The Big Picture

Big Data Management

Anis Ur Rahman
Faculty of Computer Science Information Technology
University of Malaya

September 10, 2019

Course Logistics

- Course website:
- Registration for course & exams via:
- Organization:
- Lecture: Dr. Anis Ur Rahman
- Exam: XX.XX.2019 XX:00-XX:00 in XX (XX building)
- Lecture Tue, XX.00 - XX.00 h Room XX

Prerequisites

Background

- Algorithms
- Probability and stats
- Undergraduate-level databases
- Linear algebra
- Graph theory

Graduate-level programming skills i.e.

- ability to use unfamiliar software,
- picking up new languages,
- comfortable with at least one of Python/C/C++/Ruby/Java/MATLAB/R.

Course Grading

- Details coming soon (next lecture)
- Broadly
    - 2–3 quizzes
    - 2–3 assignments
    - Mid-term exam
    - Final exam
    - Semester project

Course Project

- 2, or 3 (max) persons per project.
- Major work for this class.
- Pick your own topic
  - You have to justify why the topic is interesting, and relevant to the course, and of suitable difficulty
- Harder way:
  - Joint projects with other courses are also negotiable.
  - In that case, you will need the approval of the instructor, and you also need to clarify exactly what steps will be done for this course, as well as for the other course.
- Ask me if you need help and ideas (I may release a list of suitable topics later maybe)

Course Project

- Proposal (5th week)
- Milestone (10th week)
- Final Report (15th week)
- Poster Presentation (or in-class presentation TBD)

What is Big Data?

Buzzword? Bubble? Gold rush? Revolution?

Big Data is high **volume**, high **velocity**, and/or high **variety** information assets that require **new forms of processing** to enable enhanced decision making, insight discovery and process optimization.

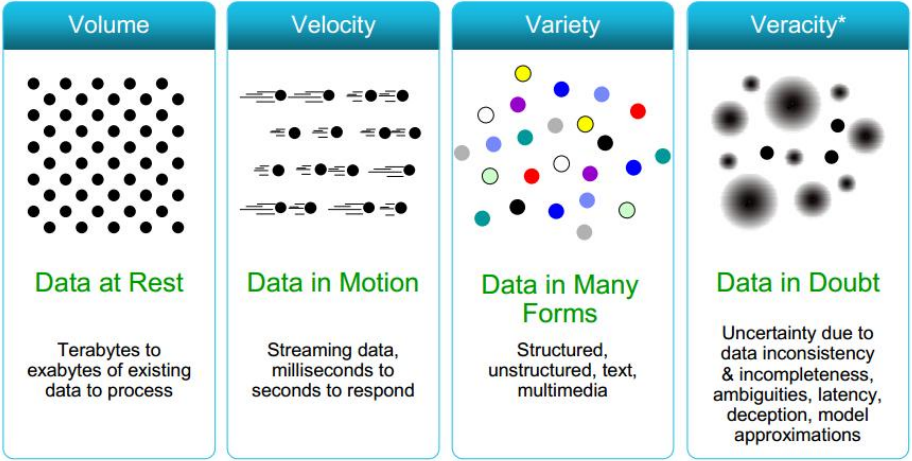
Where is Big Data?

Sources of Big Data

- Social media and networks
  - ...all of us are generating data
- Scientific instruments
  - collecting all sorts of data
- Mobile devices
  - tracking all objects all the time
- Sensor technology and networks
  - measuring all kinds of data

Big Data Characteristics

Basic 4V



Source: `https://www.datasciencecentral.com`

Big Data Characteristics

Basic 4V

- Volume (**Scale**)
  - Data volume is **increasing exponentially**, not linearly
  - Even large amounts of small data can result into Big Data
- Variety (**Complexity**)
  - Various formats, types, and structures (from semi-structured XML to unstructured multimedia)
- Velocity (**Speed**)
  - Data is being **generated** fast and needs to be **processed** fast
- Veracity (**Uncertainty**)
  - Uncertainty due to inconsistency, incompleteness, latency, ambiguities, or approximations

Big Data Characteristics

Additional V

- **Value**
  - Business value of the data (needs to be revealed)
- **Validity**
  - Data correctness and accuracy with respect to the intended use
- **Volatility**
  - Period of time the data is valid and should be maintained

Relational Databases

Implementation

- What is under-the-hood of a DB like Oracle/MySQL?

