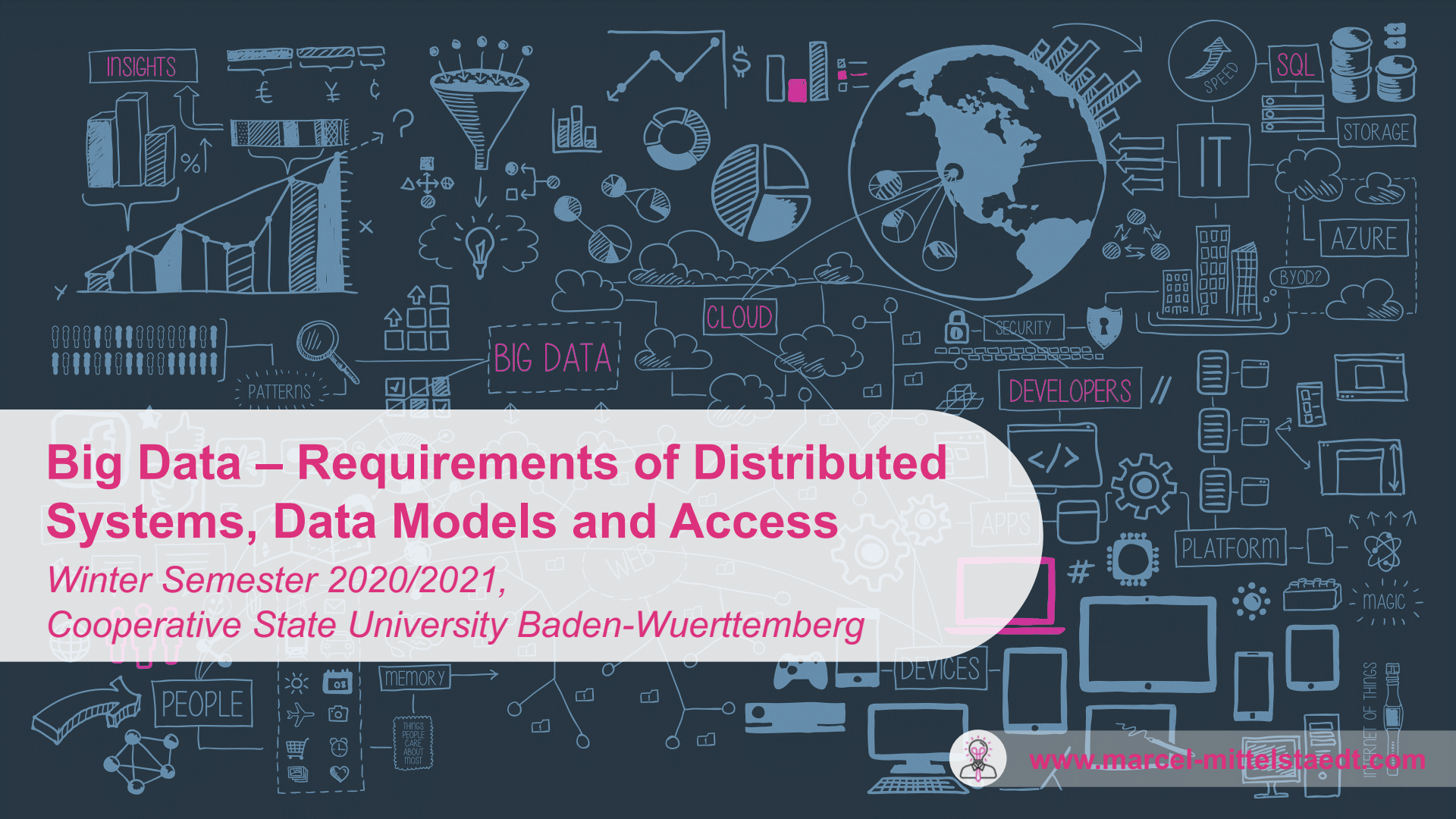


Big Data – Requirements of Distributed Systems, Data Models and Access

Winter Semester 2020/2021,
Cooperative State University Baden-Wuerttemberg

www.marcel-mittelstaedt.com

Agenda – 15.02.2021

01

Presentation and Discussion: Exercises Of Last Lecture

Setup Google Cloud, Hadoop, HDFS, YARN, MapReduce

02

Introduction To Distributed Systems, Data Models and Access

(Non-)Functional Requirements Of Distributed Data-Systems, Data Models And Access

03

HandsOn – Java MapReduce, Hive, HiveQL and IMDb Data

Quick Introduction To The MapReduce Programming Paradigm, Java MapReduce, Hive, HiveQL

04

Exercise

Write, Compile and Run MapReduce Job in Java, Hive, HiveQL and external Tables

Schedule

	Lecture Topic	HandsOn
08.02.2021 13:00-15:45 Ro. N/A	About This Lecture, Introduction to Big Data	Setup Google Cloud, Create Own Hadoop Cluster and Run MapReduce
15.02.2021 13:00-15:45 Ro. N/A	**(Non-)Functional Requirements Of Distributed Data-Systems, Data Models and Access**	Hive and HiveQL
22.02.2021 13:00-15:45 Ro. N/A	Challenges Of Distributed Data Systems: Partitioning	Data Partitioning (with HDFS/Hive, HiveServer2)
01.03.2021 13:00-15:45 Ro. N/A	Challenges Of Distributed Data Systems: Replication	Spark and Scala
08.03.2021 13:00-15:45 Ro. N/A	ETL Workflow and Automation	PySpark and Notebooks (Jupyter)
15.03.2021 13:00-15:45 Ro. N/A	Batch and Stream Processing	Airflow
22.03.2021 13:00-15:45 Ro. N/A	Practical Exam	Work On Practical Exam
29.03.2021 13:00-15:45 Ro. N/A	Practical Exam	Work On Practical Exam
12.04.2021 13:00-15:45 Ro. N/A	**Practical Exam Presentation**	
19.04.2021 13:00-15:45 Ro. N/A	**Practical Exam Presentation**	

Solution – Exercise I

Hadoop, HDFS, YARN, MapReduce Example

Solution

Prerequisites:

- install Ubuntu 18.04
- Install Java JDK 1.8.0
- Create Hadoop user
- Install and Setup SSH (*public/private key authentication, authorized_keys, …*)
- Install and Configure Hadoop 3.1.2 (*pseudo-distributed mode*)
- Start HDFS and YARN
- Clone Git Repo:

```
git clone https://github.com/marcelmittelstaedt/BigData.git
```


Solution

Exercise 2:

1. Copy sample file from GIT repo to HDFS user directory:

```
hadoop fs -put BigData/exercises/winter_semester_2020-2021/01_hadoop/sample_data/Faust_1.txt
/user/hadoop/Faust_1.txt
```

2. Use and run default MapReduce Jar (*hadoop-mapreduce-examples-3.1.2.jar*) to calculate **wordcount** for text file „*Faust_1.txt*“.

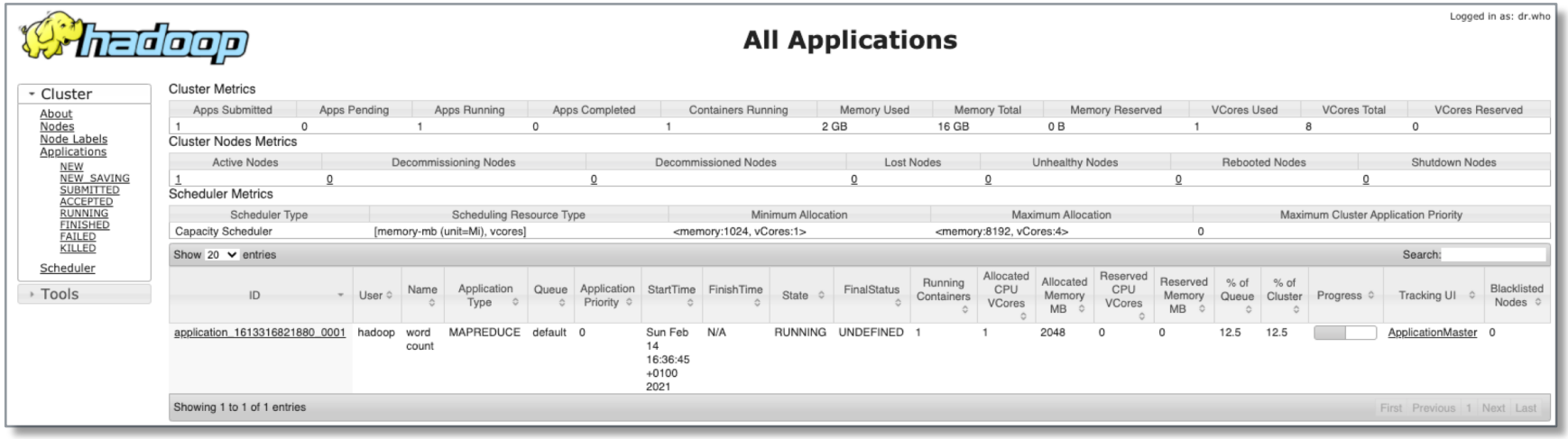
```
hadoop jar hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.1.2.jar wordcount /user/hadoop/Faust_1.txt /user
/hadoop/Faust_1_Output

[…]
2021-02-14 15:36:46,163 INFO mapreduce.Job: The url to track the job: http://big-data.c.dhbw-254309.internal:8088/proxy/application
_1613316821880_0001/
2021-02-14 15:36:46,164 INFO mapreduce.Job: Running job: job_1613316821880_0001
2021-02-14 15:36:54,318 INFO mapreduce.Job: Job job_1613316821880_0001 running in uber mode : false
2021-02-14 15:36:54,319 INFO mapreduce.Job:  map 0% reduce 0%
2021-02-14 15:37:00,404 INFO mapreduce.Job:  map 100% reduce 0%
2021-02-14 15:37:05,448 INFO mapreduce.Job:  map 100% reduce 100%
2021-02-14 15:37:06,462 INFO mapreduce.Job: Job job_1613316821880_0001 completed successfully
[…]
```


www.marcel-mittelstaedt.com

Solution

Exercise 2:

3. Take a look at Ressource Manager for Job Execution (http://XXX.XXX.XXX.XXX:8088/cluster/apps/RUNNING):



Solution

Exercise 2:

4. a) Copy MapReduce output file back to ubuntu local filesystem (using bash):

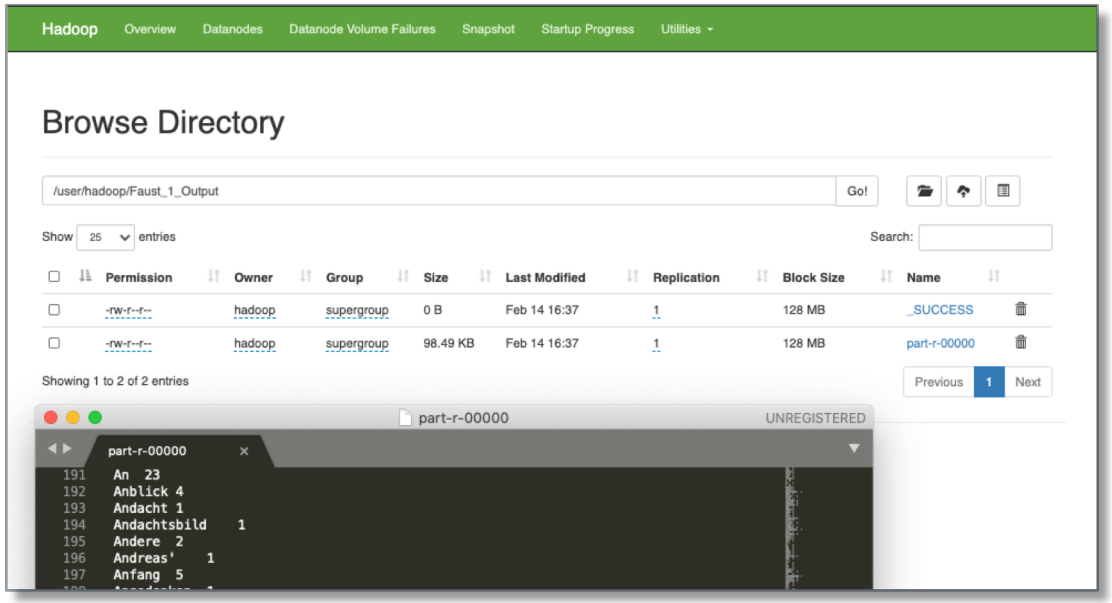
```
hadoop fs -get /user/hadoop/Faust_1_Output/part-r-00000 Faust_1_Output.csv
```

```
head -10 Faust_1_Output.csv

Allein 1
Alles 1
Als 1
Der 1
Die 2
Er 2
Ich 4
Im 1
Mein 1
Nur 1
```


