Introduction

SL01

- Syllabus and Course Project
- Data Independence and Distributed Computing
- Definition of Distributed Databases
- Promises of Distributed Databases
- Technical Problems to be Studied
- Architectural Models
Syllabus

1. Introduction
2. Distributed Database Design
3. Semantic Data Control
4. Distributed Query Processing
5. Distributed Query Optimization
6. Distributed Transaction Management
Location: BIN 2.A.01
Time: Monday 14:00 - 15:45
Course page http://www.ifi.uzh.ch/dbtg
I assume you have followed an introductory database course before (relational model; algebra; SQL; query processing; etc).
Two Generals’ Problem

Task: Generals A and B must negotiate a time to attack.

Rules:
- A and B must attack at the same time
- A and B synchronize through messengers
- Messengers can get lost

source: Zoltan Fern, Stanford University, cs347 notes
Two Generals’ Problem/2

- How many messages are needed, so that both know for sure when to attack?

source: Zoltan Fern, Stanford University, cs347 notes
We must relax rules.

Probalistic approach with retransmissions.

What is the probability that B attacks at time t?

- B attacks at t if he receives message to do so no later than t.
- Each message delivered with probability p.
- One message per time unit.
Two Generals’ Problem/4

- \( a(1) = p \)
- \( a(2) = p + (1 - p)p \)
- \( a(3) = p + (1 - p)p + (1 - p)^2p \)
- \( a(4) = p + (1 - p)p + (1 - p)^2p + (1 - p)^3p \)
Two seminal contributions:


Google File System:

- Master manages metadata
- Files broken into chunks (typically 64MB)
- Chunks are replicated across three machinery for fault-tolerance
- Data transfer happens directly between clients and chunk servers.
Data Processing Systems/3

**MapReduce**: hides details of distributed programming

- Computations are expressed with two functions: Map and Reduce.
- The Map function produces a set of intermediate key/value pairs.
- The MapReduce framework groups all key/value pairs with the same key and passes them to the Reduce function.
- The Reduce function receives an intermediate key with its set of associated values and merges them.

source: Sherif Sakr, IfI summer school 2016
Hadoop

- is an open-source solution for data-intensive distributed applications
- clones the Google’s MapReduce framework.
- processes very large amount of unstructured and complex data.
- runs on a large number of machines that don’t share memory/disks.

source: Sherif Sakr, IfI summer school 2016
- **Hive**: SQL over Apache Hadoop
- **Impala**: open source, native analytic database for Apache Hadoop
- **Tajo**: distributed data warehouse system for Apache Hadoop
- **Giraph**: iterative graph processing system built for high scalability
- **Hama**: analytics using the Bulk Synchronous Parallel (BSP) model
- **Mahout**: an environment for scalable machine learning applications
MapReduce is a giant step backward in the programming paradigm for large-scale data intensive applications because

- Schemas are good.
- Separation of the schema from the application is good.
- High-level access languages are good.

MapReduce is a sub-optimal implementation, in that it uses brute force instead of indexing.

- MapReduce has no indexes and therefore has only brute force as a processing option.
- MapReduce systems deal poorly with skew.
-Reducers that run multiple reduce tasks simultaneously induce many disk seeks.
Distributed DBS versus MapReduce/2
(MapReduce: A major step backwards [David DeWitt, 2008])

- MapReduce misses most of the features that are routinely included in current DBMS.
  - **Bulk loader** to transform input data in files into a desired format and load it into a DBMS
  - **Indexing** as noted above
  - **Updates** to change the data in the database
  - **Transactions** to support parallel update and recovery from failures during update
  - **Integrity constraints** to help keep garbage out of the database
  - **Referential integrity** to help keep garbage out of the database
  - **Views** so the schema can change without having to rewrite the application program
Thus, MapReduce is not the ultimate cure to all your data management problems.

