Week 10 Lectures

Beyond RDBMSs

Future of Database

2/52

Core "database" goals:

- deal with very large amounts of data (terabytes, petabyes, ...)
- very-high-level languages (deal with big data in uniform ways)
- query execution (if evaluation too slow ⇒ useless)

At the moment (and for the last 20 years) RDBMSs dominate ...

- simple/clean data model, backed up by theory
- high-level language for accessing data
- 40 years development work on RDBMS engine technology

RDBMSs work well in domains with uniform, structured data.

... Future of Database

3/52

Limitations/pitfalls of RDBMSs:

- NULL is ambiguous: unknown, not applicable, not supplied
- "limited" support for constraints/integrity and rules
- no support for uncertainty (data represents *the* state-of-the-world)
- data model too simple (e.g. no direct support for complex objects)
- query model too rigid (e.g. no approximate matching)
- continually changing data sources not well-handled
- data must be "molded" to fit a single rigid schema
- database systems must be manually "tuned"
- do not scale well to some data sets (e.g. Google, Telco's)

... Future of Database

4/52

How to overcome (some of) these limitations?

Extend the relational model ...

- add new data types and query ops for new applications
- deal with uncertainty/inaccuracy/approximation in data

Replace the relational model ...

- object-oriented DBMS ... OO programming with persistent objects
- XML DBMS ... all data stored as XML documents, new query model
- application-effective data model (e.g. *(key,value)* pairs)

Performance ...

- new query algorithms/data-structures for new types of queries
- DBMSs that "tune" themselves

... Future of Database

5/52

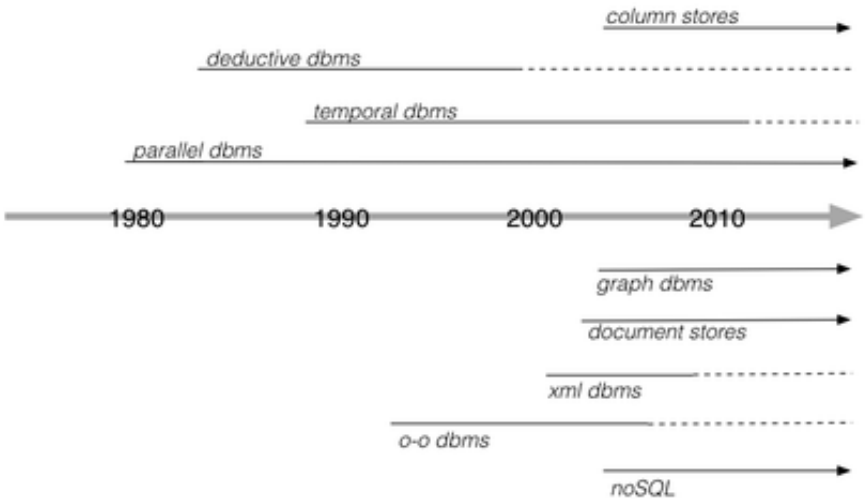
An overview of the possibilities:

- *"classical" RDBMS* (e.g. PostgreSQL, Oracle, SQLite)
- *parallel DBMS* (e.g. XPRS)
- *distributed DBMS* (e.g. Cohera)
- *deductive databases* (e.g. Datalog)
- *temporal databases* (e.g. MariaDB)
- *column stores* (e.g. C-Store?)
- *object-oriented DBMS* (e.g. ObjectStore)
- *key-value stores* (e.g. Redis, DynamoDB)
- *wide column stores* (e.g. Cassandra, Scylla, HBase)
- *graph databases* (e.g. Neo4J, Datastax)
- *document stores* (e.g. MongoDB, Couchbase)
- *search engines* (e.g. Google, Solr)

... Future of Database

6/52

Historical perspective



Big Data

7/52

Some modern applications have massive data sets (e.g. Google)

- far too large to store on a single machine/RDBMS
- query demands far too high even if could store in DBMS

Approach to dealing with such data

- distribute data over large collection of nodes (also, redundancy)
- provide computational mechanisms for distributing computation

Often this data does not need full relational selection

- represent data via *(key,value)* pairs
- unique *key*s can be used for addressing data
- *values* can be large objects (e.g. web pages, images, ...)