Solution

Exercise 2:

4. b) Copy MapReduce output file back to ubuntu local filesystem (using Web Filebrowser):



Exercise 3

MapReduce Examples within *hadoop-mapreduce-examples-3.1.1.jar*:

aggregatewordcount:	An Aggregate based mapreduce program that counts the words in the input files.
aggregatewordhist:	An Aggregate based mapreduce program that computes the histogram of the words in the input files.
bbp:	A mapreduce program that uses Bailey-Borwein-Plouffe to compute exact digits of Pi.
dbcount:	An example job that counts the pageview logs stored in a database.
distbbp:	A mapreduce program that uses a BBP-type formula to compute exact bits of Pi.
grep:	A mapreduce program that counts the matches of a regex in the input.
join:	A job that performs a join over sorted, equally partitioned datasets.
multifilewc:	A job that counts words from several files.
pentomino:	A mapreduce tile laying program to find solutions to pentomino problems.
pi:	A mapreduce program that estimates Pi using a quasi-Monte Carlo method.
randomtextwriter:	A mapreduce program that writes 10 GB of random textual data per node.
randomwriter:	A mapreduce program that writes 10 GB of random data per node.
secondarysort:	An example defining a secondary sort to the reduce phase.
sort:	A mapreduce program that sorts the data written by the random writer.
sudoku:	A sudoku solver.
teragen:	Generate data for the terasort.
terasort:	Run the terasort.
teravalidate:	Checking results of terasort.
wordcount:	A mapreduce program that counts the words in the input files.
wordmean:	A mapreduce program that counts the average length of the words in the input files.
wordmedian:	A mapreduce program that counts the median length of the words in the input files.
wordstandarddeviation:	A mapreduce program that counts the standard deviation of the length of the words in the input files.

Solution

Exercise 3:

1. Copy sample file from GIT repo to HDFS user directory:

```
hadoop fs -put BigData/exercises/winter_semester_2019-2020/01_hadoop/sample_data/Faust_1.txt /user/hadoop/Faust_1.txt
```

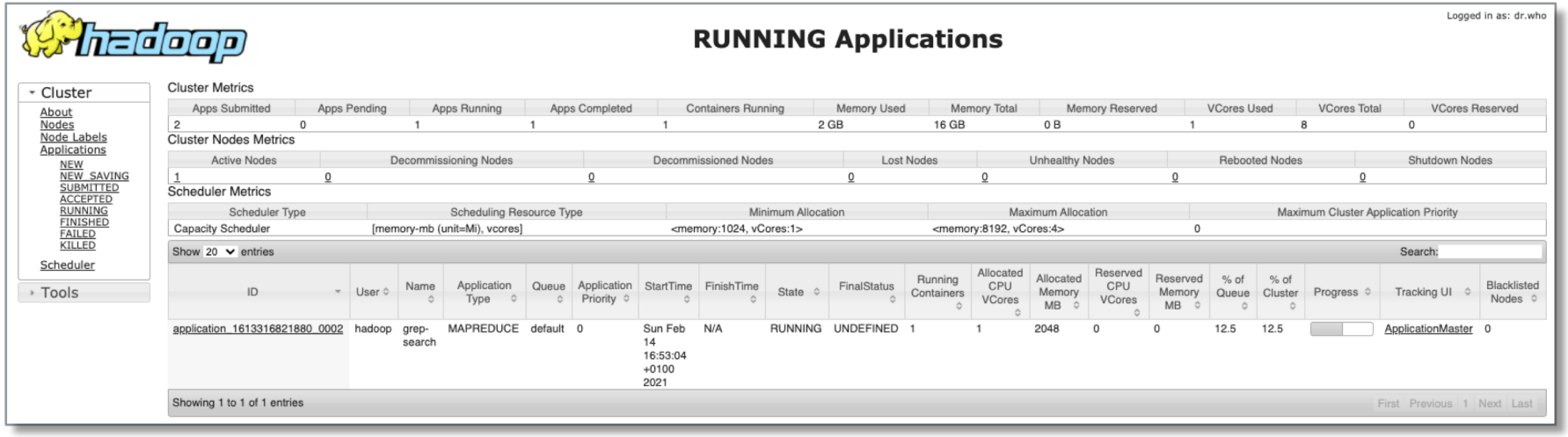
2. Use and run default MapReduce Jar (*hadoop-mapreduce-examples-3.1.2.jar*) to **grep** for string „Faust“ in text file „*Faust_1.txt*“ and count appearances of string.

```
hadoop jar hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.1.2.jar grep /user/hadoop/Faust_1.txt /user/hadoop/Faust_1_Count_Output 'Faust'

[…]
2019-10-12 16:44:16,517 INFO mapreduce.Job: Running job: job_1570893575375_0008
2019-10-12 16:44:27,637 INFO mapreduce.Job: Job job_1570893575375_0008 running in uber mode : false
2019-10-12 16:44:27,638 INFO mapreduce.Job:  map 0% reduce 0%
2019-10-12 16:44:31,678 INFO mapreduce.Job:  map 100% reduce 0%
2019-10-12 16:44:36,717 INFO mapreduce.Job:  map 100% reduce 100%
2019-10-12 16:44:37,735 INFO mapreduce.Job: Job job_1570893575375_0008 completed successfully
[…]
```

Solution

Exercise 3:

3. Take a look at Ressource Manager for Job Execution (http://XXX.XXX.XXX.XXX:8088/cluster/apps/RUNNING):



Solution

Exercise 2:

4. a) Copy MapReduce output file back to ubuntu local filesystem (using bash):

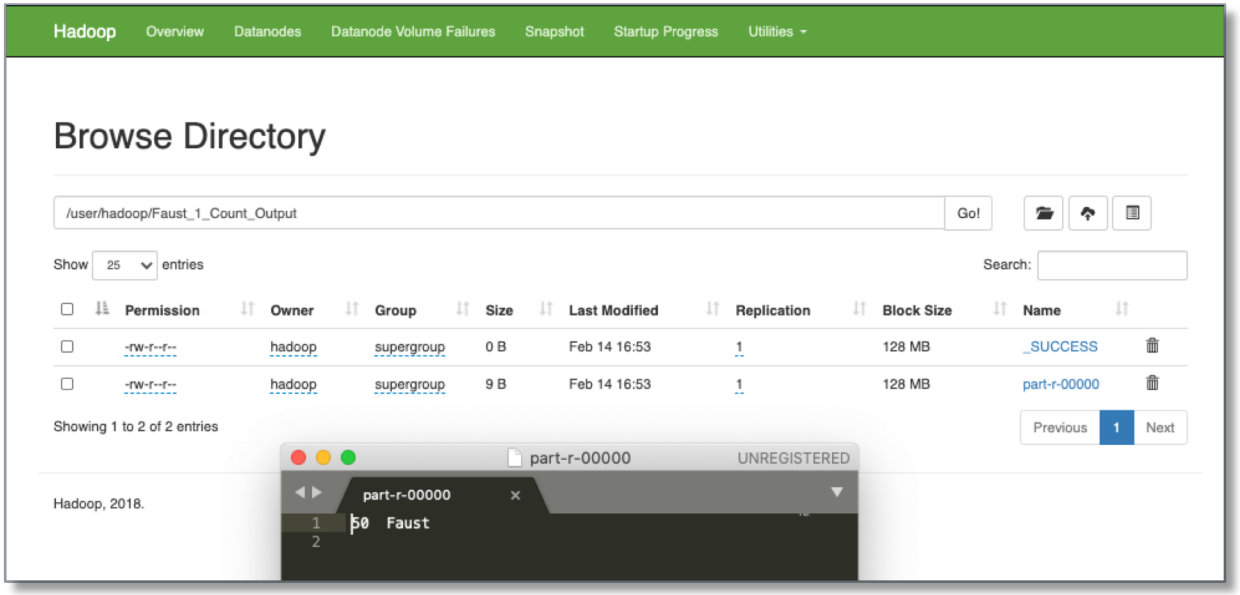
```
hadoop fs -get /user/hadoop/Faust_1_Count_Output/part-r-00000 Faust_1_Count_O
utput.csv
```

```
cat Faust_1_Count_Output.csv

50        Faust
```

Solution

Exercise 2:

4. b) Copy MapReduce output file back to ubuntu local filesystem (using Web Filebrowser):



Introduction To Requirements Of Distributed Systems

(Non-)Functional Requirements Of Distributed Data-Systems

Requirements Of A Data-System

Data Storage: We need to store data and also need to be able to find it again later (*database*).

Data Querying: We need to be able to query and filter data efficiently in certain kinds of ways (*transaction* and *indices*).

Retention and Performance: We want results fast, especially of expensive read operations (*caching*).

Data Processing: We want to be able to process a huge amount of data (*batch processing*) as well as process data asynchronously (*stream processing*).

Requirements Of A Data-System

- same requirements as for the first database ***CODASYL*** (back in the 1960's)

- even though there have been a lot of databases back in time - all of them still match those same requirements.

- Evolution:
  - **relational databases** being able to handle NoSQL data (e.g. even "retirees" like *IBM DB2* or *Oracle*) as well as **NoSQL databases** being able to handle traditional SQL (e.g. ToroDB)
  - **databases** becoming message queues (e.g. *RethinkDB* or *Redis*) and the other way around **message queueing systems** become databases (e.g. Ap ache Kafka)

  → boundaries blurr

Scalability

Scalability *is the capability of data system to handle a growing amount of load (e.g. a larger amount of data needed to store or requests to handle). A data-system is considered scalable if its capable of increasing it's total through-/output under an increased load when resources (typically hardware) are added.*

- Scalability is **NOT**:
  - **a binary tag** (scalable/not-scalable), that could be attached to a data-system.

  → For any scalable data-system you need to think about:
  *„If the **load** of a data system grows in a certain way, what are the options on the table for **coping with the growth**?"*
  *"How can we **add ressources** (hardware) to be able to **handle the additional load**?"*

Scalability - **Load**

The ***load*** *of a data system is a measurement of the amount of computational work it performs (depending on the architecture in-place), e.g. the number of (concurrent) reads from a data storage, writes to a data storage or the ratio be tween reads and writes. The maximum load is defined by the weakest part of the architecture (=bottleneck).*

- E.g.
  - **requests/second** to a website
  - **read/write requests/second** to a data-system
  - **read/write ratio** of a data-system
  - **cash hit-rate**

Scalability - **Performance**

Performance *of a data-system is defined by system* ***throughput and response time****, e.g. number of transactions (like read/write operations), processed records (like aggregations for analytical purposes) or even system commands (like an update statistics or re-balancing of several data nodes)* ***under a given workload and for a specific time-frame****.*
It usually depends on a variety of influencable as well as uninfluencable factors of the sy stem itself, like network latency, a page fault or damaged disk.

→ The *load parameter* **increases**, but all ressources (e.g. number of server, CPU or memory) stay the same - **how is the performance of the data system affected?**

→ The *load parameter* **increases** – **how much** do you need **to increase the ressources** (e.g. number of server, CPU or memory) to keep the performance stay the same?

