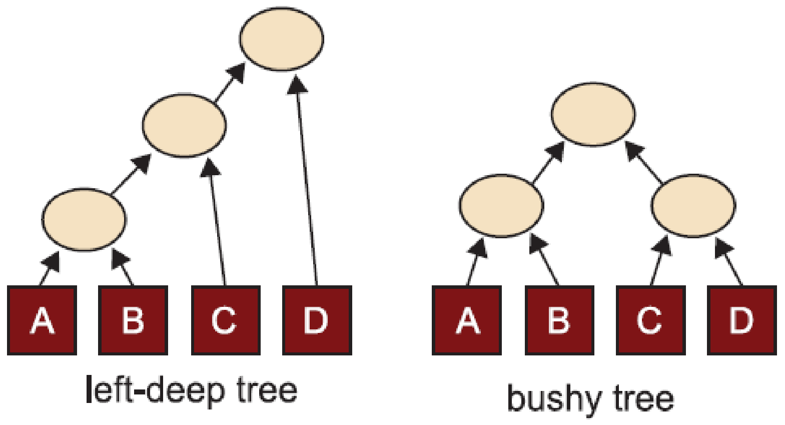


Physical Optimizer

- Transforms an internal query representation into an efficient plan
  - Replaces the logical query operators by specific algorithms (plan operators) and access methods
  - Decides in which order to execute them
- This is done by…
  - Enumerating alternative but equivalent *plans*
    - Dataflow diagram that pipes data through a graph of query operators
  - Estimating their costs
  - Searching for the best solution
    - Using available statistics regarding the physical state of the system

Global Physical Optimizer

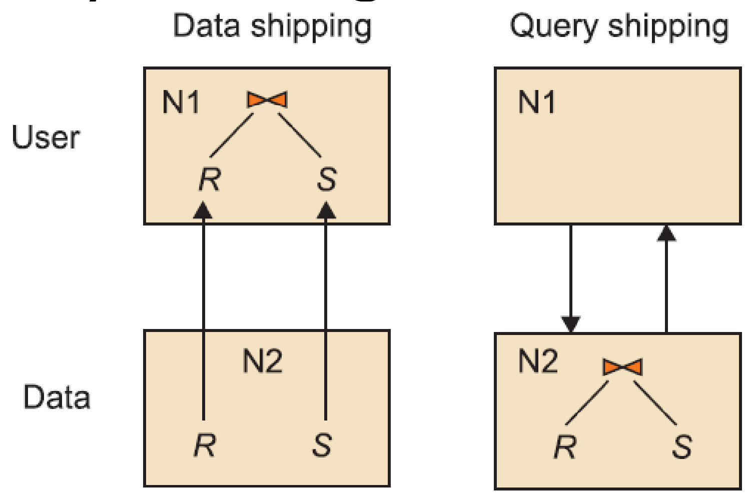
- Generation of Execution Alternatives
  - Ordering
    - Left or right deep trees
    - Bushy trees



  - Site selection (exploit **DATA LOCATION**)
    - Comparing size of the relations
    - More difficult for joins (multi-way joins)
    - Size of the intermediate joins must be considered
  - Algorithms to process the query operators
    - **Parallelism (!)**

Site Selection

- Data shipping
  - The data is retrieved from the stored site to the site executing the query
    - Avoid bottlenecks on frequently used data
- Query shipping
  - The evaluation of the query is delegated to the site where it is stored
    - To avoid transferring large amount of data
- Hybrid strategy



Parallel Query Processing

- Employ parallel hardware effectively (i.e., reduce the response time)
  - Process pieces in different processors
  - Serial algorithms adapted to multi-thread environments
  - Divide input data set into <u>disjoint</u> subsets
- May hurt overall execution time (i.e., throughput)
  - Ideally linear speed-up
    - Additional hardware for a constant problem size
      - Addition of computing power should yield proportional increase in performance
        - N nodes should solve the problem in 1/N time
  - Ideally linear scale-up
    - Problem size is altered with the resources
      - Sustained performance for a linear increase in both size and workload, and number of nodes
        - N nodes should solve a problem N times bigger in the same time

Kinds of Parallelism

- Inter-query
- Intra-query
  - Intra-operator
    - Unary
      - Static partitioning
    - Binary
      - Static or dynamic partitioning
  - Inter-operator
    - Independent
    - Pipelined
      - Demand driven (pull)
      - Producer driven (push)