... Big Data

8/52

Popular computational approach to Big Data: *map/reduce*

- suitable for widely-distributed, very-large data
- allows parallel computation on such data to be easily specified
- distribute (map) parts of computation across network
- compute in parallel (possibly with further *map*ping)
- merge (reduce) multiple results for delivery to requestor

Some Big Data proponents see no future need for SQL/relational ...

- depends on application (e.g. hard integrity vs eventual consistency)

Humour: Parody of noSQL fans (strong language warning)

Information Retrieval

9/52

DBMSs generally do precise matching (although `like`/regexps)

Information retrieval systems do approximate matching.

E.g. documents containing these words (Google, etc.)

Also introduces notion of "quality" of matching
(e.g. tuple T_1 is a *better* match than tuple T_2)

Quality also implies *ranking* of results.

Much activity in incorporating IR ideas into DBMS context.

Goal: support database exploration better.

Multimedia Data

10/52

Data which does not fit the "tabular model":

- image, video, music, text, ... (and combinations of these)

Research problems:

- how to specify queries on such data? ($image_1 \cong image_2$)
- how to "display" results? (synchronize components)

Solutions to the first problem typically:

- extend notions of "matching"/indexes for querying
- require sophisticated methods for capturing data features

Sample query: find other songs *like* this one?

Uncertainty

11/52

Multimedia/IR introduces approximate matching.

In some contexts, we have approximate/uncertain data.

E.g. witness statements in a crime-fighting database

"I think the getaway car was red ... or maybe orange ..."

"I am 75% sure that John carried out the crime"

Work by Jennifer Widom at Stanford on the *Trio* system

- extends the relational model (ULDB)
- extends the query language (TriQL)

Stream Management Systems

12/52

Makes one addition to the relational model

- *stream* = infinite sequence of tuples, arriving one-at-a-time

Applications: news feeds, telecomms, monitoring web usage, ...

RDBMSs: run a variety of queries on (relatively) fixed data

StreamDBs: run fixed queries on changing data (stream)

Approaches:

- *window* = relation formed from a stream via a rule
- *stream data type* = build new stream-specific operations

Graph Data

13/52

Uses *graphs* rather than tables as basic data structure tool.

Applications: complex data representation, via "flexible" objects, e.g. XML

Graph nature of data changes query model considerably.

(e.g. Xquery language, high-level like SQL, but different operators, etc.)

Implementing graphs in RDBMSs is often inefficient.

Research problem: query processing for XML data.

Dispersed Databases

14/52

Characteristics of dispersed databases:

- very large numbers of small processing nodes
- data is distributed/shared among nodes

Applications: environmental monitoring devices, "intelligent dust", ...

Research issues:

- query/search strategies (how to organise query processing)
- distribution of data (trade-off between centralised and diffused)

Less extreme versions of this already exist:

- grid and cloud computing
- database management for mobile devices

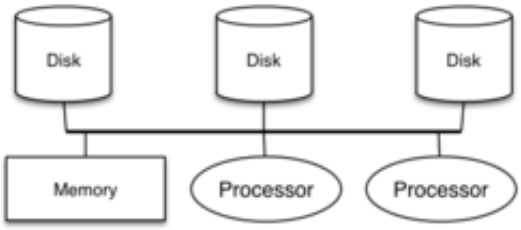
Parallelism in Databases

Parallel DBMSs

16/52

The discussion so far has revolved around systems

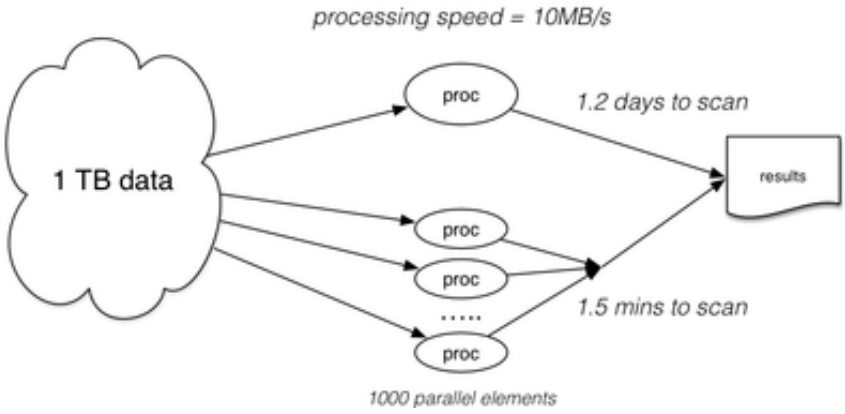
- with a single or small number of processors
- accessing a single memory space
- getting data from one or more disk devices