Scalability – **Performance Measurement**

E.g. **Throughput:**

- **read/writes per second** (in case of MongoDB up to 100.000 reads/ writes per second)
- **messages processed** (in case of Apache Kafka and Linkedin more than 2 million records per second on just 3 nodes)
- **data processed** (in case of Apache Hadoop and MapReduce terabytes of data within several seconds)

Or **Response Time:**

- **read/writes per second** (in case of MongoDB up to 100.000 reads/ writes per second)

Scalability – **Performance Measures**

Arithmetic mean:

- easy to calculate
- ignores ratios
- is highly affected by statistical outliers, so it cannot tell you how many requests, reads or writes actually have had a worse performance

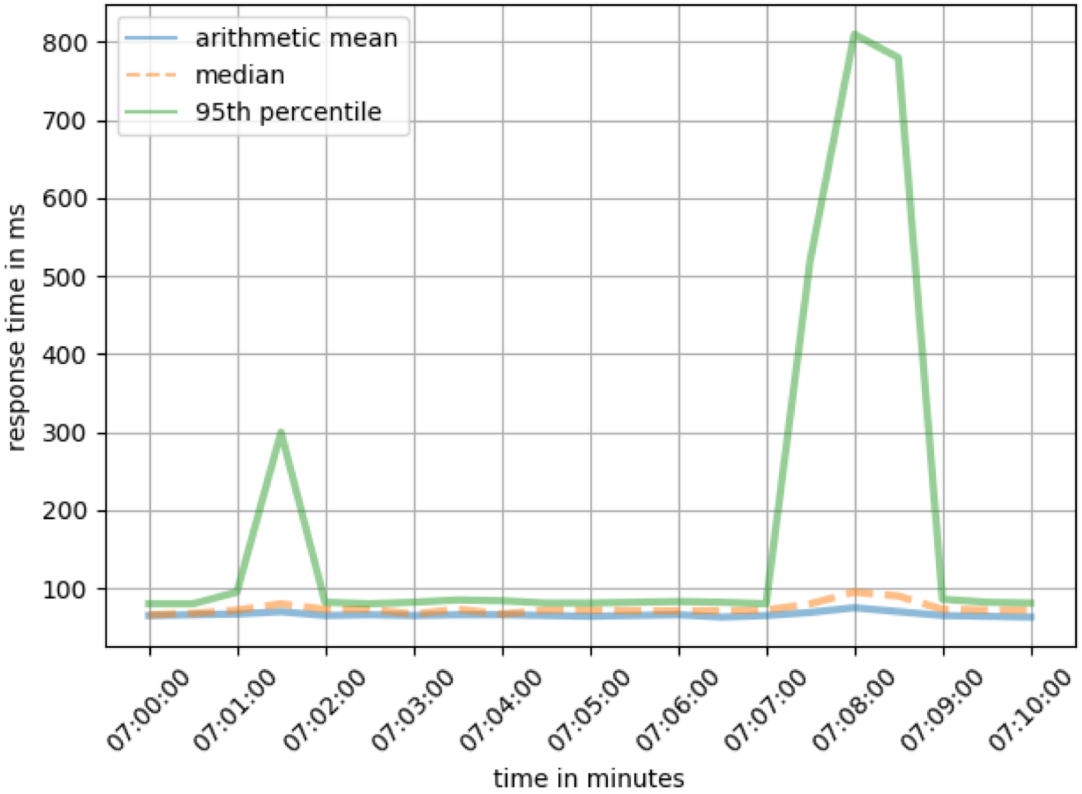
Median:

- easy to calculate
- less distorted by outliers

Percentiles (e.g. p95, p99, p999 percentiles)**:**

- easy to calculate
- not distorted by outliers

Scalability – Performance Measures



Scalability – Aproaches For Scaling

Scale Up/Vertical:

- replace a server by a more powerful one

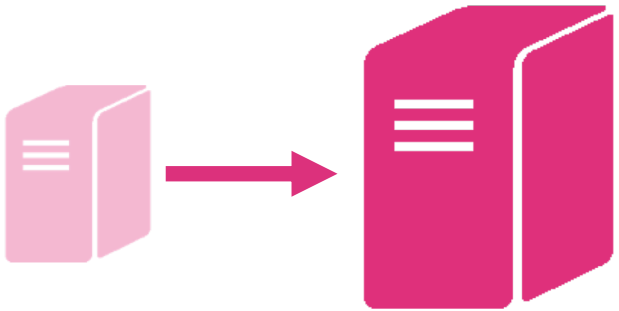
→ easier for a development

→ more expensive in terms of hardware

Scale Out/Horizontal:

- distribute the load towards multiple server instead of one

- development more complex
- cheaper in terms of hardware



→ But ist always about a mix of both

Reliability

Reliability *in terms of hardware, software or especially data-systems can be defined as the ability of a system to function as specified and expected. A reliable data-system also detects and tolerates faults due to mistakes of users, hardware or lower parts of the data-system itself as well as ensures the required performance under any expected load.*

- Or more pragmatic: "*something works correctly even if things go wrong*"

- *Typical Faults:*
  - ***Hardware** faults*
  - ***Software** faults*
  - ***Human** faults*

Reliability – Hardware Faults

- E.g.:
  - **broken HDDs** or **SSDs**
  - **faulty RAM** or **CPUs**
  - **broken power adapters**, **switches** or **whole network outages**
  - **unplugged network cables** or even connected to the wrong port
  - and many many more…

- **Seems very unlikely?**
  - imagine a 100 node Hadoop Cluster with 19 HDDs per Node = 1.900 HDDs overall
  - According to BackBlaze the average annual HDD failure rate is about 2.11%

  → approximately **each week an HDD will fail**

https://www.backblaze.com/blog/hard-drive-failure-rates-q1-2017/

Reliability – **Software Faults**

- E.g.:
  - a **runaway** and/or **zombie process** that extensively used up some shared ressource (e.g. network, CPU, RAM, disk space)
  - a **software bug** causing the whole cluster to fail (e.g. the Hadoop Ressource Manager YARN once had a bug, that if you removed a cgroup under some circumstances (*race conditions*) a *kernel panic* and in this way a failure of multi-ple server was caused
  - **bottleneck**: a service the whole data-system depends on slows done, becomes unresponsive or fails
  - **cascading failures** (e.g. one server of the data-system fails due to heavy network traffic, causing the other servers to take over
    → in this way increasing network traffic for them too and finally all server will fail)

https://issues.apache.org/jira/browse/YARN-2809

Reliability – Human Faults

- Most unreliable factor: **humans**

- Approaches for making a data-system reliable, even in terms of **humans**:

  - decouple places where people make the **most failures** from **places they can cause failures**, e.g. using production and development environments or providing interfaces or frameworks for an API instead of direct API access
  - **use extensive testing** (e.g. unit test, system tests, integration tests) and automize them
  - **measuring and monitoring** (e.g. performance metrics, error rates) all ows to check wether assumptions or constraints are violated at an earl y stage

Maintainability

- **Maintenance** = one of the biggest costs in software development
- A data-system should be designed to minimize maintenance effort

- 3 Main principles to follow:

  - **1. Operability:** *make it easy to run the data-system*
    - **good documentation** and **operational model** (a data-system which can be understand easily can be operated more easily)
    - **transparency** (visibility into the data system and runtime behaviour, e.g. by log files or monitoring tools)
    - **no dependencies** between single services or server (allow single server to go down for maintenance tasks, e.g. patches, update or restarts)
    - **self-healing** if possible, but also possibilities to override for operators

Maintainability

- **2. Simplicity:** *make it easy to understand the data-system*
  - make use of **abstraction** and **reduce complexity** (less complex doesn't require reducing functionaility, it's more about removing unnecessary complexity)
  - **clearly defined interfaces**
  - **no over-engineering**

- 3. **Evolvability:** *make it easy to adapt/change the data-system*
  - **continues integration**
  - **test-driven development**
  - **pair-programming**
  - …

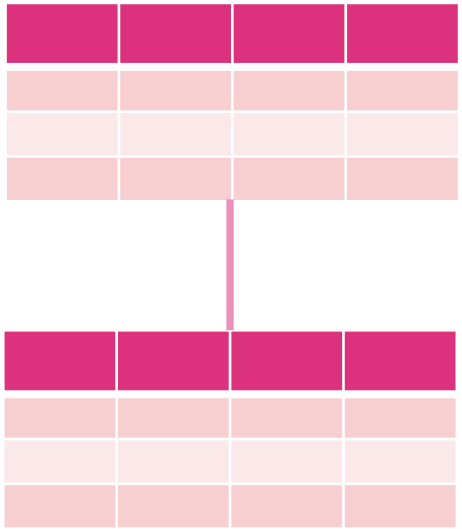
Introduction To Data Models And Access

Relational and Non-Relational Data Models, Data Access

Data Models (Relational/Non-Relational)

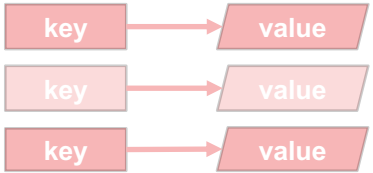
RELATIONAL

Row/Column Based

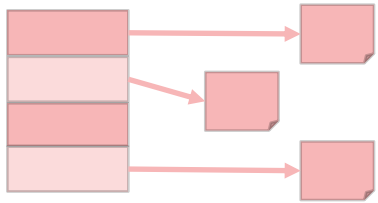


NON-RELATIONAL

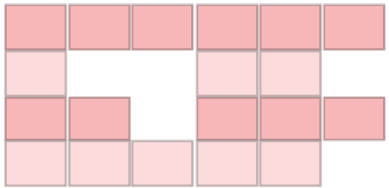
Key-Value



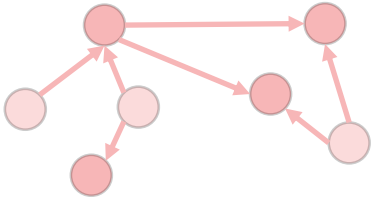
Document



Column Family



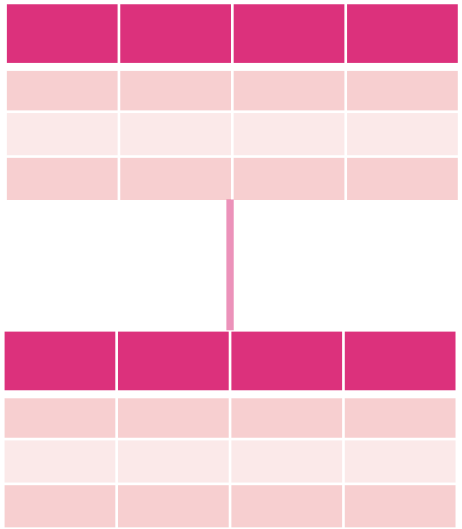
Graph



Relational Data Model

RELATIONAL

Row/Column Based



NON-RELATIONAL

Key-Value

Document

Column Family

Graph

Relational Data Model

- Originally introduced by **Edgar Frank Codd** in 1970

*The **relational model** is an approach to managing data using a structure and language consistent with first-order predicate logic where all data is represented in terms of tuples and grouped into relations.*

- Idea of: hide implementation details (representation of data in data store)
- By providing: a *declarative* and *read-on-schema* interface

→ Developers/user can easily state what information the database contains and what information they want
→ The database will take care of describing, storing and retrieving data

Relational Data Model - Example

- **Example**: Facebook Profile Page as relational model

- table `user` = main entity
  - stores *primary key* (`id`)
  - stores basic information (e.g. `first_name`, `last_name`)

- As people may have:
  - worked in multiple companies
  - studied at multiple universities
  - lived at different cities

  → separate tables with *foreign key* (`user_id`):
    - `companies`
    - `universities`
    - `cities`

table: ***user***

id	first_name	last_name	is_verified	married_to	...
1	Mark	Zuckerberg	true	4711	
4711	Priscilla	Chan	true	1	

table: ***universities***

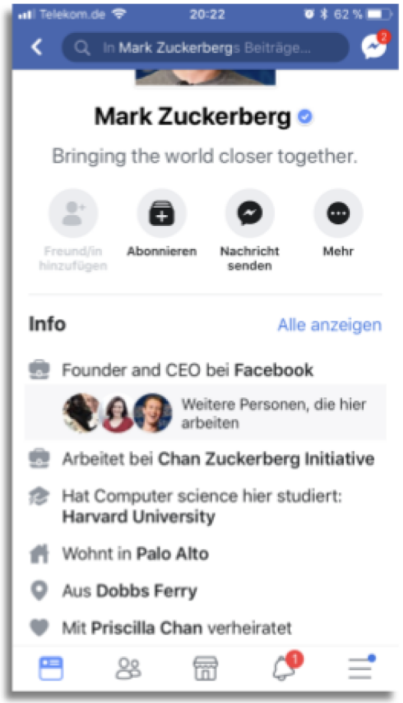
id	user_id	university_name	subject	...
1378	1	Harvard University	Computer Science	
1379	4711	University of California	Medicin	

table: ***companies***

id	user_id	company_name	...
1337	1	Facebook	
1338	1	Chan Zuckerberg Initiative	
1339	4711	UCSF Benioff Children's Hospital	

table: ***cities***

id	user_id	city_name	status	...
1378	1	Palo Alto	living	
1379	1	Dobbs Ferry	born	
1379	4711	Palo Alto	living	



Relational Data Model – List Of Software

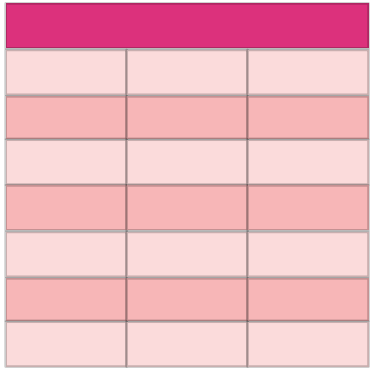
List of Software [edit]

