# Non-Standard Database Systems

## Heterogeneous Distributed Databases

Nikolaus Augsten
nikolaus.augsten@sbg.ac.at
Department of Computer Sciences
University of Salzburg

database research group
http://dbresearch.uni-salzburg.at

SS 2017/18

Version June 12, 2018



## Outline







## Outline







## Heterogeneous Distributed Databases

- Many database applications require data from a variety of preexisting databases located in a heterogeneous collection of hardware and software platforms
- Data models may differ (hierarchical, relational , etc.)
- Transaction commit protocols may be incompatible
- Concurrency control may be based on different techniques (locking, timestamping, etc.)
- System-level details almost certainly are totally incompatible.
- A multidatabase system is a software layer on top of existing database systems, which is designed to manipulate information in heterogeneous databases
  - Creates an illusion of logical database integration without any physical database integration

## Advantages

- Preservation of investment in existing
  - hardware
  - system software
  - applications
- Local autonomy and administrative control
- Allows use of special-purpose DBMSs
- Step towards a unified homogeneous DBMS
- Full integration into a homogeneous DBMS faces
  - Technical difficulties and cost of conversion
  - Organizational/political difficulties
    - Organizations do not want to give up control on their data
    - Local databases wish to retain a great deal of autonomy





## Unified View of Data

- Agreement on a common data model
  - Typically the relational model
- Agreement on a common conceptual schema
  - Different names for same relation/attribute
  - Same relation/attribute name means different things
- Agreement on a single representation of shared data
  - E.g. data types, precision,
  - Character sets
    - ASCII vs EBCDIC
    - Sort order variations
- Agreement on units of measure
- Variations in names
  - E.g. Köln vs Cologne, Mumbai vs Bombay





## Query Processing

- Several issues in query processing in a heterogeneous database
- Schema translation
  - Write a wrapper for each data source to translate data to a global schema
  - Wrappers must also translate updates on global schema to updates on local schema
- Limited query capabilities
  - Some data sources allow only restricted forms of selections
    - E.g. web forms, flat file data sources
  - Queries have to be broken up and processed partly at the source and partly at a different site
- Removal of duplicate information when sites have overlapping information
  - Decide which sites to execute query
- Global query optimization





## Mediator Systems

- Mediator systems are systems that integrate multiple heterogeneous data sources by providing an integrated global view, and providing query facilities on global view
  - Unlike full fledged multidatabase systems, mediators generally do not bother about transaction processing
  - But the terms mediator and multidatabase are sometimes used interchangeably
  - The term virtual database is also used to refer to mediator/multidatabase systems

## Transaction Management in Multidatabases

- Local transactions are executed by each local DBMS, outside of the MDBS system control.
- Global transactions are executed under multidatabase control.
- Local autonomy — local DBMSs cannot communicate directly to synchronize global transaction execution and the multidatabase has no control over local transaction execution.
  - local concurrency control scheme needed to ensure that DBMS's schedule is serializable
  - in case of locking, DBMS must be able to guard against local deadlocks.
  - need additional mechanisms to ensure global serializability





## Local vs. Global Serializability

- The guarantee of local serializability is not sufficient to ensure global serializability.
  - As an illustration, consider two global transactions $T_1$ and $T_2$, each of which accesses and updates two data items, $A$ and $B$, located at sites $S_1$ and $S_2$ respectively.
  - It is possible to have a situation where, at site $S_1$, $T_2$ follows $T_1$, whereas, at $S_2$, $T_1$ follows $T_2$, resulting in a nonserializable global schedule.
- If the local systems permit control of locking behavior and all systems follow two-phase locking
  - the multidatabase system can ensure that global transactions lock in a two-phase manner
  - the lock points of conflicting transactions would then define their global serialization order.





## Outline







## Data Storage on the Cloud

- Need to store and retrieve massive amounts of data
- Traditional parallel databases not designed to scale to 1000's of nodes (and expensive)
- Initial needs did not include full database functionality
  - Store and retrieve data items by key value is minimum functionality
    - Key-value stores
- Several implementations
  - Bigtable from Google,
  - HBase, an open source clone of Bigtable
  - Dynamo, which is a key-value storage system from Amazon
  - Cassandra, from Facebook
  - Sherpa/PNUTS from Yahoo!