Today's data processing systems are domain-specific, optimized and vertically focused systems.

source: Sherif Sakr, IfI summer school 2016
Apache **Spark** is a fast, general engine for large scale data processing on a computing cluster (new engine for Hadoop)

- Developed initially at UC Berkeley in Scala.
- RDD (Resilient Distributed Dataset), an in-memory data abstraction, is the fundamental unit of data in Spark
- Resilient: if data in memory is lost, it can be recreated from the lineage (how it was computed)
- Distributed: stored in memory across the cluster
- Dataset: data can come from a file or be created programmatically
- Spark programming consists of performing operations (e.g., Map, Filter) on RDDs
Data Processing Systems/8

- Apache **Flink** is a distributed in-memory data processing framework which represents an alternative to the MapReduce framework that supports both of batch and realtime processing.
- Flink has originated from the Stratosphere research project that was started at the Technical University of Berlin.
- Instead of the map and reduce abstractions, Flink uses a directed graph approach that leverages in-memory storage for improving the performance of the runtime execution.
- True streaming capabilities: Execute everything as streams
- Native iterative execution
- DataSet API, DataStream API, Table API
**Pregel** from Google is the first BSP (Bulk Synchronous Parallel) based implementation for graph processing

Communication through message passing (usually sent along outgoing edges from each vertex) + Shared-Nothing graph programming.

**Vertex-centric computation, each vertex:**
- Receives messages sent in the previous superstep
- Executes the same user-defined function
- Modifies its value
- If active, sends messages to other vertices (next superstep)
- Votes to halt if it has no further work to do
source: Sherif Sakr, IfI summer school 2016
Database Processing Systems/11

- **Column Stores**: MonetDB, Greenplum, C-Store (Vertica)
- **Non-relational Databases**: BigTable, Cassandra, HBase, Spanner
- **Document/XML Stores**: MongoDB, CouchDB, BaseX,
- **Key-value Stores**: Berkeley DB, NoSQL, Tokyo Cabinet
- **Graph Databases**: Neo4J, InfiniteGraph, Virtuoso,
- **Object Databases**: GemStone, db4o, ZODB,
- **Cluster Computing**: Spark
- **Graph Processing**: Pregel, Giraph, GraphLab, GraphX
- **Event Processing**: Storm, Flink, S4
Database Courses

- Datenbankpraktikum, FS,  
  A. Geppert, Tuesday 16.15-18.00

- Distributed Database Systems (DDBS), HS,  
  M. Böhlen, Monday 14.00-15.45

- XML and Databases (TSDM), FS,  
  C. Türker, Thursday 8.00-9.45

- Data Warehousing (DW), FS,  
  A. Geppert, Tuesday 16.15-18.00

- Temporal and Spatial Data Management (TSDM), HS,  
  M. Böhlen, Monday 14.00-15.45

DDBS and DW are taught in even years. TSDM is taught in odd years.
I hear and forget
I learn and remember
I do and understand

- Goal: in-depth understanding of a selected part of the course
- Task: Implement a solution from a research paper
- Outcome: Running implementation and 5-10 page report
Choose one of the following papers:


Course project shall be solved in group of 2 people.

4th week, October 10:
  - Presentation of problem in terms of a numeric example
  - Plan for implementation (components, SW, functionality)
  - Plan for evaluation

9th week, November 14:
  - Presentation of solution (problem definition and solution)
  - Strong and weak points of implementation
  - Demonstration of implementation

14th week, December 19:
  - Hand in of project report and implementation (source code, data)
  - Send pdf by email to wellenzohn@ifi.uzh.ch, boehlen@ifi.uzh.ch
## Tentative Schedule

<table>
<thead>
<tr>
<th>Date</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>19.09</td>
<td>Introduction to Distributed Database Systems</td>
</tr>
<tr>
<td>26.09</td>
<td>Distributed Database Design</td>
</tr>
<tr>
<td>03.10</td>
<td>Distributed Database Design</td>
</tr>
<tr>
<td>10.10</td>
<td>Course project: plan for implementation and example</td>
</tr>
<tr>
<td>17.10</td>
<td>Distributed Database Design</td>
</tr>
<tr>
<td>24.10</td>
<td>Semantic Data Control</td>
</tr>
<tr>
<td>31.10</td>
<td>Query Processing</td>
</tr>
<tr>
<td>07.11</td>
<td>Distributed Query Optimization</td>
</tr>
<tr>
<td>14.11</td>
<td>Course project: demonstration of implementation</td>
</tr>
<tr>
<td>21.11</td>
<td>Distributed Query Optimization</td>
</tr>
<tr>
<td>28.11</td>
<td>Distributed Transactions and Concurrency</td>
</tr>
<tr>
<td>05.12</td>
<td>Distributed Transactions and Concurrency</td>
</tr>
<tr>
<td>12.12</td>
<td>Distributed Transactions and Concurrency</td>
</tr>
<tr>
<td>19.12</td>
<td>Conclusion</td>
</tr>
</tbody>
</table>
Exam