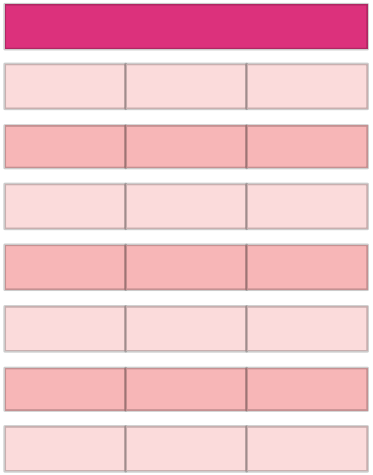
- 4th Dimension
- Adabas D
- Alpha Five
- Apache Derby
- Aster Data
- Amazon Aurora
- Altibase
- CA Datacom
- CA IDMS
- Clarion
- ClickHouse
- Clustrix
- CSQL
- CUBRID
- DataEase
- Database Management Library
- Dataphor
- dBase
- Derby aka Java DB
- Empress Embedded Database
- EXASolution
- EnterpriseDB
- eXtremeDB
- FileMaker Pro
- Firebird
- FrontBase
- Google Fusion Tables
- Greenplum
- GroveSite
- H2
- Helix database
- HSQLDB
- IBM DB2
- IBM Lotus Approach
- IBM DB2 Express-C
- Infobright
- Informix
- Ingres
- InterBase
- InterSystems Caché
- LibreOffice Base
- Linter
- MariaDB
- MaxDB
- MemSQL
- Microsoft Access
- Microsoft Jet Database Engine (part of Microsoft Access)
- Microsoft SQL Server
- Microsoft SQL Server Express
- SQL Azure (Cloud SQL Server)
- Microsoft Visual FoxPro
- Mimer SQL
- MonetDB
- mSQL
- MySQL
- Netezza
- NexusDB
- NonStop SQL
- NuoDB
- Omnis Studio
- Openbase
- OpenLink Virtuoso (Open Source Edition)
- OpenLink Virtuoso Universal Server
- OpenOffice.org Base
- Oracle
- Oracle Rdb for OpenVMS
- Panorama
- Pervasive PSQL
- Polyhedra
- PostgreSQL
- Postgres Plus Advanced Server
- Progress Software
- RDM Embedded
- RDM Server
- R:Base
- SAND CDBMS
- SAP HANA
- SAP Adaptive Server Enterprise
- SAP IQ (formerly known as Sybase IQ)
- SQL Anywhere (formerly known as Sybase Adaptive Server Anywhere and Watcom SQL)
- solidDB
- SQLBase
- SQLite
- SQream DB
- SAP Advantage Database Server (formerly known as Sybase Advantage Database Server)
- Teradata
- Tibero
- TimesTen
- Trafodion
- txtSQL
- Unisys RDMS 2200
- UniData
- UniVerse
- Vectorwise
- Vertica
- VoltDB

https://en.wikipedia.org/wiki/List_of_relational_database_management_systems

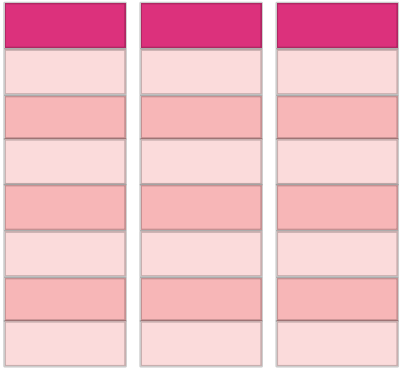
Relational Data Model – **Row vs. Column-Based**

Row-Based:
→ rows are stored continuesly
→ e.g. Oracle, IBM DB2, Microsoft SQL Server, MySQL
→ some also support column-based





Column-Based:
→ columns are stored continuesly
→ e.g. Hbase, Parquet, SAP HANA, Teradata



Relational Data Model – **Row vs. Column-Based**

Operation	Row-Based	Column-Based
Compression	**Low**	**High**
Column Scans	**Slow** (Multiple Reads)	**Fast** (One Read)
Insert Of Records	**Fast** (One Insert)	**Slow** (Multiple Inserts)
Single Record Queries	**Fast** (One Read)	**Slow** (Multiple Reads)
Single Column Aggregation	**Slow** (Full Table Scan)	**Fast** (Only Column Scan)
Typical Use Cases	**Transactional**	**Analytical**

Relational Data Model - SQL

- Originally introduced as „SEQUEL“ by **Donald D. Chamberlin** and **Raymond F Boyce** in 1974 and inspired by **Edgar Frank Codds** relational data model in 1970
- ANSI Standard 1958, ISO 1987

- → todays **most widely** used **database language**

- → **even by NoSQL** data-systems, e.g.:
  - Apache Cassandra (*CQL*)
  - Apache Hadoop (*HiveQL*)
  - Apache Flink (*FQL*)

Relational Data Model - SQL

Language:

- SQL = *declarative*
  - → no need to worry about how the data is stored and retrieved
  - → automatic performance optimization *(query optimizer)*
- *imperative* = e.g. MapReduce, Spark, …
  - → need to think about data storage
  - → no query optimizer

Usual DML Statements:

CREATE/DROP/ALTER TABLE…
INSERT INTO … VALUES…
UPDATE … SET … WHERE…
DELETE FROM .. WHERE…
…

Structure:

```
SELECT      -- attributes
FROM        -- tables
WHERE       -- conditions
GROUP BY    -- grouping attributes
HAVING      -- grouping conditions
ORDER BY    -- attributes
```

Keywords:

DISTINCT, LIMIT, AS
MIN, MAX, AVG, COUNT, SUM
AND, OR, NOT, IN, LIKE, ANY
UNION, JOIN
ASC, DESC
…

Relational Data Model – SQL Example

Query:

```
select m.tconst, m.original_title, m.start_year, r.average_rating, r.num_votes
from imdb_movies m JOIN imdb_ratings r on (m.tconst = r.tconst) WHERE r.average_rating > 6
and m.start_year > 2010 and m.title_type = 'movie' and r.num_votes > 100000
ORDER BY r.average_rating desc, r.num_votes desc LIMIT 200
```

Result:

	m.tconst	m.original_title	m.start_year	r.average_rating	r.num_votes
1	tt0816692	Interstellar	2.014	8,6	1.213.141
2	tt4154756	Avengers: Infinity War	2.018	8,6	492.952
3	tt1675434	Intouchables	2.011	8,5	635.669
4	tt2582802	Whiplash	2.014	8,5	571.172
5	tt5074352	Dangal	2.016	8,5	105.207
6	tt1345836	The Dark Knight Rises	2.012	8,4	1.331.811
7	tt1853728	Django Unchained	2.012	8,4	1.153.183
8	tt2380307	Coco	2.017	8,4	219.957
9	tt5311514	Kimi no na wa.	2.016	8,4	108.553
10	tt2106476	Jagten	2.012	8,3	226.819
11	tt1832382	Jodaeiye Nader az Simin	2.011	8,3	186.217
12	tt0993846	The Wolf of Wall Street	2.013	8,2	976.551
13	tt2096673	Inside Out	2.015	8,2	499.721
14	tt1291584	Warrior	2.011	8,2	389.935
15	tt3170832	Room	2.015	8,2	288.153
16	tt5027774	Three Billboards Outside Ebbing, Missouri	2.017	8,2	280.790

Relational Data Model - SQL

- **Strenghts:**
  - Consistency → ACID
  - Universal → a lot of data types, linked and unlinked data, “Independance“ of RDBS
  - Strict Schema → Data Quality (*Garbage-In – Garbage-Out*), Error Prevention, Compression

- **Weaknesses:**
  - Strict Schema:
    → needs to be altered at any data format change
    → data needs to be migrated
  - Object-Relational-Impedance Mismatch:
    - E.g. objects, structs, ...
    - needs ORMs
    - usually slows and complicates data access

Relational Data Model – **Advantages/Disadvantages**

- **Data Fetching:** rows can easily be inserted and fetched all-at-once by using an `id` or a *primary/foreign key,* unlike e.g. document-oriented data models, where you usually need to:
  - make use of ***access paths***
  - need to think about **nested structures**
  - need to worry about **unknown fields** as those systems are usually *schema-on-read*
  - or intensively need to **think about possible performance issues** and how the system will probably execute your query as the execution engine is usually not that mature as the one of a relational system
  - it's more **difficult to understand how the systems works**, as sometimes you don't even have a *query-explain* feature like in relational databases, so you won't be able to investigate or guess in advance how it will behave in detail during execution.

Relational Data Model – **Advantages/Disadvantages**

- **Object-Orientation:**
  applications need *translation layer* for data-system: **ORM Frameworks,** like e.g.*:*
  - ***Hibernate*** in case of Java
  - ***SQLAlchemy*** in case of Python

- **Many-to-many relations**:
  - **relational data model**: foreign key constraints
    → inexpensive (indices etc.)
  - **NoSQL data model**: nested structures, document references
    → expensive IO/CPU operations

- **Writes/Updates**:
  - **relational data model**: efficient, as single rows will be updated
  - **NoSQL data model**: usually requires rewriting a whole document

Relational Data Model – **Advantages/Disadvantages**

- **Constraints:**
  - **relational data model**:
    - Data Types
    - Primary/Foreign Keys
    - …

    → highly serves **data integrity**
  - **NoSQL data model**:
    - No or less constraints

    → Freedom but **vulnerable** to *garbage-in – garbage-out*

- **Operations**:
  - **relational data model**: relational algebra

    → mighty language
  - **NoSQL data model**: data-system specific

    → usually less mighty

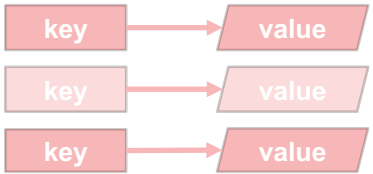
Non-Relational Data Model

RELATIONAL

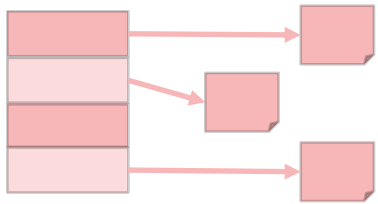
Row/Column Based

NON-RELATIONAL

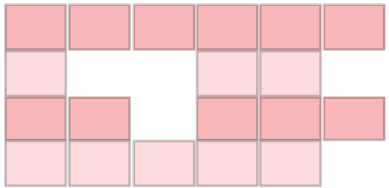
Key-Value



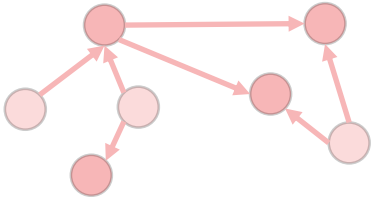
Document



Column Family



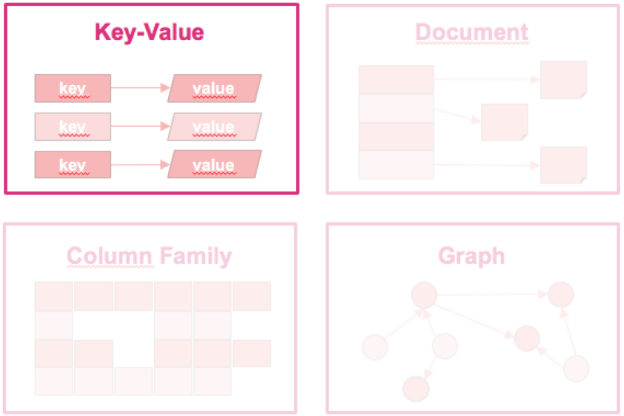
Graph



Non-Relational Data Model

NoSQL in terms of BigData, is a theorem for managing data using document-oriented, key-value or graph structures, where all data is represented in terms of documents, key-value pairs, graphs (nodes, edges, properties) or mixed approaches.