Design

- How do you **model** your **data** and **structure** your information in a database?

Programming

- How do you **use** the **capabilities** of a DBMS?

To achieves a balance between

- a firm **theoretical foundation** to designing moderate-sized databases
- **creating**, **querying**, and **implementing** realistic databases and connecting them to applications

Relational Databases

Data model

- Instance → database → table → row

Query languages

- **Real-world.** SQL (Structured Query Language)
- **Formal.** Relational algebra, relational calculi (domain, tuple)

Query patterns

- **Selection** based on complex conditions, projection, joins, aggregation, derivation of new values, recursive queries, ...

Representatives

- Oracle Database, Microsoft SQL Server, IBM DB2
- MySQL, PostgreSQL

Relational Databases: Normal Forms

Model

- Functional dependencies
- 1NF, 2NF, 3NF, BCNF (Boyce-Codd normal form)

Objective

- Normalization of database schema to BCNF or 3NF
- Algorithms. decomposition or synthesis

Motivation

- Diminish data redundancy, prevent update anomalies

However

Data is scattered into small pieces (high granularity), and so these pieces have to be joined back together when querying!

Relational Databases: Transactions

Model

- **Transaction** = flat sequence of database **operations** (READ, WRITE, COMMIT, ABORT)

Objectives

- **Enforcement** of **ACID** properties
- **Efficient** parallel / concurrent **execution** (slow hard drives, ...)

ACID properties

- **Atomicity.** **partial execution** is not allowed (all or nothing)
- **Consistency.** transactions turn **one valid** database **state** into another
- **Isolation.** **uncommitted** effects are **concealed** among transactions
- **Durability.** effects of **committed** transactions are **permanent**

What is the goal of a DBMS?

Electronic **record-keeping**

- **Fast** and **convenient** access to information

DBMS == database management system

- 'Relational' in this lecture
- **data** + **set of instructions** to access/manipulate data

What is a DBMS?

Features of a DBMS

- Support **massive** amounts of data
- **Persistent** storage
- **Efficient** and **convenient** access
- Secure, concurrent, and atomic access

Examples?

- **Traditionally.** search engines, banking systems, airline reservations, corporate records, payrolls, sales inventories.
- **New applications.** Wikis, social/biological/multimedia/scientific/geographic data, heterogeneous data.

Features of a DBMS

Support massive amounts of data

- Giga/tera/petabytes
- Far **too big** for main memory

Persistent storage

- Programs update, query, manipulate data.
- Data **continues** to live long after program **finishes.**

Efficient and convenient access

- **Efficient.** do not **search** entire database to answer a query.
- **Convenient.** allow users to **query** the data as easily as possible.

Secure, concurrent, and atomic access

- Allow **multiple users** to access database **simultaneously.**
- Allow a **user access** to only to **authorized** data.
- Provide some **guarantee** of reliability against system **failures.**

Example Scenario

Students, taking classes, obtaining grades

- Find my GPA

Obvious solution 1: Folders

Advantages?

- Cheap, easy-to-use

Disadvantages?

- No ad-hoc queries
- No sharing
- Large physical foot-print

Obvious Solution 2: Flat files

Flat files and C (C++, Java...) programs

- E.g. one (or more) UNIX/DOS files, with student records and their courses

Layout for student records?

- CSV ('comma-separated-values')

```
Hermione Grainger,123,Potions,A
Draco Malfoy,111,Potions,B
Harry Potter,234,Potions,A
Ron Weasley,345,Potions,C
```

Obvious Solution++

Layout for student records?

- Other possibilities like

```
123,Potions,A
111,Potions,B
234,Potions,A
345,Potions,C
```

Problems?

- inconvenient **access** to data
  - need 'C++' expertise, plus knowledge of file-layout
  - **data isolation**
- data **redundancy** (and inconsistencies)
- **integrity** problems
- **atomicity** problems
- **concurrent-access** problems
- **security** problems
- ...

Problems: Why?

Two main **reasons:**

- file-layout **description** is buried within the C programs, and
- no support for **transactions** (concurrency and recovery)

DBMSs handle exactly these two problems.