... Parallel DBMSs

17/52

Why parallelism? ... Throughput!



... Parallel DBMSs

18/52

DBMSs lend are a success story in application of parallelism

- can process many data elements (tuples) at the same time
- can create pipelines of query evaluation steps
- don't require special hardware
- can hide paralleism within the query evaluator
  - application programmers don't need to change habits

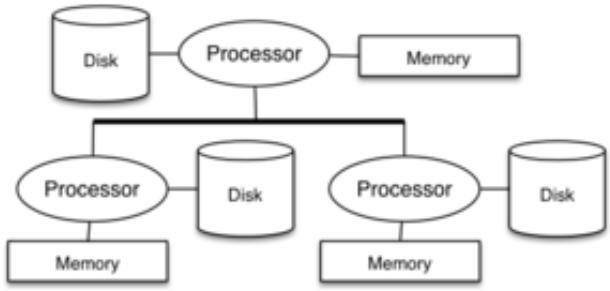
Compare this with effort to do parallel programming.

Parallel Architectures

19/52

Types: *shared memory*, *shared disk*, *shared nothing*

Example shared-nothing architecture:



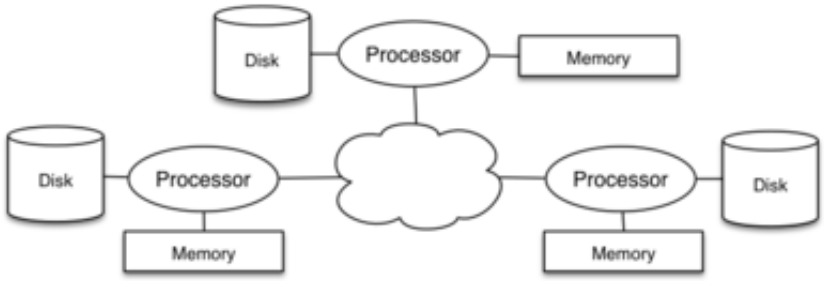
Typically same room/LAN (data transfer cost ~ 100's of µsecs .. msecs)

Distributed Architectures

20/52

Distributed architectures are ...

- effectively shared-nothing, on a global-scale network



Typically on the Internet (data transfer cost ~ secs)

Parallel Databases (PDBs)

21/52

Parallel databases provide various forms of parallelism ...

- process parallelism can speed up query evaluation
- processor parallelism can assist in speeding up memory ops
- processor parallelism introduces cache coherence issues
- disk parallelism can assist in overcoming latency
- disk parallelism can be used to improve fault-tolerance (RAID)
- one limiting factor is congestion on communication bus

... Parallel Databases (PDBs)

22/52

Types of parallelism

- *pipeline parallelism*
  - multi-step process, each processor handles one step
  - run in parallel and pipeline result from one to another
- *partition parallelism*
  - many processors running in parallel
  - each performs same task on a subset of the data

- results from processors need to be merged

Data Storage in PDBs

23/52

Consider each table as a collection of pages ...

