Distributed Data Management
Summer Semester 2013
TU Kaiserslautern

Dr.-Ing. Sebastian Michel

smichel@mmci.uni-saarland.de
Oral Exams: Available Dates

- 7. August 2013 (Wednesday)
- 8. August (Thursday)
- 9. August (Friday)
- 12. August (Monday)
- 13. August (Tuesday)
- 14. August (Wednesday)

- Starting 9am each, 5-6 slots, 30min (20+10) each

Mark the dates in your calendar. Good idea is to be available on 3 of these days.

Assignment of slots will be made later, after “official” registration.
Things Captured

• Roughly everything captured in the lecture.
• No need to learn detailed system issues of Hadoop or other systems
• But able to write MapReduce functions, e.g.
• We assume active participation in exercises for the exam, the entire sheet is relevant (again, you don’t have to code, but understand)
• Example questions or points are discussed in the lecture.
Lecture 5

NOSQL
The NoSQL “Movement”

• No one-size-fits-all database technology
• Not only SQL (not necessarily “no” SQL at all)
• Stands for group of non-traditional DBMS (not relational, often no SQL), for different purposes
  – key/value stores
  – graph databases
  – document stores
  – (also MapReduce)

Overview of systems: nosql-database.org/
NoSQL-Systems Characteristics

• Non-relational data model
• Systems are designed for distributed (scale-out) architecture. “Sharding”: one logical DB => multiple machines
• No or little schema
• Simple API (mainly no full SQL support); CRUD
• Mainly not full ACID semantics. Instead BASE.
• Often such systems are open-source projects.
Literature: Books


• In German: NoSQL: Einstieg in die Welt nichtrelationaler Web 2.0 Datenbanken von Stefan Edlich, et al., 2nd edition.

NoSQL: Key/Value Stores

• Simple storage for key->value pairs
• Value can be complex data type
• Example systems: Amazon Dynamo, Redis, Voldemort
• Simple “CRUD” operations: Create, Read, Update, Delete
• Some allow more complex queries, e.g., over ranges
Example Key/Value Store: Redis

- [http://try.redis.io/](http://try.redis.io/)  \(\leq\) check this out!

```
SET name “ddm13”
GET name #ddm13

LPUSH list “a”
LPUSH list “b”
LLENGTH #2
LRANGE list 0 1 # “b”, “a”
```
NoSQL: Document Stores

- Store JSON (Javascript Object Notation) or XML documents

- Examples: MongoDB, CouchDB

```json
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": 10021
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "fax",
      "number": "646 555-4567"
    }
  ]
}
```
Example: MongoDB

- [http://try.mongodb.org/](http://try.mongodb.org/)

```javascript
var student = {'Jim', scores: [75, 99, 87.2]};
db.lecture.store(student);

db.lecture.find();  // returns all entries

db.lecture.find({name: "Jim"});  // specific search

db.users.update({name: 'Johnny'}, {name: 'Cash',
languages: ['english']});

and you can use mapreduce!
```
NoSQL: Graph Databases

• Data model: Graph (Vertices, Edges)
• Queries: Shortest path, connected components, “followers”, …
• Examples: Neo4j, InfiniteGraph, GraphBase
• Specifically address graph queries. Aim: higher performance than “self joins” in RDBMS to traverse graph
Example: Neo4j

- [http://www.neo4j.org/learn/try](http://www.neo4j.org/learn/try)

**Graph Setup:**

```plaintext
start root=node(0) create (Neo {name:'Neo'}), (Morpheus {name: 'Morpheus'}), (Trinity {name: 'Trinity'}), (Cypher {name: 'Cypher'}), (Smith {name: 'Agent Smith'}), (Architect {name:'The Architect'}),
root-[:ROOT]->Neo, Neo-[:KNOWS]->Morpheus, Neo-[:LOVES]->Trinity, Morpheus-[:KNOWS]->Trinity, Morpheus-[:KNOWS]->Cypher, Cypher-[:KNOWS]->Smith, Smith-[:CODED_BY]->Architect
```

**Query:**

```plaintext
start n=node:node_auto_index(name='Neo')
match n-[r:KNOWS*]-m return n as Neo,r,m
```
NoSQL: Column Stores

- Logically “look” like common relational databases (tables)
- Physically organized in a per column fashion
- Good for analytical tasks over subsets of columns
- Dynamic schema, sparse data

Example systems: BigTable, HBase, Cassandra, Hypertable
HBase

• Scalable data store
• Made for sparse but wide tables: >millions of rows and/or columns
• Hadoop (Apache) project, works also together with Hadoop MapReduce.
• Modeled after BigTable* (again first introduced by Google engineers)

The Hadoop Stack of Tools/Infrastructure

- **Pig** (Data Flow)
- **Hive** (SQL)
- **MapReduce** (Distributed Programming Framework)
- **HCatalog** (Metadata)
- **HBase** (Columnar Storage)
- **HDFS** (Hadoop Distributed File System)
- **Zookeeper** (Coordinator)

**Languages**

**Computation**

**Table storage**

**Storage**

Distributed Data Management, SoSe 2013, S. Michel
HBase Data Model

• “Table” layout of data
• Each row is identified by a key Columns are grouped by so called column families
• Inside column family: columns and their values

<table>
<thead>
<tr>
<th></th>
<th>column family 1</th>
<th>column family 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>row1</td>
<td>A=&gt;10</td>
<td>X=&gt; 3.14159</td>
</tr>
<tr>
<td>row2</td>
<td>D=&gt;“hello”</td>
<td>X=&gt; 2.7182</td>
</tr>
</tbody>
</table>