Example Scenario

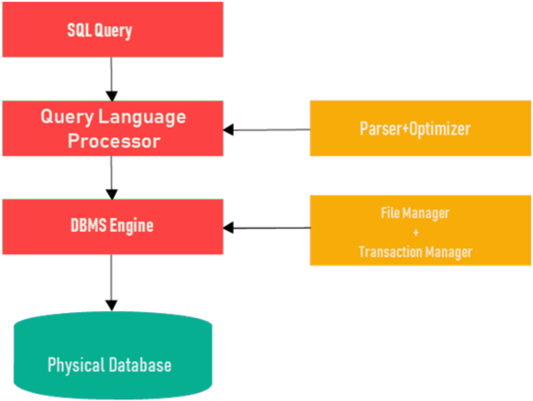
- **RDBMS** = "Relational" DBMS
- The relational model uses **relations** or **tables** to structure data
- ClassList relation:

Student	Course	Grade
Hermione Grainger	Potions	A
Draco Malfoy	Potions	B
Harry Potter	Potions	A
Ron Weasley	Potions	C

- **Relation** separates the logical view (externals) from the physical view (internals)
- Simple **query languages** (SQL) for accessing/modifying data
- E.g. find all students whose grades are better than B.

```
SELECT Student FROM ClassList WHERE Grade >"B"
```

DBMS Architecture



Transaction Processing

- One or more database operations are **grouped** into a **"transaction"**
- Transactions should meet the **"ACID test"**
  - **Atomicity.** All-or-nothing execution of transactions.
  - **Consistency.** Databases have consistency **rules** (e.g. what data is valid). A transaction should NOT violate the database's consistency. If it does, it needs to be rolled back.
  - **Isolation.** Each transaction must appear to be executed as if no other transaction is executing at the **same time**.
  - **Durability.** Any change a transaction makes to the database should **persist** and not be lost.

Disadvantages over (flat) files

Price

- additional expertise (SQL/DBA)

An **over-kill** for small, single-user data sets

- But: mobile phones (eg., android) use sqlite)

A Brief History of DBMS

- The earliest databases (1960s) evolved from file systems
- **File systems**
  - Allow storage of large amounts of data over a long period of time
  - File systems do not support:
    - **Efficient access** of data items at not known location on file
    - **Logical structure** of data is limited to creation of directory structures
    - **Concurrent access.** Multiple users modifying a single file generate non-uniform results
  - Navigational and hierarchical
  - User programmed the queries by walking from node to node in the DBMS.
- Relational DBMS (1970s to now)
  - View database in terms of **relations** or **tables**
  - High-level **query** and **definition** languages such as SQL
  - Allow user to specify what (s)he wants, not how to get what (s)he wants
- **Object-oriented DBMS (1980s).** Inspired by OO languages
  - Object-relational DBMS

The DBMS Industry

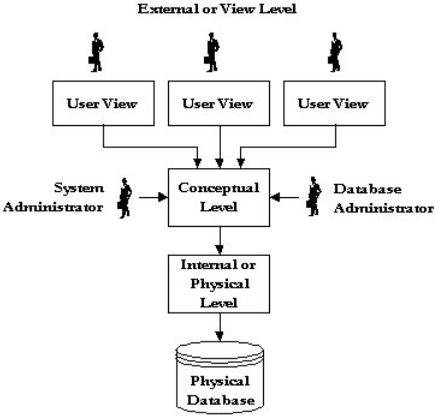
- A DBMS is a **software system.**
- **Major DBMS vendors.** Oracle, Microsoft, IBM, Sybase
- **Free/Open-source DBMS.** MySQL, PostgreSQL, Firebird
  - Used by companies such as Google, Yahoo, Lycos, BASF...
- All are "relational" (or "object-relational") DBMS.
- A multi-billion dollar industry.