Page addressing on single processor/disk: *(Table, File, Page)*

- *Table* maps to a set of files (e.g. named by tableID)
- *File* distinguishes primary/overflow files
- *PageNum* maps to an offset in a specific file

If multiple nodes, then addressing depends how data distributed

- partitioned: *(Node, Table, File, Page)*
- replicated: *({Nodes}, Table, File, Page)*

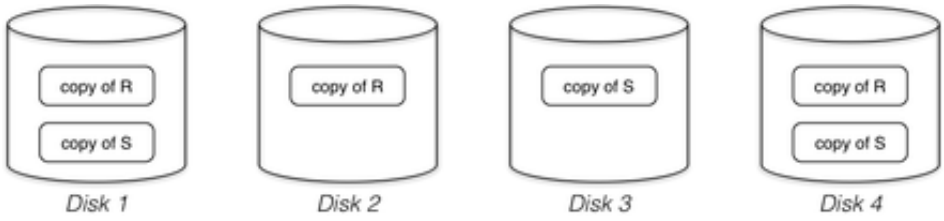
... Data Storage in PDBs

24/52

Assume that each table/relation consists of pages in a file

Can distribute data across multiple storage devices

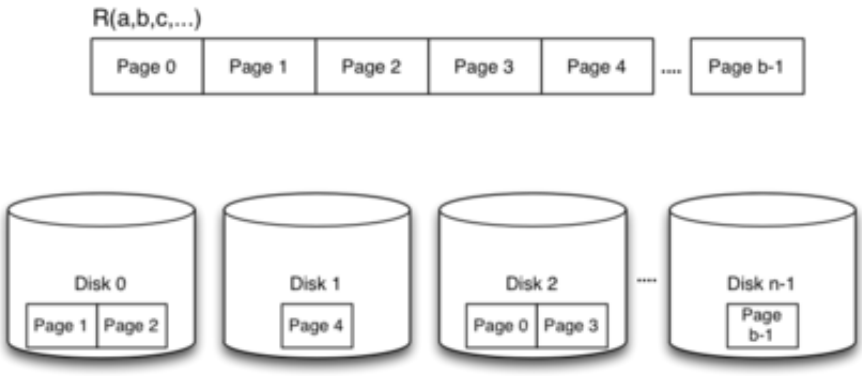
- duplicate all pages from a relation (replication)
- store some pages on one store, some on others (partitioning)



... Data Storage in PDBs

25/52

Data-partitioning example:



... Data Storage in PDBs

26/52

Assume that partitioning is based on one attribute

Data-partitioning strategies for one table on *n* nodes:

- *round-robin*, *hash-based*, *range-based*

Round-robin partitioning

- cycle through nodes, new tuple added on "next" node
- e.g. i^{th} tuple is placed on $(i \bmod n)^{th}$ node
- balances load on nodes; no help for querying

... Data Storage in PDBs

27/52

Hash partitioning

- use hash value to determine which node and page
- e.g. *i = hash(tuple)* so tuple is placed on i^{th} node
- helpful for equality-based queries ? not really

Range partitioning

- ranges of attr values are assigned to processors
- e.g. values 1-10 on $node_0$, 11-20 on $node_1$, ..., 99-100 $node_{n-1}$
- potentially helpful for range-based queries

In both cases, data skew may lead to unbalanced load

Parallelism in DB Operations

28/52

Different types of parallelism in DBMS operations

- intra-operator parallelism
  - get all machines working to compute a given operation (scan, sort, join)
- inter-operator parallelism
  - each operator runs concurrently on a different processor (exploits pipelining)
- Inter-query parallelism
  - different queries run on different processors

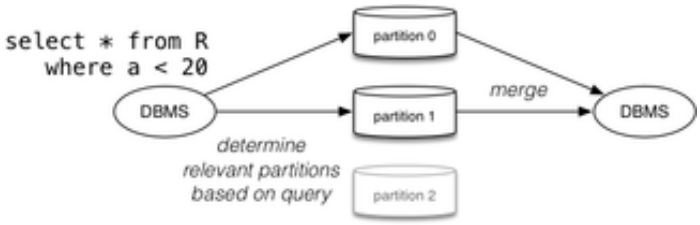
... Parallelism in DB Operations

29/52

Parallel scanning

- scan partitions in parallel and merge results
- selection may allow us to ignore some partitions
  (e.g. for range and hash partitioning)
- can build indexes on each partition

Effectiveness depends on query type vs partitioning type



... Parallelism in DB Operations

30/52

Parallel sorting

- scan in parallel, range-partition during scan
- pipeline into local sort on each processor
- "merge" sorted partitions in order