- There is an oral exam at the end of the course
- The exam takes place January 16, 2011 starting at 12:00
- The oral exam lasts half an hour in total
- You have 10 minutes for your project (demonstration and explanation)
- There will be a question about a selected topic from the course.
- The question will be drawn randomly.
- It is important that you illustrate the answer to your question through an appropriately chosen example.
In the old days, programs stored data in regular files.

Each program has to maintain its own data:

- huge overhead
- error-prone
The development of DBMSs helped to fully achieve data independence. Provide centralized and controlled data maintenance and access. Application is immune to physical and logical file organization.
Distributed database system is the union of what appear to be two diametrically opposed approaches to data processing: database systems and computer network.

- Computer networks promote a mode of work that goes against centralization.

Key issues to understand this combination:

- The most important objective of DB technology is integration not centralization.
- Integration is possible without centralization, i.e., integration of databases and networking does not mean centralization.

Goal of distributed database systems: achieve data integration and data distribution transparency.
Distributed Computing

▶ A **distributed computing system** is a collection of autonomous processing elements that are interconnected by a computer network.
▶ The elements cooperate in order to perform the assigned task.
▶ The term “distributed” is used very broadly. The exact meaning of the word depends on the context. What can be distributed?
   ▶ Processing logic
   ▶ Functions
   ▶ Data
   ▶ Control
▶ Classification of distributed systems with respect to various criteria
   ▶ Degree of coupling, i.e., how closely the processing elements are connected; e.g., measured as ratio of amount of data exchanged to amount of local processing; weak coupling, strong coupling
   ▶ Interconnection structure; e.g., point-to-point connection between processing elements, common interconnection channel
   ▶ Synchronization; synchronous, asynchronous
Definition of DDB and DDBMS/1

- **A distributed database (DDB)** is a collection of multiple, logically interrelated databases distributed over a computer network.

- **A distributed database management system (DDBMS)** is the software that manages the DDB and provides an access mechanism that makes this distribution transparent to the users.

- **DDBS (distributed database system)** = DDB + DDBMS

- **Assumptions**
  - Data stored at a number of sites each site logically consists of a single processor.
  - Processors at different sites are interconnected by a computer network (we do not consider multiprocessors in DDBS, cf. parallel systems).
  - DDB is a database, not a collection of files. Data is logically related; structured into multiple files; accessed through common interface.
  - DDBMS is a collection of DBMSs.
Definition of DDB and DDBMS/2
Definition of DDB and DDBMS/3

- **Example:** Database consists of 3 relations employees, projects, and assignment which are partitioned and stored at different sites (fragmentation).

- We study the problems with queries, transactions, concurrency, and reliability.
What is not a DDBS?

- The following systems are different from (though related to) distributed DB systems
Promises of DDBSs

Distributed Database Systems provide the following advantages:

- Transparency of distributed and replicated data
- Higher reliability
- Improved performance
- Easier system expansion
Transparency

- Refers to the separation of the higher-level semantics of the system from the lower-level implementation issues.
- A transparent system “hides” implementation details from users.
- A fully transparent DBMS provides high-level support for the development of complex applications.

(a) User wants to see one database

(b) Programmer sees many databases
The goal of transparency is to provide a system where users do not have to worry about representation, fragmentation, location, and replication of data.

Various forms of transparency can be distinguished for DDBSs:

- Network transparency (also called distribution transparency)
  - Location transparency
  - Naming transparency
- Replication transparency
- Fragmentation transparency
  - Horizontal fragmentation
  - Vertical fragmentation
Network (or distribution) transparency allows a user to perceive a DDBS as a single, logical entity.