Choosing the Best Execution Plan

- Response Time
  - Time needed to execute a query (user’s clock)
  - Benefits from parallelism
    - Operations divided into N operations
- Total Cost Model
  - Sum of local cost and communication cost
    - Local cost
      - Cost of central unit processing (#cycles),
      - Unit cost of I/O operation (#I/O ops)
    - Communication cost
      - Commonly assumed it is linear in the number of bytes transmitted
      - Cost of initiating a message and sending a message (#messages)
      - Cost of transmitting one byte (#bytes)
  - Knowledge required
    - Size of elementary data units processed
    - Selectivity of operations to estimate intermediate results
  - Does not account the usage of parallelisms (!)
- Hybrid solutions

Cost Model Example

- Parameters:
  - Local processing:
    - Average CPU time to process an instance (T_{cpu})
    - Number of instances processed (#inst)
    - I/O time per operation ($T_{I/O}$)
    - Number of I/O operations (#I/Os)
  - Global processing:
    - Message time (T_{Msg})
    - Number of messages issued (#msgs)
    - Transfer time (send a byte from one site to another) (T_{TR})
    - Number of bytes transferred (#bytes)
- It could also be expressed in terms of packets
- Calculations:

Resources = T_{cpu} * #inst + $T_{I/O}$ * #I/Os + T_{Msg} * #msgs + T_{TR} * #bytes
Respose Time = T_{cpu} * $seq_{\#inst}$ + $T_{I/O}$ * $seq_{\#I/Os}$ + T_{Msg} * $seq_{\#msgs}$ + T_{TR} * $seq_{\#bytes}$

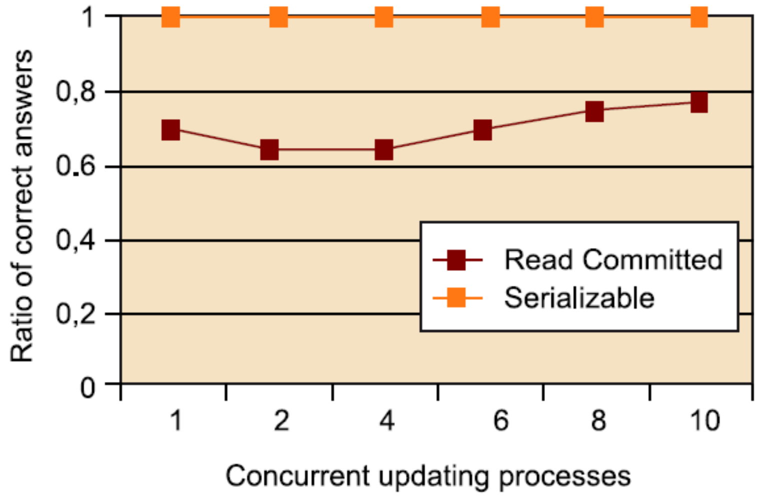
Challenge IV: Distributed Tx Management

- ACID properties are not always necessary
  - All can be relaxed
- Relaxing Consistency and Durability
  - Entails data loss
  - Save synchronization time
- Relaxing Atomicity and Isolation
  - Generate interferences
  - Save locks and contention

Trade-Off: Performance Vs. Consistency

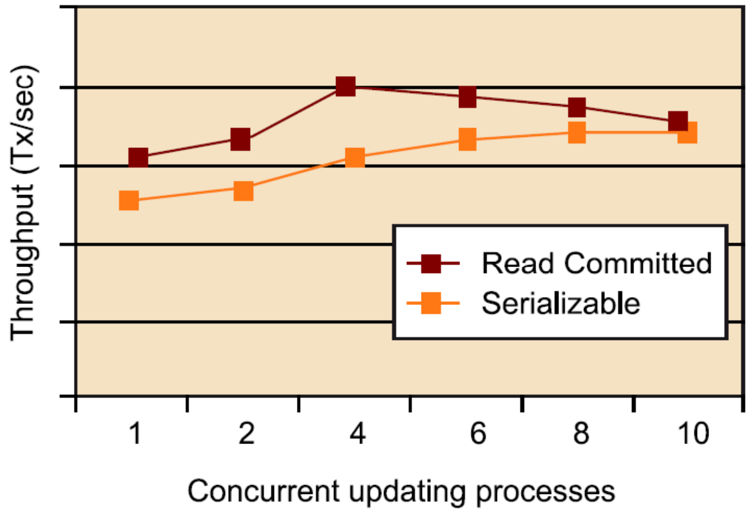
Consistency
(Ratio of correct answers)