Non-Relational Data Model – **Key-Value**



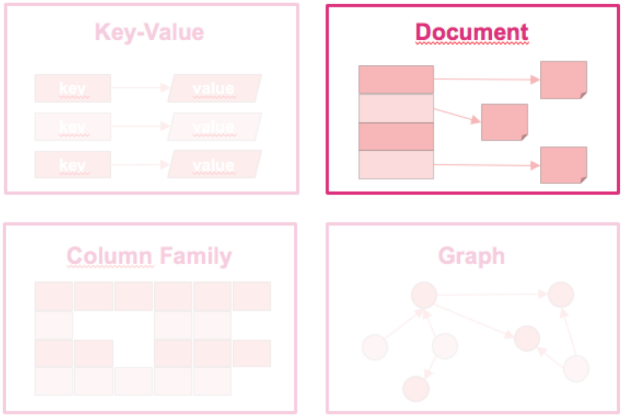
- **Examples:**
  - Redis,
  - BerkeleyDB,
  - VoldemortDB,
  - ArangoDB,
  - Riak, …

- **Strenghts:**
  - **fast queries** (value lookups)
  - **fast inserts** (key-value pairs)
  - **easy to replicate** and **distribute** (e.g. *consistent hashing*)

- **Weaknesses**:
  - **no** or **less efficient** and slow:
    - aggregation
    - filtering
    - joining

    → needs to be done by application

Non-Relational Data Model – **Document**



- **Examples:**
  - MongoDB,
  - Couchbase,
  - RethinkDB,
  - ArangoDB,
  - DynamoDB,
  - CouchDB, …

- **Strenghts:**
  - **schema flexibility** (documents can have different f ormats)
  - **fast inserts** and **point queries**
  - **data locality**
  - **easy to replicate** and **distribute** (e.g. *to achieve l oad balancing and failure tolerance*)
  - **application code simplicity**

- **Weaknesses**:
  - **updates** on documents are usual **less efficient/ IO expensive** (especially if size changes)
  - **no** or **less efficient** and slow:
    - aggregation
    - filtering
    - joining

Non-Relational Data Model – **Document**

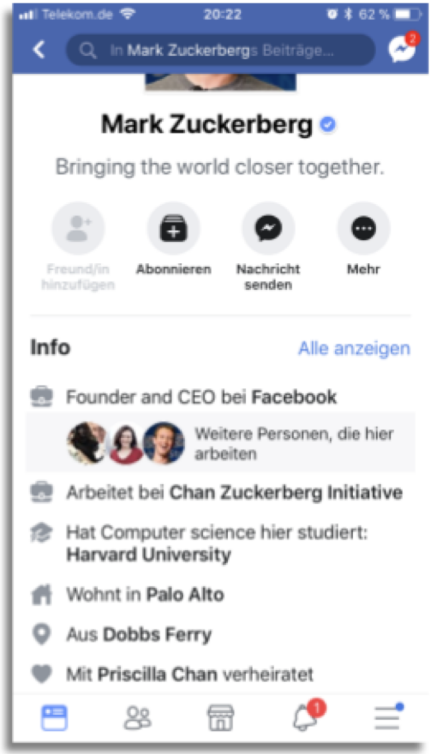
- **Example**: Facebook Profile Page as document model

- No need to fetch mutliple tables (***data locality***)

- **Easy to parse** by applications (*object-oriented*)

- schema flexibility (no expensive ALTER TABLE … needed)

```
{
  "id": 1,
  "first_name": "Mark",
  "last_name": "Zuckerberg",
  "is_verified": "true",
  "married_to": 4711,
  "universities": [
    {
      "university_name": "Harvard University",
      "subject": "Computer Science"
    }
  ],
  "companies": [
    {
      "company_name": "Facebook"
    },
    {
      "company_name": "Chan Zuckerberg Initiative"
    }
  ],
  "cities": [
    {
      "city_name": "Palo Alto",
      "status": "living"
    },
    {
      "city_name": "Dobbs Ferry",
      "status": "born"
    }
  ]
}
```



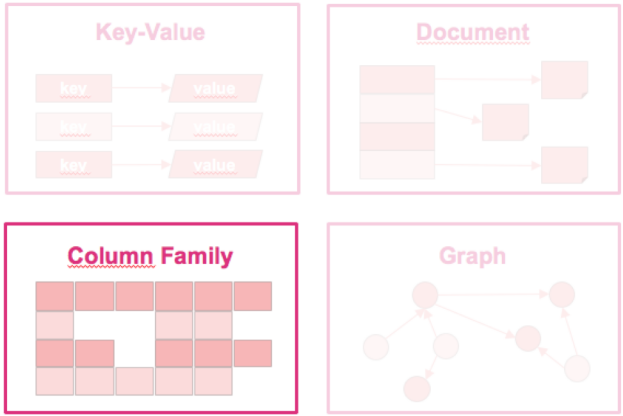
Non-Relational Data Model – **Document**

SQL	MongoDB
SELECT * **FROM** user	`db.user.find()`

SELECT id, first_name, last_name
FROM user **WHERE** is_verified = true
	`db.user.find(`
	`{ is_verified : true },`
	`{ id : 1, first_name: 1,`
	`last_name: 1 }`
	`)`

INSERT INTO user(id, first_name, last_name
VALUES (1, „Mark“, „Zuckerberg“)
	`db.user.insertOne(`
	`{ id : 1,`
	`first_name: “Mark“,`
	`last_name: “Zuckerberg“ }`
	`)`

Non-Relational Data Model – **Column Family**

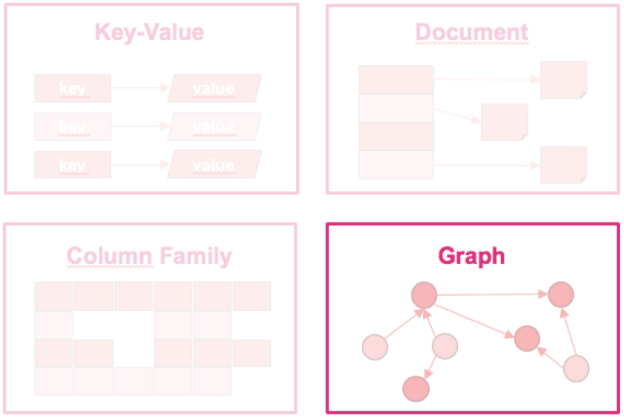


- **Examples:**
  - Cassandra,
  - BigTable
  - HBase

- **Strenghts:**
  - **schema flexibility** (documents can have different f ormats)
  - **fast inserts** and **point queries**
  - **Fast point queries/value lookups**
  - **easy to replicate** and **distribute** (e.g. *to achieve l oad balancing and failure tolerance*)

- **Weaknesses**:
  - **non-point queries** are **less efficient**/ **IO expensive** (e.g. range queries)
  - **no** or **less efficient** and slow:
    - aggregation
    - filtering
    - joining

Non-Relational Data Model – **Graph**



- **Examples:**
  - Neo4j,
  - ArangoDB,
  - OrientDB,
  - Titan,
  - Giraph, …

- **Strenghts:**
  - **many-to-many relationships** (unlike all other data models
  - **schema flexibility** (due to edges and property definition)
  - **efficient relationship queries**
  - usually support **ACID**

- **Weaknesses**:
  - not useful for transactional data or CRUD operations

Non-Relational Data Model – **MapReduce**

- **MapReduce** = **programming paradigm** and an associated **implementation** for **processing** and **generating large datasets in parallel** and **distributed** on a cluster
- originally introduced by Jeffrey Dean and Sanjay Ghemawat (Google Inc.) in 2004

- MapReduce is neither a declarative language nor a an imperative programming language → it's something in between

- The paradigm is based on specifying:
  - a **map function**, which performs filtering and sorting, resulting in an intermediate set of key/value pairs and
  - a **reduce function** that merges all intermediate values associated with the same key (e.g. sum all values).

- MapReduce Jobs are automatically parallized and executed on several nodes of a cluster

www.marcel-mittelstaedt.com

MapReduce

- The MapReduce **runtime system** (e.g. YARN in case of Hadoop) takes care of:
  - the details of providing and partitioning the input data,
  - scheduling the application code execution across all cluster nodes,
  - saving intermediate states,
  - handling node failures,
  - inter-node communication and much more.

MapReduce – **WordCount**

```
1  map(String key, String value):
2    // key: document name
3    // value: document content
4    for each word w in value:
5      EmitIntermediate(w, 1);
6
7  reduce(String key, Iterator values):
8    // key: a word
9    // values: a list of counts
10   int result = 0;
11   for each v in values:
12     result += v;
13   Emit(key, result);
```

The **map function** **takes** one **pair of data** with a type in one data dom ain, and **returns** a **list of pairs** in a different domain:

$$map(k1,v1) \rightarrow list(k2, v2)$$

This produces a list of pairs (keyed by k2) for each call.

After that, the MapReduce framework collects all pairs with the same key (*k2*) from all lists and groups them together, creating one group for each key.

The **reduce function** runs in parallel for each group, which in turn produces a collection of values (*v3*) to an associated key (*k2*) within the same domain:

$$reduce(k2, list(v2)) \rightarrow list(k2, v3)$$

The returns of all reduce processes are collected as the desired result list.

www.marcel-mittelstaedt.com

MapReduce – **WordCount**

```
1 document1 = "Da steh ich nun, ich armer Tor!";
2 document2 = "Und bin so klug als wie zuvor;";
3 document3 = "Heiße Magister, heiße Doktor gar";
4 document4 = "Und ziehe schon an die zehen Jahr";
5 document5 = "Herauf, herab und quer und krumm":
6 document6 = "Meine Schüler an der Nase herum.";
```

Code Snippet 2.2: MR Example - *Word Count (Input Documents)*

```
1 map(document1, "da steh ich nun ich armer tor");
2 map(document2, "und bin so klug als wie zuvor");
3 map(document3, "heiße magister heiße doktor gar");
4 map(document4, "und ziehe schon an die zehen jahr");
5 map(document5, "herauf, herab und quer und krumm");
6 map(document6, "meine schüler an der nase herum");
```

Code Snippet 2.3: MR Example - *Word Count (`map()` Calls)*

```
1 p1 = [ ("da",1), ("steh",1), ("ich",1), ("nun",1), ("ich",1),
2        ("armer",1), ("tor",1) ];
3 p2 = [ ("und",1), ("bin",1), ("so",1), ("klug",1), ("als",1),
4        ("wie",1), ("zuvor",1) ];
5 ...
6 p6 = [ ("meine",1), ("schüler",1), ("an",1), ("der",1), ("nase",1),
7        ("herum",1)];
```

Code Snippet 2.4: MR Example - *Word Count (Partial `map()` Results)*

MapReduce – **WordCount**

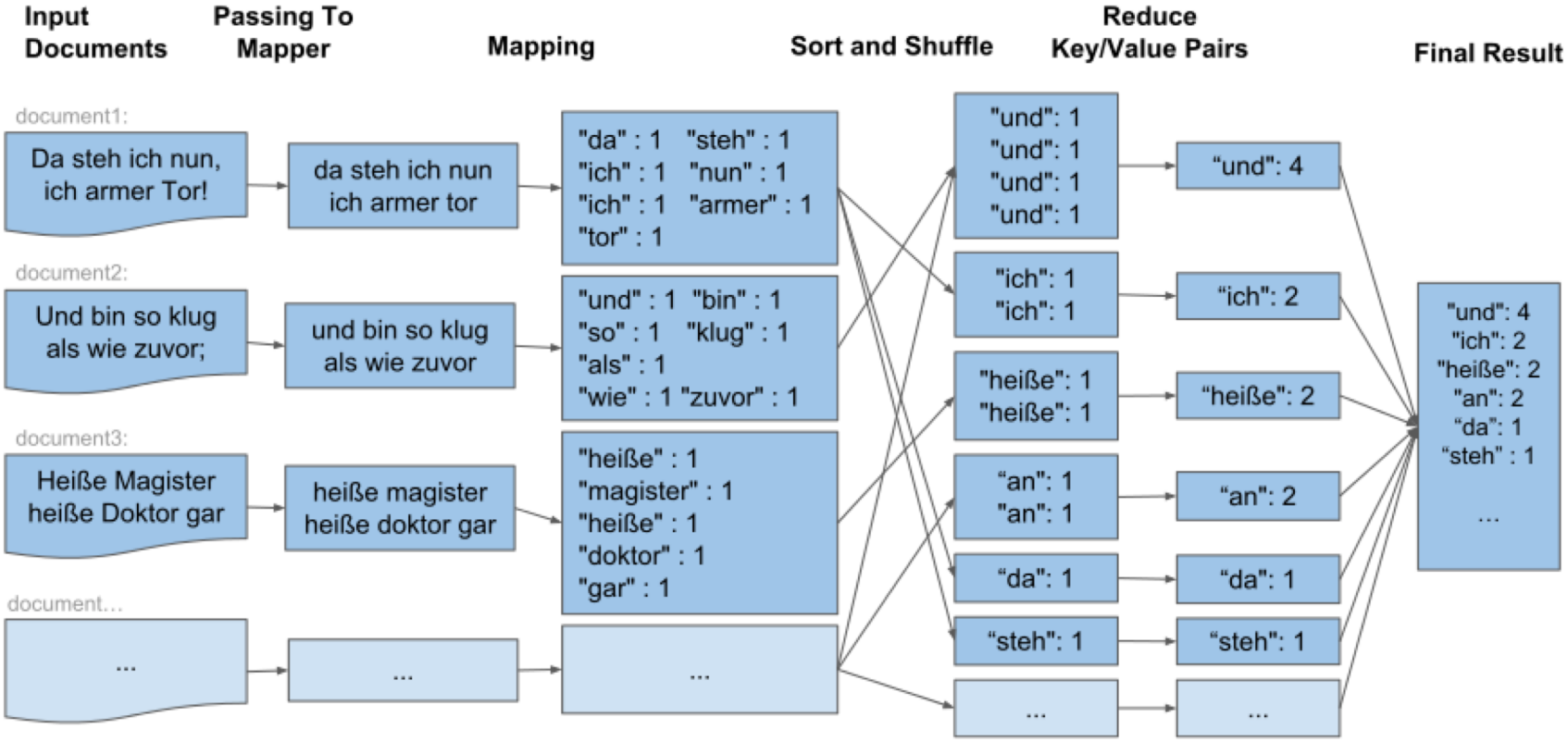
```
reduce("da", [1]); // = ("da", 1)
reduce("ich", [1,1]); // = ("ich", 2)
reduce("und", [1,1,1,1]); // = ("und", 4)
...
```