The user is protected from the operational details of the network (or even does not know about the existence of the network).

Two types of network transparency can be identified: location transparency and naming transparency.

The user does not need to know the location of data items and a command used to perform a task is independent from the location of the data and the site the task is performed (location transparency).

A unique name is provided for each object in the database (naming transparency).

In absence of this, users are required to embed the location name as part of an identifier.
Common (suboptimal) ways to get naming transparency at the application level:

- **Solution 1:** Create a central name server; however, this results in
  - loss of some local autonomy
  - central site may become a bottleneck
  - low availability (if the central site fails remaining sites cannot create new objects)

- **Solution 2:** Prefix object with identifier of site that created it
  - e.g., branch created at site S1 might be named S1.BRANCH
  - Also need to identify each fragment and its copies
  - e.g., copy 2 of fragment 3 of Branch created at site S1 might be referred to as S1.BRANCH.F3.C2
  - Difficult for user
  - Relocation of objects becomes difficult
Replication transparency ensures that the user is not involved in the management of copies of some data.

The user should even not be aware about the existence of replicas, rather should work as if there exists a single copy of the data.

Replication of data is needed for various reasons:
- e.g., increased efficiency for read-only data access.
Fragmentation transparency ensures that the user is not aware of and is not involved in the fragmentation of the data.

The user is not involved in finding query processing strategies over fragments or formulating queries over fragments.

The evaluation of a query that is specified over an entire relation but now has to be performed on top of the fragments requires an appropriate query evaluation strategy.

Fragmentation is commonly done for reasons of performance, availability, and reliability.

Two fragmentation alternatives:
- Horizontal fragmentation: divide a relation into subsets of tuples
- Vertical fragmentation: divide a relation by columns
Promises of DDBSs – Reliability

Higher reliability

- Replication of components and data should make DDBMS more reliable.
- No single points of failure
- e.g., a broken communication link or processing element does not bring down the entire system
- Distributed transaction processing guarantees the consistency of the database and concurrency
Promises of DDBSs – Performance

**Improved performance**

- Proximity of data to its points of use
  - Reduces remote access delays
  - Requires some support for fragmentation and replication
- Parallelism in execution
  - Inter-query parallelism
  - Intra-query parallelism
- Update and read-only queries influence the design of DDBSs substantially
  - If mostly read-only access is required, as much as possible of the data should be replicated
  - Writing becomes more complicated with replicated data
Promises of DDBSs – Extendability

Easier system expansion

- Issue is database scaling
- Emergence of microprocessor and workstation technologies
  - Network of workstations much cheaper than a single mainframe computer
- Data communication cost versus telecommunication cost
- Increasing database size
Distributed database design
- How to fragment the data?
- Partitioned data vs. replicated data?

Distributed query processing
- Design algorithms that analyze queries and convert them into a series of data manipulation operations
- Distribution of data, communication costs, etc. has to be considered
- Find optimal query plans
Technical Problems to be Studied/2

- **Distributed concurrency control**
  - Synchronization of concurrent accesses such that the integrity of the DB is maintained
  - Integrity of multiple copies of (parts of) the DB have to be considered (mutual consistency)

- **Reliability**
  - How to make the system resilient to failures
  - Atomicity and durability

- Many techniques and solutions for DDBMSs build on standard DBMS solutions and extend these to distributed settings.
Architecture

- **Architecture**: The architecture of a system defines its structure:
  - the components of the system are identified;
  - the function of each component is specified;
  - the interrelationships and interactions among the components are defined.