Percentage of correct results depending on the number of concurrent transactions



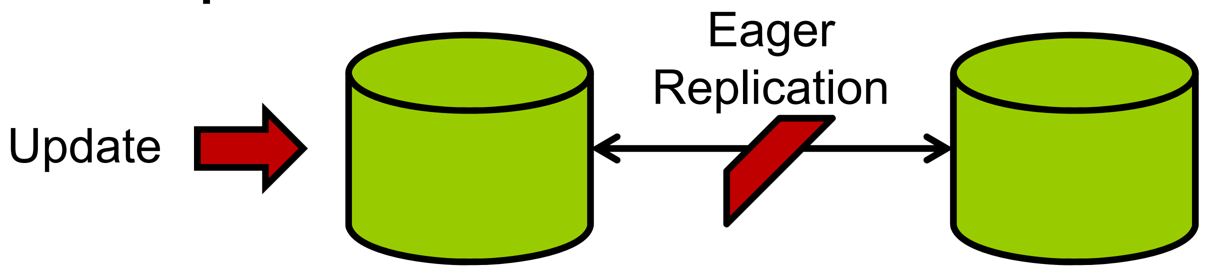
Performance
(System throughput)

Throughput (transactions per second) depending on the number of concurrent transactions.



CAP Theorem

- Eric Brewer. CAP Theorem: *any networked shared-data system can have at most two of three desirable properties*:
  - consistency (C) equivalent to having a single up-to-date copy of the data;
  - high availability (A) of that data (for updates); and
  - tolerance to network partitions (P)
- Example:

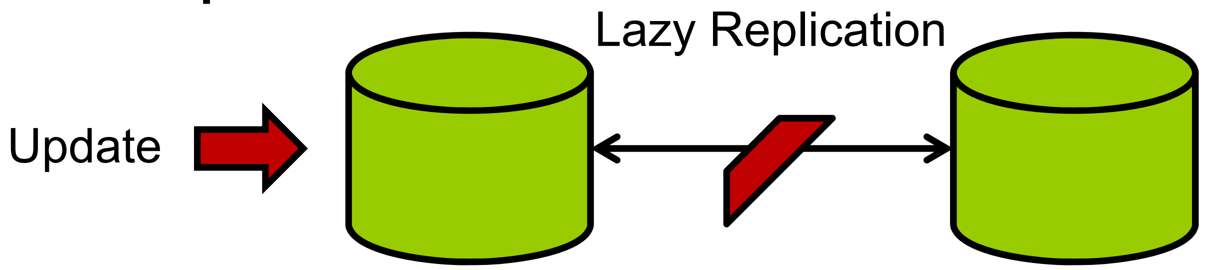


ACID

C ->	OK
A ->	NO
P ->	OK

CAP Theorem

- Eric Brewer. CAP Theorem: *any networked shared-data system can have at most two of three desirable properties*:
  - consistency (C) equivalent to having a single up-to-date copy of the data;
  - high availability (A) of that data (for updates); and
  - tolerance to network partitions (P).
- Example:

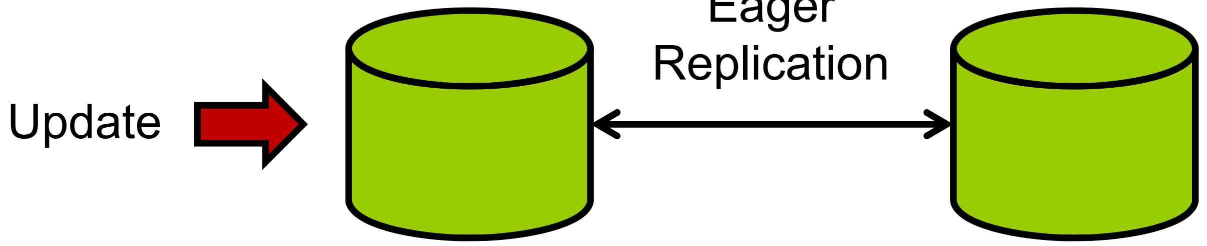


BASE

C ->	NO
A ->	OK
P ->	OK

CAP Theorem

- Eric Brewer. CAP Theorem: *any networked shared-data system can have at most two of three desirable properties*:
  - consistency (C) equivalent to having a single up-to-date copy of the data;
  - high availability (A) of that data (for updates); and
  - tolerance to network partitions (P).
- Example:



????

C ->	OK
A ->	OK
P ->	NO

CAP Theorem Revisited

- The CAP theorem is not about choosing two out of the three **forever and ever**
  - Distributed systems are not always partitioned
- Without partitions: CA
- Otherwise…
  - Detect a partition
    - Normally by means of latency (time-bound connection)
  - Enter an explicit partition mode limiting some operations choosing either:
    - CP (i.e., ACID by means of e.g., 2PCP or PAXOS) or,
      - If a partition is detected, the operation is aborted
    - AP (i.e., BASE)
      - The operation goes on and we will tackle this next
  - If AP was chosen, enter a recovery process commonly known as *partition recovery* (e.g., compensate mistakes and get rid of inconsistencies introduced)
    - Achieve consistency: Roll-back to consistent state and apply ops in a deterministic way (e.g., using time-stamps)
      - Reduce complexity by only allowing certain operations (e.g., Google Docs)
      - Commutative operations (concatenate logs, sort and execute them)
    - Repair mistakes: Restore invariants violated
      - Last writer wins