Fundamental concepts

- Three-level architecture
- Logical data independence
- Physical data independence

Three-level architecture

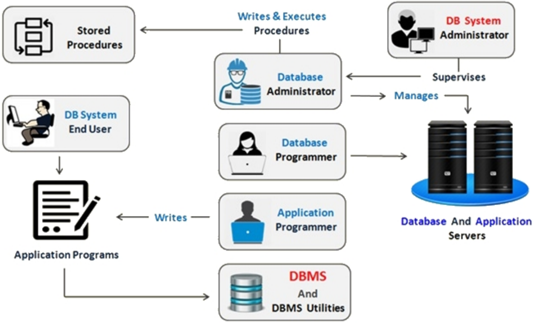
- **view level**
  - SELECT ssn FROM Student
  - SELECT ssn, cid FROM Takes
- **logical level**
  - e.g. tables
    - Student(ssn, name)
    - Takes(ssn, cid, grade)
  - can add (drop) column; add/drop table
- **physical level**
  - how are tables stored; how many bytes/attributes, etc
  - can add index; change record order



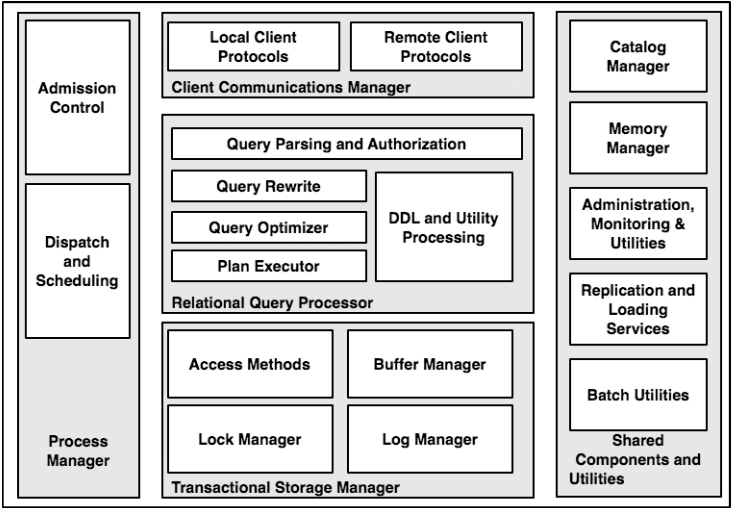
Hence, physical and logical data independence

Database users

- Naive users
- Casual users
- Application programmers
- DBA (Data base administrator)



Overall system architecture



Overall system architecture

- Users
- DBMS
  - **query processor**
    - DML compiler
    - embedded DML pre-compiler
    - DDL interpreter
    - Query evaluation engine
  - **storage manager**
    - authorization and integrity manager
    - transaction manager
    - buffer manager
    - file manager
  - **transaction manager**
- Files
  - data files
  - data dictionary = catalog = metadata
  - indices
  - statistical data

Some examples

- DBA doing a DDL (data definition language) operation, eg.,

```
create table student ...
```

- Casual user, asking for an update, eg.:

```
update student set name to 'smith' where ssn = '345'
```

- app. programmer, creating a report, eg

```
main(){
  ....
  exec sql "select * from student"
  ...
}
```

- 'naive' user, running the previous app.

Conclusions

- (relational) DBMSs: **electronic record keepers**
- **customize** them with create table commands
- **ask** SQL queries to retrieve info
- main advantages over (flat) files and scripts:
    - logical + physical data **independence**
        - i.e. **flexibility** of adding new attributes, new tables and indices
    - concurrency control and recovery

Current Trends

Big Data

- **Volume.** terabytes → zettabytes
- **Variety.** structured → structured and unstructured data
- **Velocity.** batch processing → streaming data
- ...

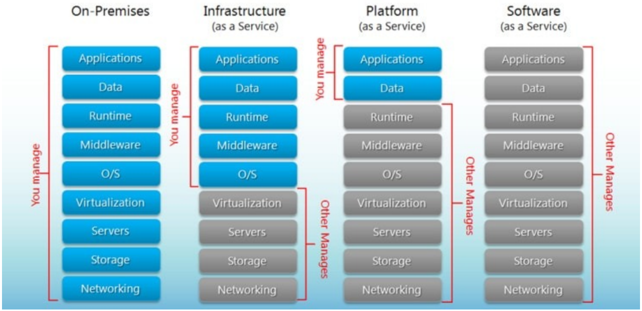
Big users

- Population **online**, hours spent online, devices online, ...
- Rapidly **growing** companies/web applications