Distributed Data Management, SoSe 2013, S. Michel
HBase Data Model: Map (Dictionary) Style

Key_1 => COLUMN_FAMILY_ONE => A => Value
       B => Value
 => COLUMN_FAMILY_TWO => C => Value

Key_2 => COLUMN_FAMILY_ONE => A => Value
       => COLUMN_FAMILY_TWO => D => Value
       E => Value

....
Column Families and Columns

- column family identifier has to be “printable” name
- need to be defined at table creation

- columns of a certain column family are stored together, physically
- column names can be arbitrary
- columns can be added at runtime
HBase Shell Example Usage

create 'cars', 'features'

put 'cars', 'row1', 'features:make', 'bmw'
put 'cars', 'row1', 'features:model', '5 series'
put 'cars', 'row1', 'features:year', '2012'

put 'cars', 'row2', 'features:make', 'mercedes'
put 'cars', 'row2', 'features:model', 'e class'
put 'cars', 'row2', 'features:year', '2011'

Read on: http://akbarahmed.com/2012/08/13/hbase-command-line-tutorial/
HBase (Shell) Queries

scan 'cars'  // returns all entries of this table
get 'cars', 'row1'  // get row with key row1

// select only specific columns
scan 'cars', {COLUMNS => ['features:make']}

// restrict results to a certain number
scan 'cars', {COLUMNS => ['features:make'], LIMIT => 1}

No SQL, No Joins
HBase Java Example

Configuration config = HBaseConfiguration.create();
HTable table = new HTable(config, "cars");

Put p = new Put(Bytes.toBytes("rowid"));

p.add(Bytes.toBytes("features"), Bytes.toBytes("make"), Bytes.toBytes("bmw"));

table.put(p);
HBase: Other Properties

- **Time to Live (TTL)** can be set per column family
- Multiple **versions** of can be kept
- **Bloomfilter** (remember?) can be used to speed up looking up if row/column exists (inserting row + column family + column family qualifier, or just row)
- **Consistency** guaranteed per single row, using Multiversion concurrency control (**MVCC**)
- Use of **write-ahead logging** (**WAL**)
HBase Architecture

- rows are assigned to so called RegionServers

source: larsgeorge.com
NoSQL: Wide Spectrum

- Systems come with different properties.
- In-memory vs. disk based.
- ACID vs. BASE
- CRUD, SQL (subset?), or MapReduce support

- **Different systems for specific requirements.** Often triggered by demands inside companies.
- Like Voldemort at Linkedin, BigTable (also Hadoop) at Google, etc.
Overview of **Forthcoming Topics**

- Consistency: CAP Theorem. ACID vs. Base.
- Placement of data/nodes in network: Consistent hashing.
- Ordering of events: Vector Clocks
- Will look at sample systems, with hands-on experience through exercises.
Wanted Properties

• Data should be always consistent

• Provided service should be always quickly responding to requests

• Data can be (is) distributed across many machines (partitions)

• Even if some machines fail, the system should be up and running
Recap ACID paradigm of traditional RDBMS

- **Atomicity**
  - transaction is executed as a whole or not at all

- **Consistency**
  - transaction leaves database in consistent state

- **Isolation**
  - single transaction sees database as if it is the only transaction (client)

- **Durability**
  - changed of a successfully finished (committed) transaction is persistently stored
CAP Theorem (Brewer's Theorem)

• System cannot provide all 3 properties at the same time:
  – Consistency
  – Availability
  – Partition Tolerance

Consistent + Available

- No support of multiple partitions.
- Strong (ACID) consistency enforced.

- Example: Single-site database
Partition Tolerant + Available

• Example: Domain Name Service (DNS)
Consistent + Partition Tolerant

- Example: Distributed Databases with distributed locking/commit, HBase
Can’t do without “P”

• Large data => Scale out architecture => Partition Tolerance is a strict requirement
• Leaves: Trading off consistency and availability
Idea

• Trade off consistency and availability
• Write changes to subset of machines
• Read data from subsets of machines

• Maintain multiple versions per data item
• Resolve conflicts based on version mismatch once it occurs (be optimistic)
Generic Architecture

Client

Coordinator

write request

ACK

write requests (or subset and sync between nodes)

response

node1

node2

nodeN
Reads and Writes to Subset of Nodes before Returning to Client
Writes and Sync

node1

D0

node2

D0

write

time

D1

sync

node1

node2

D0

D1

node1

node2

D1
Architecture/Problem

- Writes are propagated to machines + sync between
- Can arrive (and be acknowledged) at different times.
- Leading to inconsistencies
Inconsistent Read

Can be resolved by using the latest version.

different data “version”

node1

D0

node2

D1
Consistent Read

node1

D0

read

node2

D0

read

Distributed Data Management, SoSe 2013, S. Michel
**CAP Theorem: Proof Idea**

- Consider a system with multiple partitions
- Failure prevents sync between node1 and node2.

- Now? Prohibit reads until synced? (system not available). Or let clients read (system not consistent)
Best effort: BASE

• Basically Available
• Soft State
• Eventual Consistency

see http://www.allthingsdistributed.com/2007/12/eventually_consistent.html

W. Vogels. Eventually Consistent. ACM Queue vol. 6, no. 6, December 2008.
Eventual Consistency

• After a write, data can be at some nodes/machines inconsistent
• But will eventually(!) become consistent again
Literature (Other than books)

- [http://hadoop-hbase.blogspot.de/2012/03/acid-in-hbase.html](http://hadoop-hbase.blogspot.de/2012/03/acid-in-hbase.html)