- Applies both for computer systems as well as for software systems, e.g,
  - division into modules, description of modules, etc.
  - architecture of a computer

- There is a close relationship between the architecture of a system, standardisation efforts, and a reference model.
ANSI/SPARC Architecture of DBMS

- ANSI/SPARC architecture is based on data
- 3 views of data: external view, conceptual view, internal view
- Defines a total of 43 interfaces
Example/1

- Conceptual schema: Provides enterprise view of entire database

```
RELATION EMP [  
    KEY = {ENO}  
    ATTRIBUTES = {  
        ENO : CHARACTER(9)  
        ENAME: CHARACTER(15)  
        TITLE: CHARACTER(10)  
    }  
]  

RELATION PAY [  
    KEY = {TITLE}  
    ATTRIBUTES = {  
        TITLE: CHARACTER(10)  
        SAL : NUMERIC(6)  
    }  
]  

RELATION PROJ [  
    KEY = {PNO}  
    ATTRIBUTES = {  
        PNO : CHARACTER(7)  
        PNAME : CHARACTER(20)  
        BUDGET: NUMERIC(7)  
        LOC : CHARACTER(15)  
    }  
]  

RELATION ASG [  
    KEY = {ENO,PNO}  
    ATTRIBUTES = {  
        ENO : CHARACTER(9)  
        PNO : CHARACTER(7)  
        RESP : CHARACTER(10)  
        DUR : NUMERIC(3)  
    }  
]  
```
Internal schema: Describes the storage details of the relations.

- Relation EMP is stored on an indexed file
- Index is defined on the key attribute ENO and is called EMINX
- A HEADER field is used that might contain flags (delete, update, etc.)

```
INTERNAL_REL  EMPL  [
  INDEX ON  E#  CALL  EMINX
  FIELD  =
    HEADER:  BYTE (1)
    E#:  BYTE (9)
    ENAME:  BYTE (15)
    TIT:  BYTE (10)
]
```

Conceptual schema:

```
RELATION  EMP  [
  KEY  =  {ENO}
  ATTRIBUTES  =
    ENO:  CHARACTER (9)
    ENAME:  CHARACTER (15)
    TITLE:  CHARACTER (10)
]
```
• External view: Specifies the view of different users/applications

  • Application 1: Calculates the payroll payments for engineers

    ```sql
    CREATE VIEW PAYROLL (ENO, ENAME, SAL) AS
    SELECT EMP.ENO, EMP.ENAME, PAY.SAL
    FROM EMP, PAY
    WHERE EMP.TITLE = PAY.TITLE
    ```

  • Application 2: Produces a report on the budget of each project

    ```sql
    CREATE VIEW BUDGET(PNAME, BUD) AS
    SELECT PNAME, BUDGET
    FROM PROJ
    ```
Architectural Models for DDBSs (or more generally for multiple DBMSs) can be classified along three dimensions:

- Autonomy
- Distribution
- Heterogeneity
Autonomy: Refers to the distribution of control (not of data) and indicates the degree to which individual DBMSs can operate independently.

- **Tight integration**: a single-image of the entire database is available to any user who wants to share the information (which may reside in multiple DBs); realized such that one data manager is in control of the processing of each user request.

- **Semiautonomous** systems: individual DBMSs can operate independently, but have decided to participate in a federation to make some of their local data sharable.

- **Total isolation**: the individual systems are stand-alone DBMSs, which know neither of the existence of other DBMSs nor how to communicate with them; there is no global control.
Autonomy has different dimensions

- **Design autonomy**: each individual DBMS is free to use the data models and transaction management techniques that it prefers.
- **Communication autonomy**: each individual DBMS is free to decide what information to provide to the other DBMSs.
- **Execution autonomy**: each individual DBMS can execute the transactions that are submitted to it in any way that it wants to.
Distribution: Refers to the physical distribution of data over multiple sites.

- *No distribution*: No distribution of data at all
- *Client/Server distribution*:
  - Data are concentrated on the server, while clients provide application environment/user interface
  - First attempt to distribution
- *Peer-to-peer distribution* (also called *full distribution*):
  - No distinction between client and server machine
  - Each machine has full DBMS functionality
Heterogeneity: Refers to heterogeneity of the components at various levels

- hardware
- communications
- operating system
- DB components (e.g., data model, query language, transaction management algorithms)
General idea is to **divide the functionality** of the database system into two classes:

- **server functions**: mainly data management, including query processing, optimization, transaction management, etc.
- **client functions**: might also include some data management functions (consistency checking, transaction management, etc.) not just user interface

Client and server refer to actual machines (i.e., **different machines** where the SW runs).

Leads to a more efficient division of the work.

Yields a two-tier architecture (a further distribution of the functionality leads to n-tier architectures with application servers).