Current Trends

Everything is **in cloud**

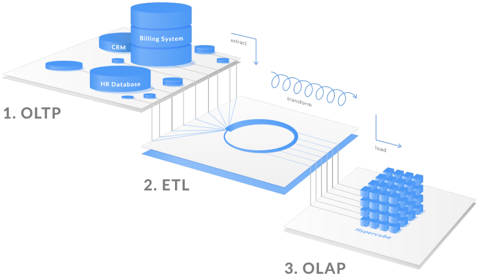
- **SaaS.** Software as a Service
- **PaaS.** Platform as a Service
- **IaaS.** Infrastructure as a Service



Current Trends

Processing paradigms

- **OLTP.** Online Transaction Processing
- **OLAP.** Online Analytical Processing
- ...
- **RTAP.** Real-Time Analytical Processing



Source: `https://www.imaginarycloud.com`

Current Trends

Data assumptions

- **Data format** is becoming **unknown** or **inconsistent**
- Linear growth → **unpredictable** exponential **growth**
- **Read requests** often **prevail** write requests
- **Data updates** are **no longer frequent**; data is expected to be **replaced**
- **Strong consistency** is no longer mission-critical

Current Trends

New approach is required

- Relational databases simply do not follow the current trends

Key technologies

- Distributed file systems
- MapReduce and other programming models
- Grid computing, cloud computing
- NoSQL databases
- Data warehouses
- Large scale machine learning

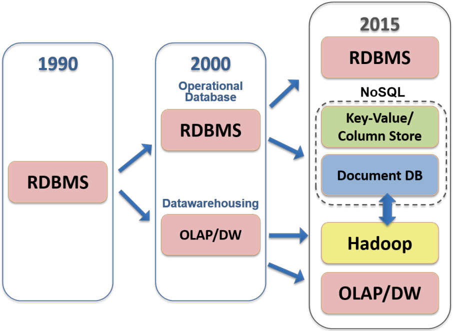
NoSQL Databases

A bit of history ...

- **1998.** First used for a relational database that **omitted usage** of SQL
- **2009.** First used during a conference to **advocate non-relational** databases

So?

- Not: no to SQL
- Not: not only SQL
- NoSQL is an accidental term with no precise definition



NoSQL Databases

What does NoSQL actually mean?

NoSQL movement

The whole point of seeking **alternatives** is that you need to **solve** a problem that relational databases are a bad fit for.

NoSQL databases

- Next generation databases mostly **addressing**,
  - being non-relational, distributed, open-source and horizontally scalable.
- original intention is modern **web-scale** databases.
- more **characteristics** apply as:
  - schema-free, easy replication support, simple API, eventually consistent, a huge data amount, and more.

Source: `http://nosql-database.org/`

Types of NoSQL Databases

Core types

- **Key-value** stores
- **Wide column** (column family, column oriented, ...) stores
- **Document** stores
- **Graph** databases

Non-core types

- **Object** databases
- Native **XML** databases
- **RDF** stores
- ...

Key-Value Stores

Data model

- The most **simple** NoSQL database type
- Works as a simple **hash table** (mapping)
- Key-value **pairs**
  - **Key.** (id, identifier, primary key)
  - **Value.** binary object, black box for the database system

Query patterns

- **Create**, **update** or **remove** value for a **given key**
- **Get** value for a given key

Characteristics

- Simple model → great **performance**, easily **scaled**, ...
- Simple model → **not** for **complex** queries nor complex data

Key-Value Stores

Suitable use cases

- Session data, user profiles, user preferences, shopping carts, ...
- I.e. when values are **only** accessed via **keys**

When **not** to use

- **Relationships** among entities
- Queries requiring access to the **content** of the value part
- Set operations involving **multiple** key-value pairs

Representatives

- Redis, MemcachedDB, Riak KV, Hazelcast, Ehcache, Amazon SimpleDB, Berkeley DB, Oracle NoSQL, Infinispan, LevelDB, Ignite, Project Voldemort
- Multi-model: OrientDB, ArangoDB

Document Stores

Data model

- **Documents**
  - Self-describing
  - Hierarchical tree structures (JSON, XML, …)
    - Scalar values, maps, lists, sets, nested documents, …
  - Identified by a unique identifier (key, …)
- Documents are **organized** into collections

Query patterns

- **Create**, **update** or **remove** a document
- **Retrieve** documents according to complex query conditions

Observation

Extended key-value stores where the value part is examinable!