# Key Value Stores

- Key-value stores support
  - *put*(*key*, *value*): used to store values with an associated key,
  - *get*(*key*): which retrieves the stored value associated with the specified key.
- Some systems such as Bigtable additionally provide range queries on key values
- Multiple versions of data may be stored, by adding a timestamp to the key





# Data Representation/1

- Records in many big data applications need to have a flexible schema
  - Not all records have same structure
  - Some attributes may have complex substructure
- XML and JSON data representation formats widely used





# Data Representation/2

- An example of a JSON object is:

```
{
  "ID": "22222",
  "name": {
    "firstname: "Albert",
    "lastname: "Einstein"
  },
  "deptname": "Physics",
  "children": [
    { "firstname": "Hans", "lastname": "Einstein" },
    { "firstname": "Eduard", "lastname": "Einstein" }
  ]
}
```





# Partitioning and Retrieving Data/1

- Key-value stores partition data into relatively small units (hundreds of megabytes).
- These partitions are often called tablets (a tablet is a fragment of a table)
- Partitioning of data into tablets is dynamic:
  - as data are inserted, if a tablet grows too big, it is broken into smaller parts
  - if the load (get/put operations) on a tablet is excessive, the tablet may be broken into smaller tablets, which can be distributed across two or more sites to share the load.
  - the number of tablets is much larger than the number of sites
    - similar to virtual partitioning in parallel databases

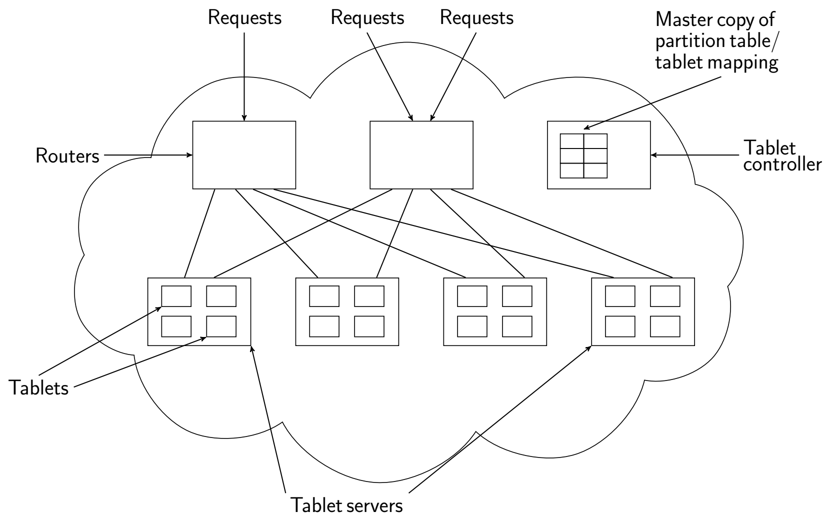
## Partitioning and Retrieving Data/2

- Each get/put request must be routed to the correct site
- Tablet controller tracks the partitioning function and tablet-to-site mapping
  - map a *get*() request to one or more tablets,
  - Tablet mapping function to track which site responsible for which tablet





## Partitioning and Retrieving Data/2








## Outline







## Directory Systems

- Typical kinds of directory information
  - Employee information such as name, id, email, phone, office addr, ...
  - Even personal information to be accessed from multiple places
    - e.g. Web browser bookmarks
- White pages
  - Entries organized by name or identifier
    - Meant for forward lookup to find more about an entry
- Yellow pages
  - Entries organized by properties
  - For reverse lookup to find entries matching specific requirements
- When directories are to be accessed across an organization
  - Alternative 1: Web interface. Not great for programs
  - Alternative 2: Specialized directory access protocols
    - Coupled with specialized user interfaces

## Directory Access Protocols

- Most commonly used directory access protocol:
  - LDAP (Lightweight Directory Access Protocol)
  - Simplified from earlier X.500 protocol
- Question: Why not use database protocols like ODBC/JDBC?
- Answer:
  - Simplified protocols for a limited type of data access, evolved parallel to ODBC/JDBC
  - Provide a nice hierarchical naming mechanism similar to file system directories
    - Data can be partitioned amongst multiple servers for different parts of the hierarchy, yet give a single view to user
    - E.g. different servers for Bell Labs Murray Hill and Bell Labs Bangalore
  - Provide a nice hierarchical naming mechanism similar to file system directories





## LDAP: Lightweight Directory Access Protocol

- LDAP Data Model
- Data Manipulation
- Distributed Directory Trees





## LDAP Data Model/1

- LDAP directories store entries
  - Entries are similar to objects
- Each entry must have unique distinguished name (DN)
- DN made up of a sequence of relative distinguished names (RDNs)
- E.g. of a DN
  - cn=Silberschatz, ou=Bell Labs, o=Lucent, c=USA
  - Standard RDNs (can be specified as part of schema)
    - cn: common name   ou: organizational unit
    - o: organization   c: country
  - Similar to paths in a file system but written in reverse direction





## LDAP Data Model/2

- Entries can have attributes
  - Attributes are multi-valued by default
  - LDAP has several built-in types
    - Binary, string, time types
    - Tel: telephone number   PostalAddress: postal address
- LDAP allows definition of object classes
  - Object classes specify attribute names and types
  - Can use inheritance to define object classes
  - Entry can be specified to be of one or more object classes
    - No need to have single most-specific type