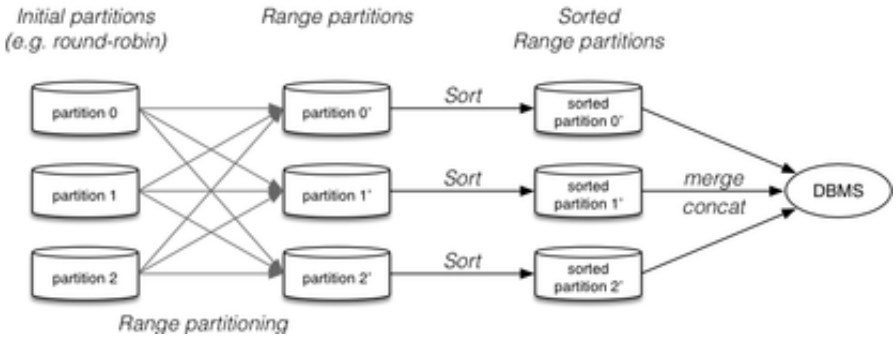
Potential problem:

- data skew because of unfortunate choice of partition points
- resolve by initial data sampling to determine partitions

... Parallelism in DB Operations

31/52

Parallel sort:



... Parallelism in DB Operations

32/52

Parallel nested loop join

- each outer tuple needs to examine each inner tuple
- but only if it could potentially join
- range/hash partitioning reduce partitions to consider

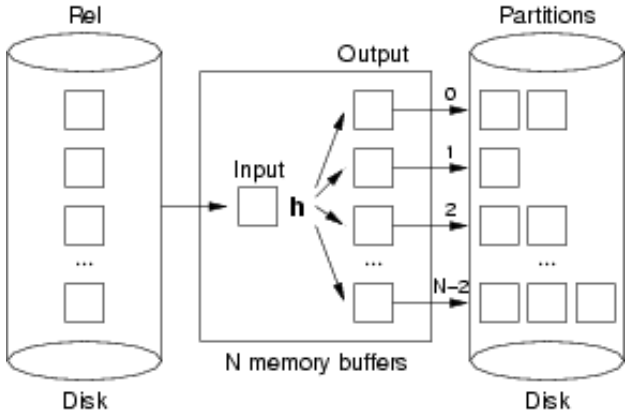
Parallel sort-merge join

- as noted above, parallel sort gives range partitioning
- merging partitioned tables has no parallelism (but is fast)