Code Snippet 2.5: MR Example - *Word Count (`reduce()` Calls)*

```
result = [ ("und", 4), ("ich",2), ("heiße",2), ("an", 2), ("da",1),
    ("steh",1), ("nun",1), ("armer",1), ("tor",1), ("bin",1), ("so",1),
    ("klug",1). ("als",1), ("wie",1), ("zuvor",1), ("magister",1),
    ("doktor",1), ("gar",1), ("ziehe",1), ("schon",1), ("die",1),
    ("zehen",1), ("jahr",1), ("herauf",1), ("herab",1), ("quer",1),
    ("krumm",1), ("meine",1), ("schüler",1), ("der",1), ("nase",1),
    ("herum",1) ]
```

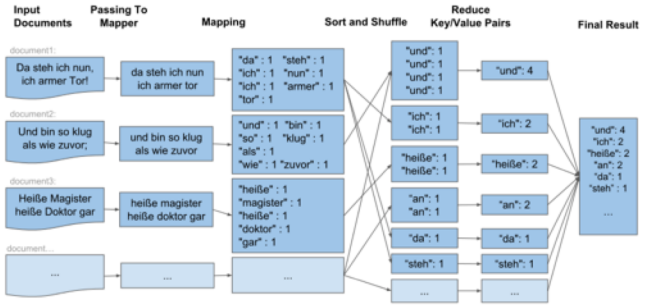
Code Snippet 2.6: MR Example - *Word Count (Final Result)*

MapReduce – **Phases**



www.marcel-mittelstaedt.com

MapReduce – Phases

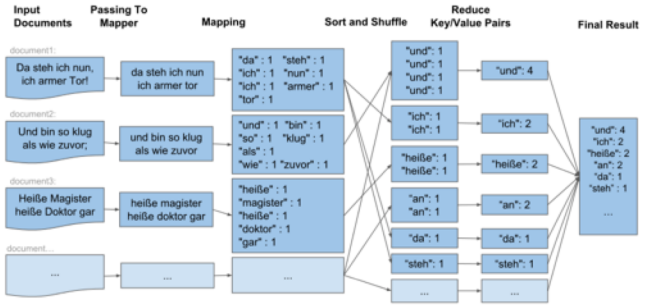


1. Map Phase: Each map function gets multiple key/value pairs and processes each of them separately. All Map processes run independently of each other, faciliating the whole processing to run concurrently on multiple nodes. In our example resulting in to several lists of words with an associated count of one and represented as key/values pairs.

2. Sort and Shuffle Phase: Phase in between Map and Reduce with the purpose of transferring data from map to reduce processes efficiently, which is usually done automatically by the MapReduce framework.
All output produced by the map processes is grouped and sorted by their key, partioned and distributed to the reduce processes.
A common way of partitioning is to hash the keys and use the hash value modulo amount of reduce processes to achieve an evenly distribution of load among all nodes of a cluster (the total number of partitions is equal to the number of reduce processes). Usually you do not need to make use of the frameworks default Shuffle, Sort and Partitioning, for instance using Hadoop you could implement your own ones, but as those parts are centerpieces in case of the whole MapReduce performance, think twice about not using the default ones.

www.marcel-mittelstaedt.com

MapReduce – Phases



3. Reduce Phase: The reduce function gets called once for each unique key. All Reduce processes run indepently of each other, faciliating the whole processing to run concurrently on multiple nodes. In this way the reduce functions iterates through all values associated to a key and produces zero or more output.
In case of the WordCount example an increment happens for each *value* (count) associated to the same *key* (word).

4. Output Phase: The output of all reduce processes are collected and consolidated by the MapRedcue framework and usually written to a distributed file system.

Beside the listed phases, most MapReduce frameworks also make use of a **phase** called **combiner**, which happens between Map phase and Sort&Shuffle phase. The combiner is also known as a "*mini-reducer*", it takes the intermediate output of the map processes - shuffles, sorts and reduces them partly by combining key/value pairs with the same key.
In case of our WordCount example (Figure 2.8 on page 52) the output of the first mapper wouldn't be:

```
[ ("da",1), ("steh",1), ("ich",1), ("nun",1), ("ich",1), ("armer"
   ,1), ("tor",1) ];
```

but

```
[ ("da",1), ("steh",1), ("ich",2), ("nun",1), ("armer",1), ("tor"
   ,1) ];
```


www.marcel-mittelstaedt.com

Break

TIME FOR A BREAK

HandsOn – **MapReduce**

Quick Introduction To Java and MapReduce

WordCount – Mapper

```
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {

  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }
```


www.marcel-mittelstaedt.com

WordCount – Reducer

```
  public static class IntSumReducer
       extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
                       Context context
                       ) throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }
      result.set(sum);
      context.write(key, result);
    }
  }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```


Run Java MapReduce Job

1. Compile Java file:

```
export HADOOP_CLASSPATH=$JAVA_HOME/lib/tools.jar
hadoop/bin/hadoop com.sun.tools.javac.Main WordCount.java
```

2. Create Jar File:

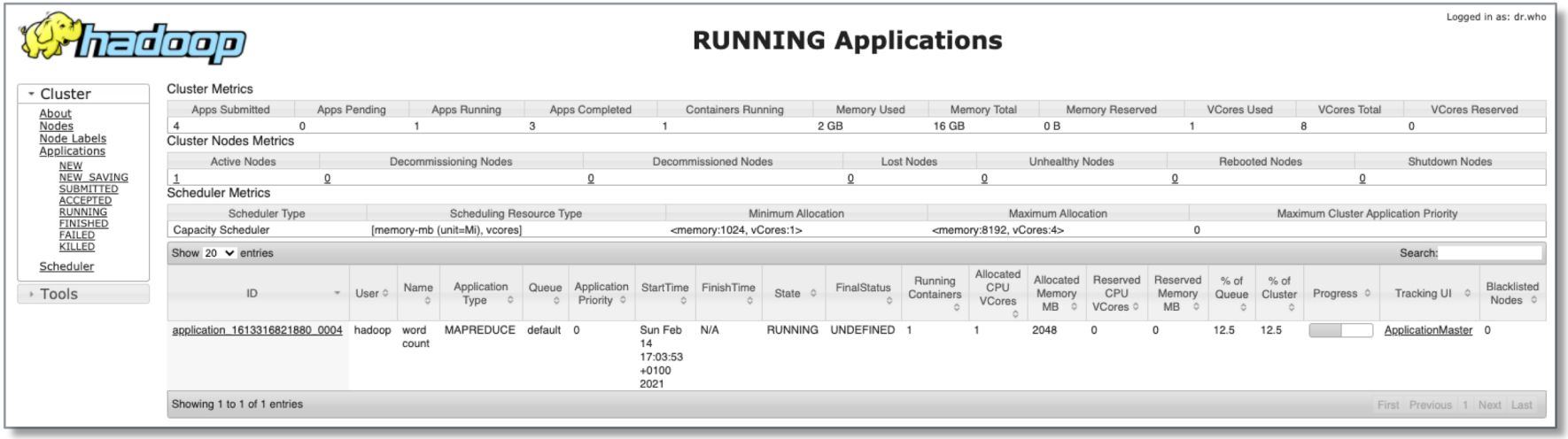
```
jar cf WordCount.jar WordCount*.class
```

3. Execute MapRedcue Job:

```
hadoop jar WordCount.jar WordCount /user/hadoop/Faust_1.txt
/user/hadoop/wordcount_output
```


Run Java MapReduce Job

4. Take a look at Ressource Manager for Job Execution (http://XXX.XXX.XXX.XXX:8088/cluster/apps/RUNNING):



Run Java MapReduce Job

4. Take a look at the results:

```
hadoop fs -cat /user/hadoop/wordcount_output/part-r-00000 | head -10

"Allein,  1
"Alles    1
"Als      1
"Der      1
"Die      2
"Er       2
"Ich      4
"Im       1
"Mein     1
"Nur      1
[…]
```

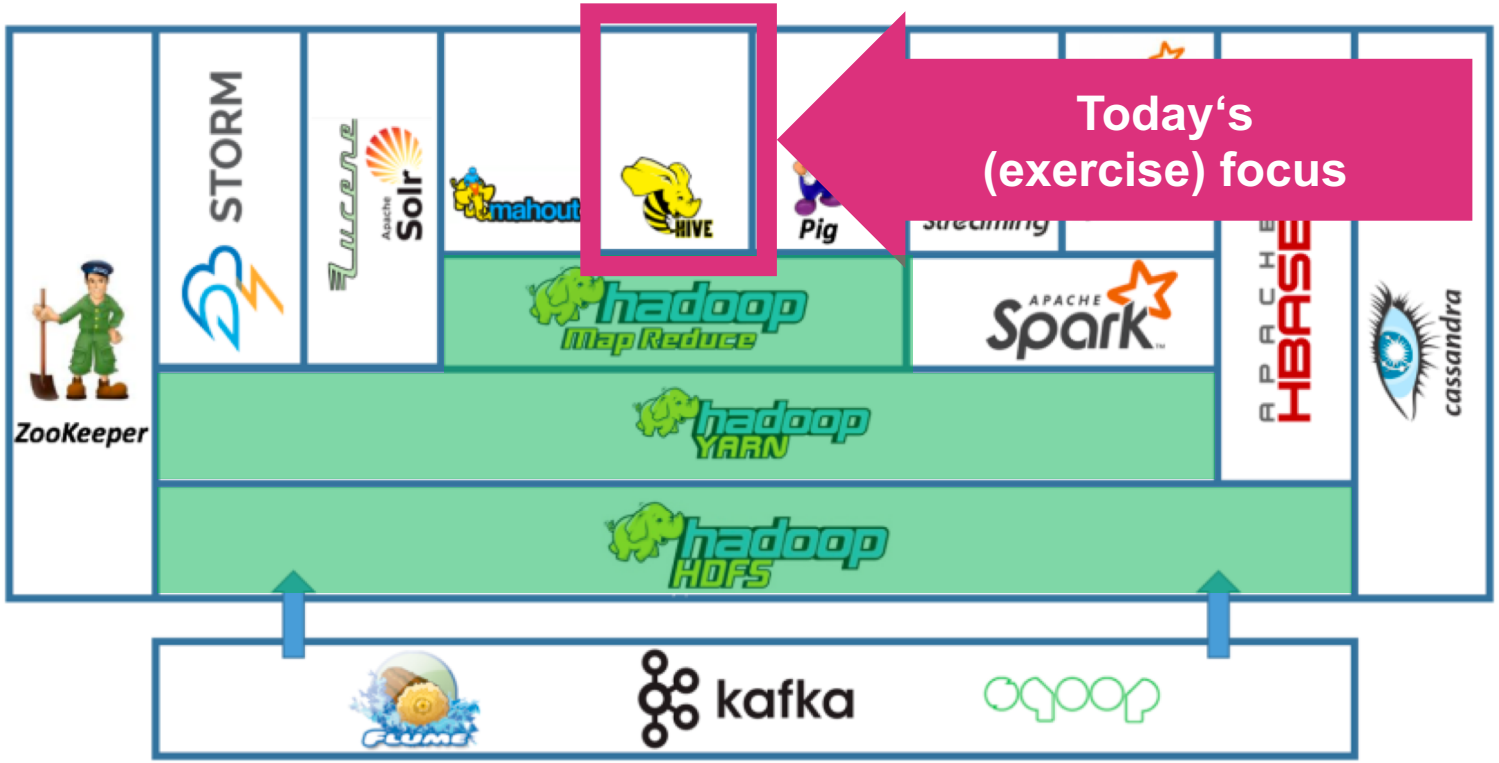
Break

TIME FOR A BREAK

HandsOn – **Hive and HiveQL**

Quick Introduction To Hive and HiveQL

The Hadoop Ecosystem



Apache Hive

Apache Hive

- Initial Release in October 2010
- written in Java
- current Version: 3.1.2

http://hive.apache.org

- **What is Hive:**
  - **easy to use** (HiveQL)
  - **based on Hadoop** (HDFS, YARN)
  - **good scale-out capabilities** (e.g. by partitioning and underlying HDFS and YARN)
  - **best used for:**
    - (BigData) **datawarehousing tasks**
    - a **DataLake**
    - **interface** for Analysts, DataScientists, Developers
    - **adhoc** (batch) **querying, aggregation** and **analysis** of large amounts of data (**PB!**) and hundreds of nodes
- **What Hive is NOT**:
  - **transactional database**
  - **highly responsive**

www.marcel-mittelstaedt.com

HiveQL

- SQL like **query language**
- Supported by: Hive CLI (*deprecated*), Beeline CLI and most JDBC clients
- Hive supported HDFS file formats:
  - **TextFile** (even compressed by gzip or bzip2)
  - **SequenceFile**
  - **RCFile**
  - **ORC**
  - **Parquet**
  - **Avro**

HDFS/Hive - **Wordcount**

- Previous WordCount example achieved by using Hive and HiveQL:

```
CREATE TABLE faust (line STRING);

LOAD DATA INPATH '/user/hadoop/faust' OVERWRITE INTO TABLE faust;

CREATE TABLE word_counts AS
SELECT word, count(1) AS count FROM
(SELECT explode(split(line, '\\s')) AS word FROM faust) temp
 GROUP BY word ORDER BY word;
```

```
SELECT * from word_counts ORDER BY count DESC LIMIT 10;
```

word	count
und	509
die	463
der	440
ich	435
Und	400
nicht	346
zu	319
[…]	

www.marcel-mittelstaedt.com

Break

TIME FOR A BREAK

Exercises Preparation

Install and Setup Hive

Create VM Instance

1. Delete previously created VM instance:

```
gcloud compute instances delete big-data
```

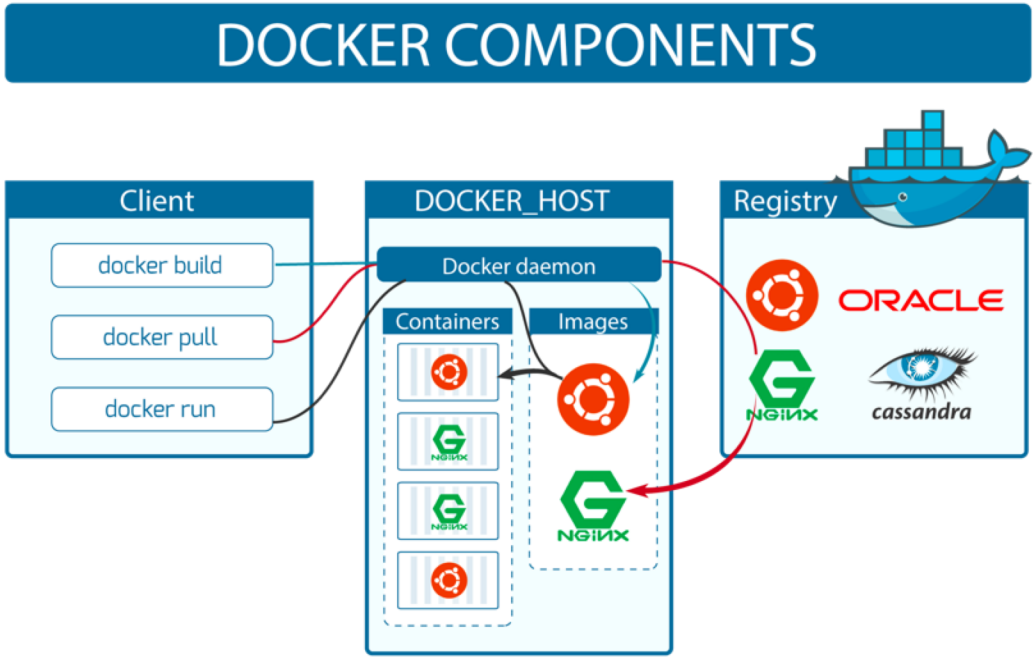
2. Create new instance:

```
gcloud compute --project=[your-project-id] instances create big-data \
            --zone=europe-west3-c \
            --machine-type=n1-highmem-4 \
            --subnet=default --network-tier=PREMIUM \
            --maintenance-policy=MIGRATE \
            --image=ubuntu-2004-focal-v20210129 \
            --image-project=ubuntu-os-cloud \
            --boot-disk-size=40GB \
            --boot-disk-type=pd-standard \
            --boot-disk-device-name=big-data
```

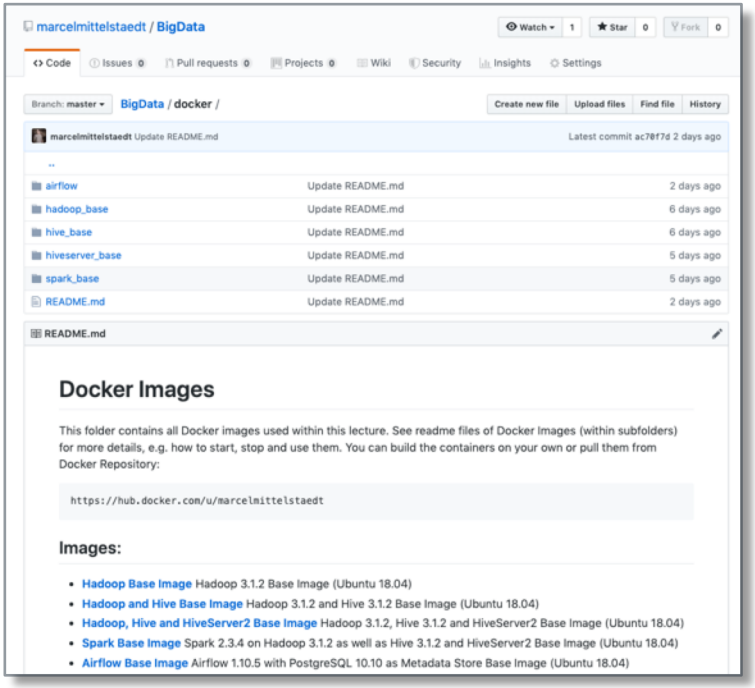
```
ssh hans.wurst@XXX.XXX.XXX.XXX
```


Docker

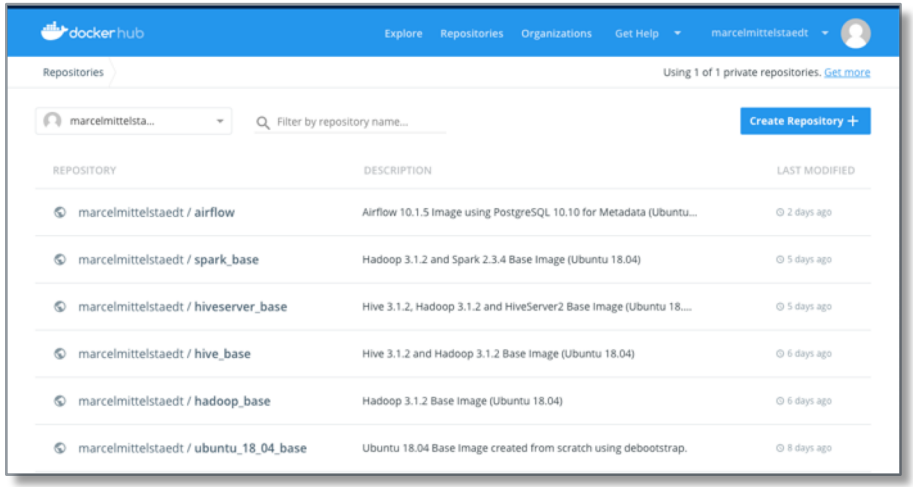
To speed things up and not waste time on installation and configuration of Hive and other tools, we will make use of docker container I‘ve already prepared.



Docker Images/Dockerfiles



https://github.com/marcelmittelstaedt/BigData/tree/master/docker



https://hub.docker.com/u/marcelmittelstaedt

Setup Docker Container

3. Install and setup docker

```
sudo apt-get update
sudo apt-get install docker.io
sudo usermod -aG docker $USER
# exit and login again
```

4. Pull Hadoop with Hive Image

```
docker pull marcelmittelstaedt/hive_base:latest
```

5. Start Container from pulled image:

```
docker run -dit --name hive_base_container -p 8088:8088 -p 9870:9870 -p 9864:
9864 marcelmittelstaedt/hive_base:latest
```


Setup Docker Container

6. Show Running Container:

```
docker ps -a
```

7. Show Logs of container (wait till finished):

```
docker logs hive_base_container

[…]
Stopping namenodes on [localhost]
Stopping datanodes
Stopping secondary namenodes [c821a0e1bdcf]
Stopping nodemanagers
Stopping resourcemanager
Container Startup finished.
```

Setup Docker Container

8. Get a shell inside the container:

```
hans.wurst@big-data:~$ docker exec -it hive_base_container bash
root@c821a0e1bdcf:/#
```

9. Switch to hadoop user:

```
root@c821a0e1bdcf:/# sudo su hadoop
hadoop@c821a0e1bdcf:/$ cd
hadoop@c821a0e1bdcf:~$
```

10. Start DFS and YARN:

```
start-all.sh
```


Test Hive

11. Test if Hive Installation and Configuration is successful. Start Hive:

```
hive
```

```
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/hadoop/hive/lib/log4j-slf4j-impl-2.10.0.jar!/org/slf4j/
impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/hadoop/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7
.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
Hive Session ID = c120d0b1-9025-43db-96e4-48ccfb875f1a

Logging initialized using configuration in jar:file:/home/hadoop/hive/lib/hive-common-3.1.0.jar
!/hive-log4j2.properties Async: true
Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider us
ing a different execution engine (i.e. spark, tez) or using Hive 1.X releases.
Hive Session ID = fdd6f06a-d4e4-48e6-8971-997e8a0a8e2c
hive>
```


www.marcel-mittelstaedt.com

Install and Setup Hive

12. Execute First SQL Query:

```
hive> show databases;
OK
default
Time taken: 0.083 seconds, Fetched: 1 row(s)
hive>
```


Break

TIME FOR A BREAK

Hive: Create and Work with External Tables

Using public dataset of IMDb.com

Get IMDb Data And Move It To HDFS

1. Get **IMDb Data** (*https://www.imdb.com/interfaces/*):

```
wget https://datasets.imdbws.com/title.basics.tsv.gz
wget https://datasets.imdbws.com/title.ratings.tsv.gz
```

2. Uncompress IMDb Data:

```
gunzip title.basics.tsv.gz
gunzip title.ratings.tsv.gz
```

3. Create HDFS Directories for IMDb Data:

```
hadoop fs -mkdir /user/hadoop/imdb
hadoop fs -mkdir /user/hadoop/imdb/title_basics
hadoop fs -mkdir /user/hadoop/imdb/title_ratings
```

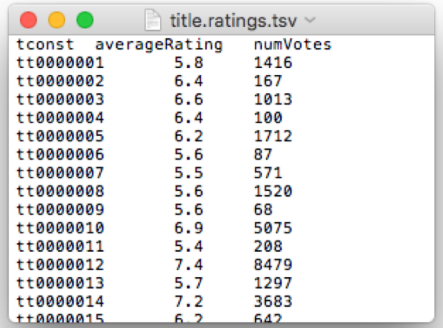

Create External Tables In Hive

4. Transfer IMDb data files to HDFS:

```
hadoop fs -put title.basics.tsv /user/hadoop/imdb/title_basics/title.basics.tsv
hadoop fs -put title.ratings.tsv /user/hadoop/imdb/title_ratings/title.ratings.tsv
```

5. Create External Table `title_ratings` (file *title.ratings.tsv*) in Hive:

```
hive > CREATE EXTERNAL TABLE IF NOT EXISTS title_ratings(
          tconst STRING,
          average_rating DECIMAL(2,1),
          num_votes BIGINT
) COMMENT 'IMDb Ratings'
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' STORED AS
TEXTFILE LOCATION '/user/hadoop/imdb/title_ratings'
TBLPROPERTIES ('skip.header.line.count'='1');
```



Create External Tables In Hive

6. Create External Table `title_basics` for file *title.basics.tsv* in Hive:

title.basics.tsv

tconst	titleType	primaryTitle	originalTitle	isAdult	startYear	endYear	runtimeMinutes	genres
tt0000001	short	Carmencita	Carmencita	0	1894	\N	1	Documentary,Short
tt0000002	short	Le clown et ses chiens	Le clown et ses chiens	0	1892	\N	5	Animation,Short
tt0000003	short	Pauvre Pierrot	Pauvre Pierrot	0	1892	\N	4	Animation,Comedy,Romance
tt0000004	short	Un bon bock	Un bon bock	0	1892	\N	\N	Animation,Short
tt0000005	short	Blacksmith Scene	Blacksmith Scene	0	1893	\N	1	Comedy,Short
tt0000006	short	Chinese Opium Den	Chinese Opium Den	0	1894	\N	1	Short
tt0000007	short	Corbett and Courtney Before the Kinetograph	Corbett and Courtney Before the Kinetograph	0	1894	\N	1	Short,Sport
tt0000008	short	Edison Kinetoscopic Record of a Sneeze	Edison Kinetoscopic Record of a Sneeze	0	1894	\N	1	Documentary,Short
tt0000009	movie	Miss Jerry	Miss Jerry	0	1894	\N	45	Romance
tt0000010	short	Employees Leaving the Lumière Factory	La sortie de l'usine Lumière à Lyon	0	1895	\N	1	Documentary,Short
tt0000011	short	Akrobatisches Potpourri	Akrobatisches Potpourri	0	1895	\N	1	Documentary,Short
tt0000012	short	The Arrival of a Train	L'arrivée d'un train à La Ciotat	0	1896	\N	1	Documentary,Short
tt0000013	short	The Photographical Congress Arrives in Lyon	Neuville-sur-Saône: Débarquement du congrès des photographes à Lyon	0	1895	\N	1	Documentary,Short
tt0000014	short	Tables Turned on the Gardener	L'arroseur arrosé	0	1895	\N	1	Comedy,Short
tt0000015	short	Autour d'une cabine	Autour d'une cabine	0	1894	\N	2	Animation,Short
tt0000016	short	Barque sortant du port	Barque sortant du port	0	1895	\N	1	Documentary,Short
tt0000017	short	Italienischer Bauerntanz	Italienischer Bauerntanz	0	1895	\N	1	Documentary,Short
tt0000018	short	Das boxende Känguruh	Das boxende Känguruh	0	1895	\N	1	Short
tt0000019	short	The Clown Barber	The Clown Barber	0	1898	\N	\N	Comedy,Short
tt0000020	short	The Derby 1895	The Derby 1895	0	1895	\N	1	Documentary,Short,Sport
tt0000022	short	Blacksmith Scene	Les forgerons	0	1895	\N	1	Documentary,Short
tt0000023	short	The Sea Baignade en mer	0	1895	\N	1	Documentary,Short	
tt0000024	short	Opening of the Kiel Canal	Opening of the Kiel Canal	0	1895	\N	\N	News,Short
tt0000025	short	The Oxford and Cambridge University Boat Race	The Oxford and Cambridge University Boat Race	0	1895	\N	\N	News,Short,Sport
tt0000026	short	The Messers. Lumière at Cards	Partie d'écarté	0	1896	\N	1	Documentary,Short
tt0000027	short	Cordeliers' Square in Lyon	Place des Cordeliers à Lyon	0	1895	\N	1	Documentary,Short
tt0000028	short	Fishing for Goldfish	La pêche aux poissons rouges	0	1895	\N	1	Documentary,Short
tt0000029	short	Baby's Dinner	Repas de bébé	0	1895	\N	1	Documentary,Short
tt0000030	short	Rough Sea at Dover	Rough Sea at Dover	0	1895	\N	1	Documentary,Short

Create External Tables In Hive

6. Create External Table `title_basics` for file *title.basics.tsv* in Hive:

```
hive > CREATE EXTERNAL TABLE IF NOT EXISTS title_basics (
    tconst STRING,
    title_type STRING,
    primary_title STRING,
    original_title STRING,
    is_adult DECIMAL(1,0),
    start_year DECIMAL(4,0),
    end_year STRING,
    runtime_minutes INT,
    genres STRING
) COMMENT 'IMDb Movies' ROW FORMAT DELIMITED FIELDS TERMINATED BY
'\t' STORED AS TEXTFILE LOCATION '/user/hadoop/imdb/title_basics'
TBLPROPERTIES ('skip.header.line.count'='1');
```

Create External Tables In Hive

7. Query Table `title_basics` in Hive using SQL (HiveQL):

```
hive> select * from title_basics limit 3;
OK
tt0000001 short Carmencita Carmencita 0 1894 NULL 1 Documentary,Short
tt0000002 short Le clown et ses chiens Le clown et ses chiens 0 1892 NULL 5 Animation,Short
tt0000003 short Pauvre Pierrot Pauvre Pierrot 0 1892 NULL 4 Animation,Comedy,Romance
Time taken: 0.139 seconds, Fetched: 5 row(s)
hive>
```

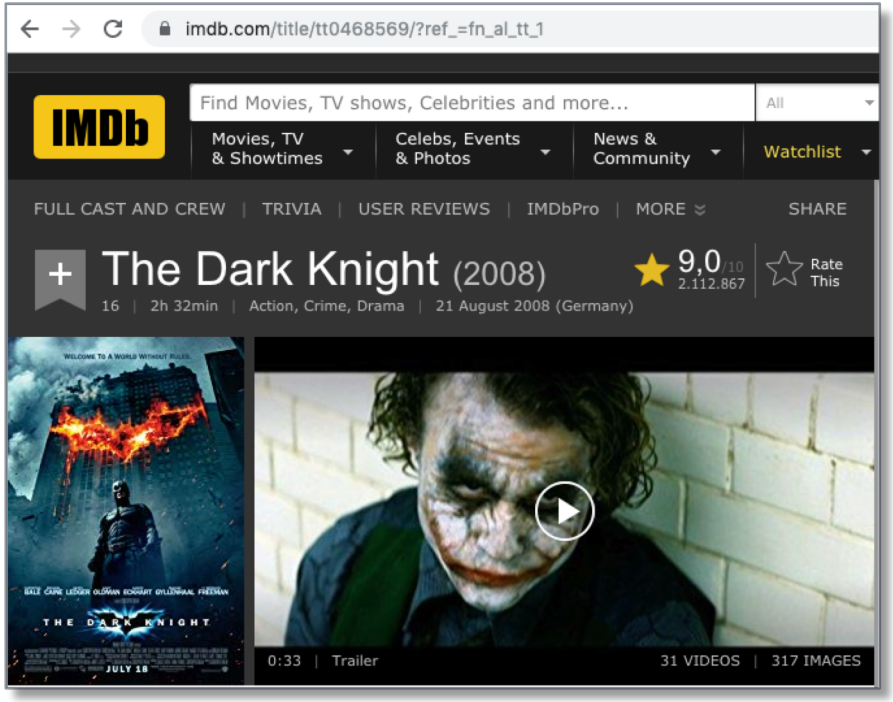
8. Query Table `title_ratings` in Hive using SQL (HiveQL):

```
hive> select * from title_ratings limit 3;
OK
tt0000001 5.6 1540
tt0000002 6.1 186
tt0000003 6.5 1199
Time taken: 0.119 seconds, Fetched: 3 row(s)
hive>
```


Create External Tables In Hive

9. Run a complex query which starts a MapReduce Job on Yarn, e.g. get Rating of movie „*The Dark Knight*“:

```
SELECT
   *
FROM
   title_basics b
   JOIN title_ratings r ON (b.tconst=r.tconst)
WHERE
   original_title = 'The Dark Knight'
   AND title_type='movie';
```



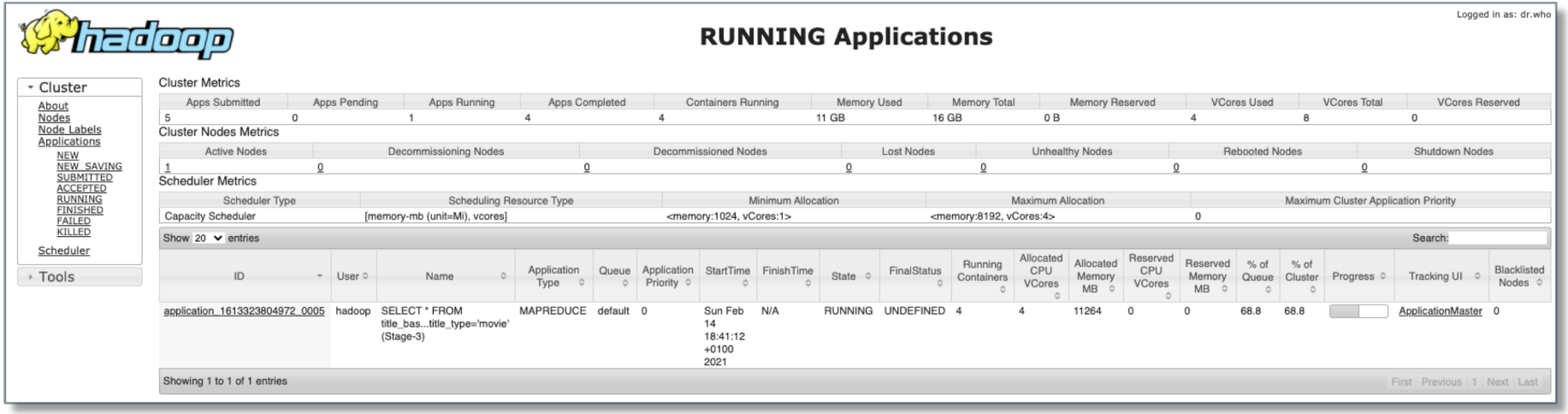
Create External Tables In Hive

9. Execute Query

```
hive> SELECT * FROM title_basics b JOIN title_ratings r ON (b.tconst=r.tconst) WHERE original_title =
'The Dark Knight' and title_type='movie';
[…]
Starting Job = job_1613323804972_0003, Tracking URL = http://154172c92bc7:8088/proxy/application_1613323804972_0003/
Kill Command = /home/hadoop/hadoop/bin/mapred job  -kill job_1613323804972_0003
Hadoop job information for Stage-3: number of mappers: 3; number of reducers: 0
2021-02-14 17:37:59,326 Stage-3 map = 0%,  reduce = 0%
2021-02-14 17:38:20,617 Stage-3 map = 50%,  reduce = 0%, Cumulative CPU 35.59 sec
2021-02-14 17:38:21,656 Stage-3 map = 67%,  reduce = 0%, Cumulative CPU 52.65 sec
2021-02-14 17:38:22,718 Stage-3 map = 100%,  reduce = 0%, Cumulative CPU 56.84 sec
MapReduce Total cumulative CPU time: 56 seconds 840 msec
Ended Job = job_1613323804972_0003
MapReduce Jobs Launched:
Stage-Stage-3: Map: 3   Cumulative CPU: 56.84 sec   HDFS Read: 650217476 HDFS Write: 376 SUCCESS
Total MapReduce CPU Time Spent: 56 seconds 840 msec
OK
tt0468569 movie The Dark Knight The Dark Knight 0 2008 NULL 152 Action,Crime,Drama tt0468569 9.0 2111245
Time taken: 53.094 seconds, Fetched: 1 row(s)
hive>
```

Create External Tables In Hive

9. Take a look at YARN (http://XXX.XXX.XXX.XXX:8088/cluster/):



Break

TIME FOR A BREAK

Exercises

Hive: Create and Work with External Tables

HDFS and HiveQL Exercises - **IMDB**

1. Execute Tasks of previous HandsOn Slides
2. Download https://datasets.imdbws.com/name.basics.tsv.gz
3. Create HDFS Directory **/user/hadoop/imdb/name_basics**/ for file name.basics.tsv
4. Create External Hive Table `name_basics` for name.basics.tsv
5. Use HiveQL to answer following questions:
   a) How many **movies** and how many **TV series** are within the IMDB dataset?
   b) Who is the **youngest** actor/writer/… within the dataset?
   c) Create a list (`tconst, original_title, start_year, average_rating, num_votes`) of movies which are:
   - equal or newer than year 2010
   - have an average rating equal or better than 8,1
   - have been voted more than 100.000 times

   d) How many movies are in list of c)?

HDFS and HiveQL Exercises - **IMDB**

5. Use HiveQL to answer following questions:

e) We want to know which years have been great for cinema.
Create a list with one row per year and a related count of movies which:
- have an average rating better than 8
- have been voted more than 100.000 times

ordered descending by count of movies.

Well Done

WE'RE DONE FOR ...TODAY

Stop Your VM Instances

DON'T FORGET TO STOP YOUR VM INSTANCE!

```
gcloud compute instances stop big-data
```