On the Need of a New Architecture

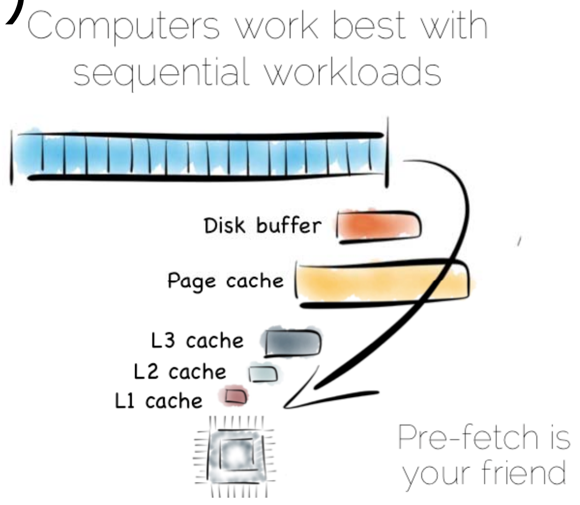
- Distribution is needed to overcome the challenges presented
  - To provide scalability, efficiency (by means of parallelism), reliability / availability
    - But RDDBMS do not meet them (RDDBMS bottlenecks)
- **CAP theorem formulation**: There is a trade-off in distributed systems; either availability or consistency can be always guaranteed (not both)
  - RDDBMS choose consistency
  - NOSQL systems, most of the times, choose availability

CAP Theorem: Examples

- Yahoo's PNUTS (AP): Assumes data is partitioned according to where the user is
  - Remote copies are maintained asynchronously
  - The master copy is local (to decrease latency)
- Facebook (AP): Unique master
  - Master copy in always one location (higher latency)
  - After 20'' the user's traffic reverts to a closer copy (which hopefully, by that time, should reflect the change)
- HTML5 (AP): Full Availability
  - On-client persistent storage (master copy)
  - Allows to go offline **->** massive recovery process
- Google (CP): Considers primary partitions (data centers in USA, Germany, Sweden, etc.)
  - CA within partitions
  - CP between partitions using PAXOS (e.g., Megastore)

Most Typical Solutions (I)

- Random Vs. Sequential Reads
- Take the most out of databases by boosting sequential reads
  - Enables pre-fetching
  - Option to maximize the effective read ratio (by a good DB design)



Most Typical Solutions (II)

- Sequential reads
  - Key design
- Primary indexes to implement the global catalog
  - Distributed Tree: WiredTiger, HBase, etc.
  - Consistent Hashing: Voldemort, MongoDB (until 2.X), etc.
- Bloom filters to avoid distributed look ups
- In-memory processing
- Columnar block iteration: Vertical fragmentation + fixed-size values + compression (run length encoding)
  - Heavily exploited by column-oriented databases
  - Only for read-only workloads

New Architectures: NewSQL

(idea) For OLTP systems RDBMS can also be outperformed

- Main memory DB
  - A DB less than 1Tb fits in memory
    - 20 nodes x 32 Gb (or more) costs less than 50,000US$
  - Undo log is in-memory and discarded on commit
- One thread systems
  - Perform incoming SQL commands to completion, without interruption
    - One transaction takes less than 1ms
  - No isolation needed
- Grid computing
  - Enjoy horizontal partitioning and parallelism
    - Add new nodes to the grid without going down
- High availability
  - Cannot wait for the recovery process
    - Multiple machines in a Peer-To-Peer configuration
- Reduce costs
  - Human costs are higher than Hw and Sw
    - An expert DBA is expensive and rare
  - Alternative is brute force
    - Automatic horizontal partitioning and replication
    - Execute queries at any replica and updates to all of them
- Optimize queries at compile time

Summary

- Big Data Management Goals
- Distributed Database Systems Architecture
- Distributed Database Design
  - Fragmentation
  - Replication
- Distributed Catalog
- Distributed Query Processing
  - Kinds of Parallelism
- Distributed Transaction Processing
  - CAP Theorem
- Universal Scalability Law
- NewSQL

Bibliography

- M. T. Özsu and P. Valduriez. *Principles of Distributed Database Systems*, 3rd Ed. Springer, 2011
- L. Liu, M.T. Özsu (Eds.). *Encyclopedia of Database Systems*. Springer, 2009
- N. J. Gunther. *A Simple Capacity Model of Massively Parallel Transaction Systems*. CMG National Conference, 1993
- M. Stonebraker, S. Madden, D. J. Abadi, S. Harizopoulos, N. Hachem, P. Helland. *The End of an Architectural Era (It's Time for a Complete Rewrite)*.