... Parallelism in DB Operations

33/52

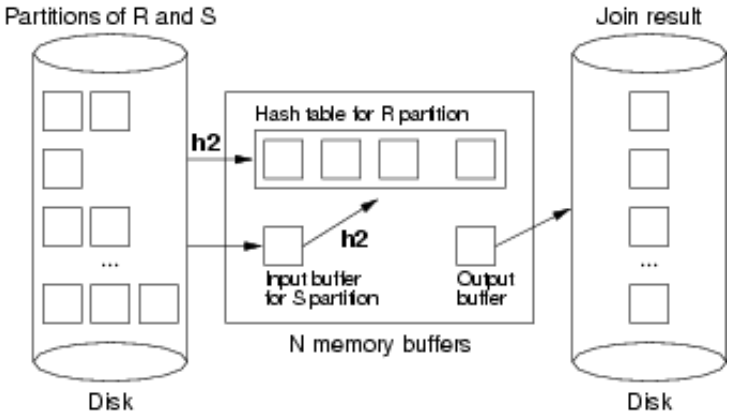
Recall hash join



... Parallelism in DB Operations

34/52

Hash join phase 2



... Parallelism in DB Operations

35/52

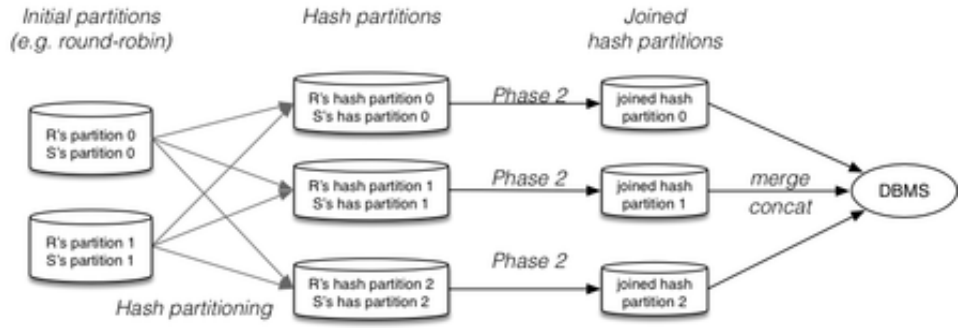
Parallel hash join

- distribute partitions to different processors
- partition 0 of R goes to same site as partition 0 of S
- join phase can be done in parallel on each processor
- then results need to be merged
- very effective for equijoin

... Parallelism in DB Operations

36/52

Parallel hash join:



PostgreSQL and Parallelism

37/52

PostgreSQL assumes

- shared memory space accessable to all back-ends
- files for one table are located on one disk

PostgreSQL allows

- data to be distributed across multiple disk devices

So could run on ...

- shared-memory, shared-disk architectures
- hierarchical architectures with distributed virtual memory

... PostgreSQL and Parallelism

38/52

PostgreSQL can provide

- multiple servers running on separate nodes
- application #1: high availability
  - "standby" server takes over if primary server fails
- application #2: load balancing
  - several servers can be used to provide same data
  - direct queries to least loaded server

Both need *data synchronisation* between servers

PostgreSQL uses notion of *master* and *slave* servers.

... PostgreSQL and Parallelism

39/52

High availability ...

- updates occur on master, recorded in tx log
- tx logs shipped/streamed from master to slave(s)
- slave uses tx logs to maintain current state
- configuration controls frequency of log shipping
- bringing slave up-to-date is "fast" (~1-2secs)

Note: small window for data loss (committed tx log records not sent)

Load balancing ...

- not provided built-in to PostgreSQL, 3rd-party tools exist

Distributed Databases

Distributed Databases

41/52

A *distributed database* (DDB) is

- collection of multiple logically-related databases
- distributed over a network (generally a WAN)

A *distributed database management system* (DDBMS) is

- software that manages a distributed database
- providing access that hides complexity of distribution

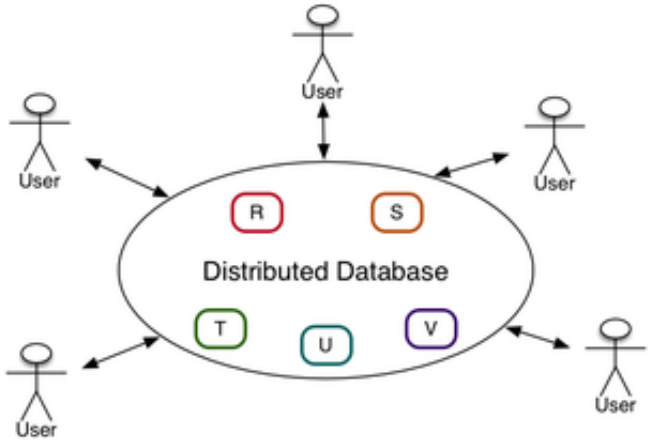
A DDBMS may involve

- instances of a single DBMS (e.g. ≥1 PostgreSQL servers)
- a layer over multiple different DBMSs (e.g. Oracle+PostgreSQL+DB2)

... Distributed Databases

42/52

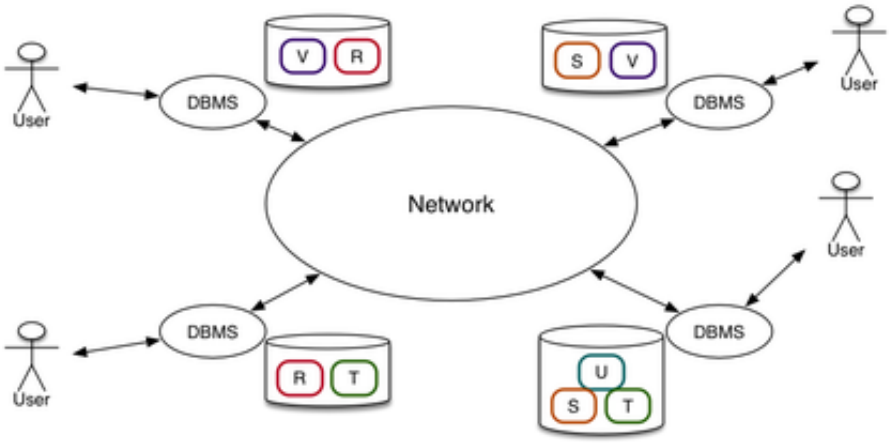
User View:



... Distributed Databases

43/52

Arhcitecture:



... Distributed Databases

44/52

Two kinds of distributed databases

- parallel database on a distributed architecture
  - single schema, homogeneous DBMSs
- independent databases on a distributed architecture
  - independent schemas, heterogeneous DBMSs

The latter are also called *federated* databases

Ultimately, the distributed database (DDB) provides

- global schema, with mappings from constituent schemas
- giving the impression of a single database

... Distributed Databases

45/52

Advantages of distributed databases

- allow information from multiple DBs to be merged
- provide for replication of some data (resilience)
- allow for possible parallel query evaluation

Disadavtanges of distributed databases

- cost of mapping between different schemas (federated)
- communication costs (write-to-network vs write-to-disk)
- maintaining ACID properties in distributed transactions

... Distributed Databases

46/52

Application examples:

- bank with multiple branches
  - local branch-related data (e.g. accounts) stored in branch
  - corporate data (e.g. HR) stored on central server(s)
  - central register of credit-worthiness for customers
- chain of department stores
  - store-related data (e.g. sales, inventory) stored in store
  - corporate data (e.g. HR, customers) stored on central server(s)
  - sales data sent to data warehouse for analysis

... Distributed Databases

47/52

In both examples

- some data is conceptually a single table at corporate level
- but does not physically exist as a table in one location

E.g. `account(acct_id, branch, customer, balance)`

- each branch maintains its own data (for its accounts)
- set of tuples, all with same `branch`
- bank also needs a view of *all* accounts

Data Distribution

48/52

Partitioning/distributing data

- where to place (parts of) tables
  - determined by usage of data (locality, used together)
  - affects communication cost ⇒ query evaluation cost
- how to partition data within tables
  - no partitioning ... whole table stored on ≥1 DBMS
  - *horizontal partitioning* ... subsets of rows
  - *vertical partitioning* ... subsets of columns

Problem: maintaining consistency

... Data Distribution

49/52

Consider table R decomposed into $R_1, R_2, ..., R_n$

Fragmentation can be done in multiple ways, but need to ensure ...

Completeness

- decompostion is complete iff each $t \in R$ is in some R_i

Reconstruction

- original R can be produced by some relational operation

Disjoint

- if item $t \in R_i$, then $t \notin R_k$, $k \neq i$ (assuming no replication)

Query Processing

50/52

Query processing typically involves *shipping* data

- e.g. reconstructing table from distributed partitions
- e.g. join on tables stored on separate sites

Aim: minimise shipping cost (since it is a networking cost)

Shipping cost becomes the "disk access cost" of DQOpt

Larger space of possibilities than standard QOpt, but same principle

- consider possible execution plans, choose cheapest

... Query Processing

51/52

Distributed query processing

- may require query ops to be executed on different nodes
  - node provides only source of some data
  - some nodes may have limited set of operations
- needs to merge data received from different nodes
  - may require data transformation (to fit schemas together)

Query optimisation in such contexts is *complex*.

Transaction Processing

Distribution of data complicates tx processing ...

- potential for multiple copies of data to become inconsistent
- commit or abort must occur consistently on all nodes

Distributed tx processing handled by *two-phase commit*

- initiating site has transaction coordinator C_i ...
  - waits for all other sites executing tx `T` to "complete"
  - sends `<prepare T>` message to all other sites
  - waits for `<ready T>` response from all other sites
  - if not received (timeout), or `<abort T>` received, flag abort
  - if all other sites respond `<ready T>`, flag commit
  - write `<commit T>` or `<abort T>` to log
  - send `<commit T>` or `<abort T>` to all other sites
- non-initiating sites write log entries before responding

Produced: 4 Oct 2018