Different types of client/server architecture

- Multiple client/single server
- Multiple client/multiple server
Client-Server Architecture/2

- One server, many clients
- Similar to centralized DBMSs
- Differences are execution of transactions and cache management.

Diagram:

- Operating System
  - User Interface
  - Application Program
  - Client DBMS
  - Communication Software
- SQL queries
- Result relation

Diagram layers:
- Communication Software
- Semantic Data Controller
- Query Optimizer
- Transaction Manager
- Recovery Manager
- Runtime Support Processor
- Database
Client-Server Architecture/3

- Many servers, many clients
- General application server; heavier clients
Client-Server Architecture/4

- Many servers, many clients
- Specialized application servers; lighter clients
Peer-to-Peer Architecture/1

- Data organizational view.
- Provides transparency since it is an extension of the ANSI/SPARC architecture.
- **Local internal schema (LIS)**
  - Describes the local physical data organization (which often is different on each machine)
- **Local conceptual schema (LCS)**
  - Describes logical data organization at each site
  - Required since the data are fragmented and replicated
- **Global conceptual schema (GCS)**
  - Describes the global logical view of the data
  - Union of the LCSs
- **External schema (ES)**
  - Describes the user/application view of the data
Components of a distributed peer-to-peer database systems.

User processor handles interactions with users/applications.

Data processor handles storage and data processing.

User and data processor are present on each machine (this is organizational separation; not functional separation as for client/server).
Individual DBMSs are fully autonomous and have no concept of cooperation.

“Data integration system” is an often used term for such systems.

Fundamental difference to peer-to-peer DBMS is in the definition of the global conceptual schema (GCS)

- In a MDBMS the GCS represents only the collection of some of the local databases that each local DBMS want to share.
- The GCS is a (proper) subset of the union of the LCSs (no complete view exists)

This leads to the question, whether the GCS should even exist in a MDBMS.

Two different architecture models:

- Models with a GCS
- Models without GCS
Multi-DBMS Architecture/2

- Model with a GCS
  - GCS is the union of parts of the LCSs
  - Local DBMS define their own views on the local DB
Multi-DBMS Architecture/3

- Model without a GCS
  - The local DBMSs present to the multi-database layer the part of their local DB they are willing to share.
  - External views are defined on top of LCSs
Multi-DBMS Architecture/4

- Full-fledged DBMSs, each managing a different database.
- SW layer on top that allows to access various databases.
- For individual DBMSs the SW layer is simply yet another application.
- Implemented through mediator/wrapper technology.
Cloud Computing

Cloud computing is
  ▶ on-demand access to shared computing resources (HW and SW)
  ▶ a business model (computing resources are rented)
  ▶ abstracts from the computing resources; hardware virtualization
  ▶ infrastructure/hardware/software as a customizable service

Cloud computing versus
  ▶ Client-server system: functional division of entities in a system
  ▶ Software-as-a-service (SaaS): rented non-customizable software
  ▶ Massively parallel computing: large number of processors/computers
  ▶ Grid computing: large number of distributed computers
  ▶ Peer-to-peer computing: no central server; file sharing
  ▶ Distributed computing: coordinated computation of $n > 1$ computers
A distributed database (DDB) is a collection of logically interrelated databases distributed over a computer network.

Data is stored at a number of sites, the sites are connected by a network.

DDBS supports the relational model. DDBS is not a remote file system.

Transparent system ‘hides’ implementation details from users:
- Distribution/network transparency
- Replication transparency
- Fragmentation transparency

Higher reliability; improved performance; easier system expansion.

Programming a distributed database involves:
- Distributed database design
- Distributed query processing
- Distributed concurrency control
Architecture defines the structure of the system. There are different ways to define the architecture: e.g., based on components or data.

DDBS might be based on identical components (homogeneous systems) or different components (heterogeneous systems).

ANSI/SPARC architecture defines external, conceptual, and internal schemas.

There are three orthogonal implementation dimensions for DDBS: level of distribution, autonomy, and heterogeneity.

Different architectures are discussed:

- Client-Server Systems
- Peer-to-Peer Systems
- Multi-DBMS