Document Stores

Suitable **use cases**

- Event logging, content management systems, blogs, web analytics, e-commerce applications, ...
- I.e. for **structured documents** with **similar** schema

When **not** to use

- Set operations involving **multiple** documents
- Design of document structure is constantly **changing**
- I.e. when the required level of **granularity** outbalances the advantages of **aggregates**

Representatives

- MongoDB, Couchbase, Amazon DynamoDB, CouchDB, RethinkDB, RavenDB, Terrastore
- Multi-model: MarkLogic, OrientDB, OpenLink Virtuoso, ArangoDB

Wide Column Stores

Data model

- **Column family** (table)
  - Table is a **collection** of **similar** rows (not necessarily identical)
- **Row.** a collection of columns
  - Should encompass a **group** of data that is **accessed together**
  - Associated with a **unique row key**
- **Column.** consists of a column name and column value (and possibly other metadata records)
  - Scalar values, but also flat sets, lists or maps may be allowed

Query patterns

- **Create**, **update** or **remove** a row within a given column family
- Select rows according to a **row key** or **simple conditions**

Warning

Wide column stores are not just a **special kind** of RDBMSs with a **variable** set of columns!

Wide Column Stores

Suitable use cases

- Event logging, content management systems, blogs, ...
- I.e. for **structured flat data** with **similar** schema

When not to use

- **ACID transactions** are required
- **Complex queries.** aggregation (SUM, AVG, ...), joining, ...
- **Early prototypes.** i.e. when database design may change

Representatives

- Apache Cassandra, Apache HBase, Apache Accumulo, Hypertable, Google Bigtable

Graph Databases

Data model

- **Property graphs.** Directed/undirected graphs, i.e. collections of ...
  - **nodes** (vertices) for real-world entities, and
  - **relationships** (edges) between these nodes
  - Both the nodes and relationships can be **associated** with **additional properties**

Types of databases

- **Non-transactional** = **small** number of very **large** graphs
- **Transactional** = **large** number of **small** graphs

Graph Databases

Query patterns

- **Create**, **update** or **remove** a node / relationship in a graph
- Graph **algorithms** (shortest paths, spanning trees, ...)
- General graph **traversals**
- **Sub-graph** queries or **super-graph** queries
- **Similarity** based queries (approximate matching)

Representatives

- Neo4j, Titan, Apache Giraph, InfiniteGraph, FlockDB
- **Multi-model.** OrientDB, OpenLink Virtuoso, ArangoDB

Graph Databases

Suitable use cases

- Social networks, routing, dispatch, and location-based services, recommendation engines, chemical compounds, biological pathways, linguistic trees, ...
- I.e. simply for **graph structures**

When **not** to use

- Extensive **batch operations** are required
  - **Multiple** nodes/relationships are to be **affected**
- Only too **large graphs** to be stored
  - Graph **distribution** is **difficult** or impossible at all

Native XML Databases

Data model

- XML documents
  - **Tree** structure with **nested** **elements**, **attributes**, and **text values** (beside other less important constructs)
  - Documents are **organized** into **collections**

Query languages

- **XPath.** XML Path Language (**navigation**)
- **XQuery.** XML Query Language (**querying**)
- **XSLT.** XSL Transformations (**transformation**)

Representatives

- Sedna, Tamino, BaseX, eXist-db
- **Multi-model.** MarkLogic, OpenLink Virtuoso

RDF Stores

Data model

- **RDF triples**
  - **Components.** subject, predicate, and object
  - Each triple **represents a statement** about a real-world **entity**
- Triples can be viewed as **graphs**
  - **Vertices** for subjects and objects
  - **Edges** directly correspond to individual statements

Query language

- **SPARQL.** SPARQL Protocol and RDF Query Language

Representatives

- Apache Jena, rdf4j (Sesame), Algebraix
- **Multi-model.** MarkLogic, OpenLink Virtuoso