## LDAP Data Model/3

- Entries organized into a directory information tree according to their DNs
  - Leaf level usually represent specific objects
  - Internal node entries represent objects such as organizational units, organizations or countries
  - Children of a node inherit the DN of the parent, and add on RDNs
    - E.g. internal node with DN c=USA
    - Children nodes have DN starting with c=USA and further RDNs such as o or ou
    - DN of an entry can be generated by traversing path from root
  - Children of a node inherit the DN of the parent, and add on RDNs
    - Entries can thus have more than one DN
    - E.g. person in more than one organizational unit





## LDAP Data Manipulation

- Unlike SQL, LDAP does not define DDL or DML
- Instead, it defines a network protocol for DDL and DML
  - Users use an API or vendor specific front ends
  - LDAP also defines a file format
    - LDAP Data Interchange Format (LDIF)
- Querying mechanism is very simple: only selection & projection





## LDAP Queries

- LDAP query must specify
  - Base: a node in the DIT from where search is to start
  - A search condition
    - Boolean combination of conditions on attributes of entries
    - Equality, wild-cards and approximate equality supported
  - A scope
    - Just the base, the base and its children, or the entire subtree from the base
  - Attributes to be returned
  - Limits on number of results and on resource consumption
  - May also specify whether to automatically dereference aliases
- LDAP URLs are one way of specifying query
- LDAP API is another alternative





## LDAP URLs

- First part of URL specifis server and DN of base
  - ldap:://aura.research.bell-labs.com/o=Lucent,c=USA
- Optional further parts separated by ? symbol
  - ldap:://aura.research.bell-labs.com/o=Lucent,c=USA??sub?cn=Korth
  - Optional parts specify
    1. attributes to return (empty means all)
    2. Scope (sub indicates entire subtree)
    3. Search condition (cn=Korth)

# C Code using LDAP API/1

```
#include <stdio.h>
#include <ldap.h>

main() {
  LDAP *ld;
  LDAPMessage *res, *entry;
  char *dn, *attr, *attrList[] = {"telephoneNumber", NULL};
  BerElement *ptr;
  int vals, i;

  // Open a connection to server
  ld = ldap_open("aura.research.bell-labs.com", LDAP_PORT);

  ldap_simple_bind(ld, "avi", "avi-passwd");

  ... actual query (next slide) ...

  ldap_unbind(ld);
}
```





# C Code using LDAP API/2

```
ldap_search_s(ld, "o=Lucent, c=USA", LDAP_SCOPE_SUBTREE,
              "cn=Korth", attrList, /* attrsonly*/ 0, &res);
/*attrsonly = 1 => return only schema not actual results*/
printf("found%d entries", ldap_count_entries(ld, res));
for (entry=ldap_first_entry(ld, res); entry != NULL;
     entry=ldap_next_entry(id, entry)) {
  dn = ldap_get_dn(ld, entry);
  printf("dn: %s", dn); /* dn: DN of matching entry */
  ldap_memfree(dn);
  // for each attribute
  for(attr = ldap_first_attribute(ld, entry, &ptr); attr != NULL;
      attr = ldap_next_attribute(ld, entry, ptr)) {
    printf("%s:", attr); // print name of attribute
    vals = ldap_get_values(ld, entry, attr);
    for (i = 0; vals[i] != NULL; i ++)
      printf("%s", vals[i]); // since attrs can be multivalued
      ldap_value_free(vals);
  }
}
ldap_msgfree(res);
```





# LDAP API

- LDAP API also has functions to create, update and delete entries
- Each function call behaves as a separate transaction
  - LDAP does not support atomicity of updates





# Distributed Directory Trees/1

- Organizational information may be split into multiple directory information trees
  - Suffix of a DIT gives RDN to be tagged onto to all entries to get an overall DN
    - E.g. two DITs, one with suffix o=Lucent, c=USA and another with suffix o=Lucent, c=India
  - Organizations often split up DITs based on geographical location or by organizational structure
  - Many LDAP implementations support replication (master-slave or multi-master replication) of DITs (not part of LDAP 3 standard)

## Distributed Directory Trees/2

- A node in a DIT may be a referral to a node in another DIT
  - E.g. Ou=Bell Labs may have a separate DIT, and DIT for o=Lucent may have a leaf with ou=Bell Labs containing a referral to the Bell Labs DIT
  - Referalls are the key to integrating a distributed collection of directories
  - When a server gets a query reaching a referral node, it may either
    - Forward query to referred DIT and return answer to client, or
    - Give referral back to client, which transparently sends query to referred DIT (without user intervention)