Features of NoSQL Databases

Data model

- **Traditional approach.** relational model
- **(New) possibilities:**
  - Key-value, document, wide column, graph
  - Object, XML, RDF, ...
- **Goal.** Respect the **real-world nature of data** (i.e. data structure and mutual relationships)

Features of NoSQL Databases

Aggregate structure

- **Aggregate definition**
  - Data unit with a complex structure
  - Collection of **related data pieces** treated as a unit (w.r.t data manipulation and data consistency)
- **Examples**
  - **Value part** of key-value pairs in key-value stores
  - **Document** in document stores
  - **Row of a column family** in wide column stores

Features of NoSQL Databases

Aggregate structure

- **Types** of systems
  - **Aggregate-ignorant.** relational, graph
    - It is not a bad thing, it is a feature
  - **Aggregate-oriented.** key-value, document, wide column

Design notes

- **No universal strategy** how to draw aggregate **boundaries**
- **Atomicity** of database operations: just a single aggregate at a time

Features of NoSQL Databases

Elastic scaling

- **Traditional approach.** scaling-up
  - Buying **bigger** servers as database load increases
- **New approach.** scaling-out
  - **Distributing** database data across **multiple** hosts
    - Graph databases (unfortunately): **difficult** or impossible at all

Data distribution

- **Sharding.** Particular ways how database data is **split into** separate groups
- **Replication.** Maintaining several **data copies** (performance, recovery)

Features of NoSQL Databases

Automated processes

- **Traditional approach.** Expensive and highly trained **database administrators**
- **New approach.** automatic recovery, distribution, tuning, ...

Relaxed consistency

- **Traditional approach.** Strong consistency (ACID properties and transactions)
- **New approach.** Eventual consistency only (BASE properties)
    - I.e. we have to make **trade-offs because of** the data distribution

Features of NoSQL Databases

Schemalessness

- **Relational databases.** Database schema **present** and strictly **enforced**
- **NoSQL databases.** **Relaxed** schema or completely **missing**

Consequences: Higher flexibility

- Dealing with **non-uniform data**
- Structural **changes** cause **no overhead**
- However, there is (usually) an **implicit schema**
  - need to know the data structure at the application level anyway

Features of NoSQL Databases

Open source

- Often community and enterprise versions (with extended features or extent of support)

Simple APIs

- Often state-less application interfaces (HTTP)

Features of NoSQL Databases

Current state. Five advantages

- **Scaling.** Horizontal distribution of data among hosts
- **Volume.** High volumes of data that cannot be handled by RDBMS
- **Administrators.** No longer needed because of the **automated** maintenance
- **Economics.** Usage of cheap commodity servers, lower overall costs
- **Flexibility.** Relaxed or missing data schema, **easier design changes**

Features of NoSQL Databases

Current State. Five challenges

- **Maturity.** Often still in **pre-production** phase with key features **missing**
- **Support.** Mostly **open source**, limited sources of **credibility**
- **Administration.** Sometimes relatively **difficult** to **install** and **maintain**
- **Analytics. Missing** support for **business intelligence** and **ad-hoc querying**
- **Expertise.** Low number of NoSQL **experts** available in the market

Conclusion

The **end of** relational databases?

- Certainly **no**
  - They are still **suitable** for **most projects**
  - Familiarity, stability, feature set, available support, ...
- However, we should also **consider** different database models and systems
  - **Polyglot persistence** = usage of **different** data stores in different circumstances

Course Overview: Outline and Objectives

Principles.

- Scaling, distribution, consistency
- Transactions, visualization, ...

Technologies.

- MapReduce programming model
  - Apache Hadoop
- **Data formats**
  - XML, JSON, RDF, ...
- **NoSQL databases**
  - Core: RiakKV, Redis, MongoDB, Cassandra, Neo4j
  - Non-core: XML, RDF
  - Data models, query languages, ...

Lecture Conclusion

Big Data

- **4V characteristics.** volume, variety, velocity, veracity
- **NoSQL** databases
- (New) logical **models**
  - **Core.** key-value, wide column, document, graph
  - **Non-core.** XML, RDF, ...
- **(New) principles and features**
  - **Horizontal** scaling, data **sharding** and **replication**, **eventual** consistency, ...