Jannic Alexander Cutura*

Why is my computer so slow? How distributed computing can help you with data intensive workloads**

A non-computer scientists perspective

* Software Engineer @ ECB DGIS

20/12/2022

*** The views expressed in this presentation are those of the authors and do not represent the views of the ECB or the ESCB*

Agenda for today

- **Topic:** How to tackle challenges when working with big data sets?
- **Target audience:**
  - Analysts/economists with no background in computer science/engineering
  - "*Let me load all data from network drive/Oracle into memory to work with it in Stata/Matlab/R/Python…*"– Is this you?
- **Learning Objectives:**
  - Understand why your computer gets slow when working with big data
  - What is a Distributed System (Hadoop/Spark)?
  - How can I use those to speed up data handling/analysis?

Nice meeting you all!
My name is Jannic and I am an…

- **Economist (by training)…**
  - B.Sc. Economics @ Uni Freiburg
  - M.Sc. and Ph.D. from Goethe University Frankfurt
  - Research stays at: BIS, IMF, Columbia
- **… turned data engineer…**
  - PhD trainee/Analyst @ECB financial stability & monetary policy divisions
  - Data engineering topics: "I need some help putting 300GB+ AnaCredit data into an excel plot"
- **… turned software engineer**
  - Software engineer in Stress Testing
- **Connect with me on [LinkedIn]** ☺

Big data in economics

Journal of Data Science

Search within journa   Submit your article   Information

Article info   Related articles

Econometrics at Scale: Spark up Big Data in Economics☆

Volume 20, Issue 3 (2022): Special Issue: Data Science Meets Social Sciences, pp. 413–436

Benjamin Bluhm   Jannic Alexander Cutura

https://doi.org/10.6339/22-JDS1035

Pub. online: 7 April 2022   **Type:** Data Science Reviews   Open Access

☆ The views expressed in this paper are those of the authors alone and do not represent the view of the European Central Bank (ECB).

Received	**Accepted**	**Published**
9 November 2021	12 January 2022	7 April 2022

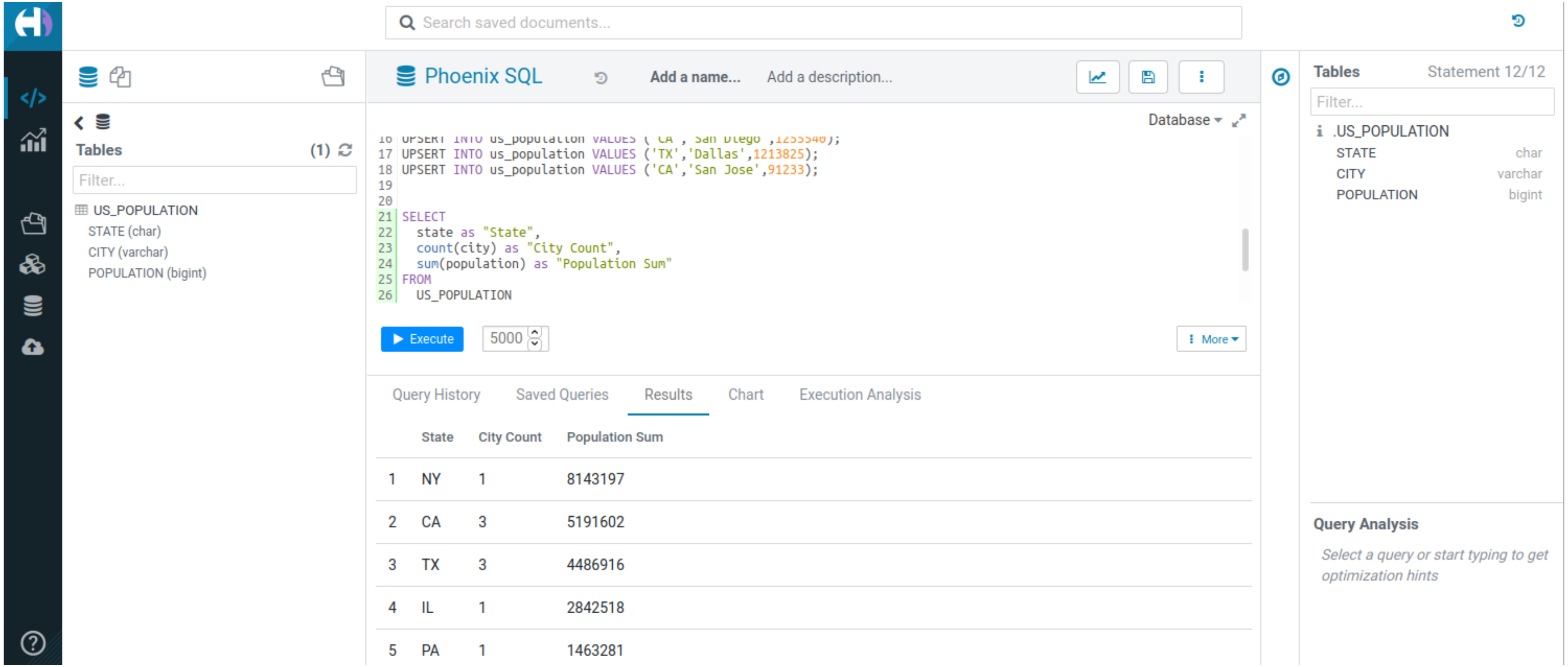
How to work with (big) data

- Your laptop/desktop
- Remote desktop (Citrix)
- SQL databases/datawarehouses (Oracle Exadata, Microsoft SQL Server,…)
- Hadoop/Spark
  - On prem: e.g., Cloudera Distributed Hadoop (CDH),…
  - On cloud: e.g., Cloudera Data Platform (CDP), AWS EMR, Databricks,…

How to work with (big) data

- Your laptop/desktop
- Remote desktop (Citrix)
- SQL databases/datawarehouses (Oracle Exadata, Microsoft SQL Server,…)
- **Hadoop/Spark**
    - On prem: e.g., Cloudera Distributed Hadoop (CDH),…
    - On cloud: e.g., Cloudera Data Platform (CDP), AWS EMR, Databricks,…

Hadoop User Experience (HUE)



Source: https://github.com/cloudera/hue

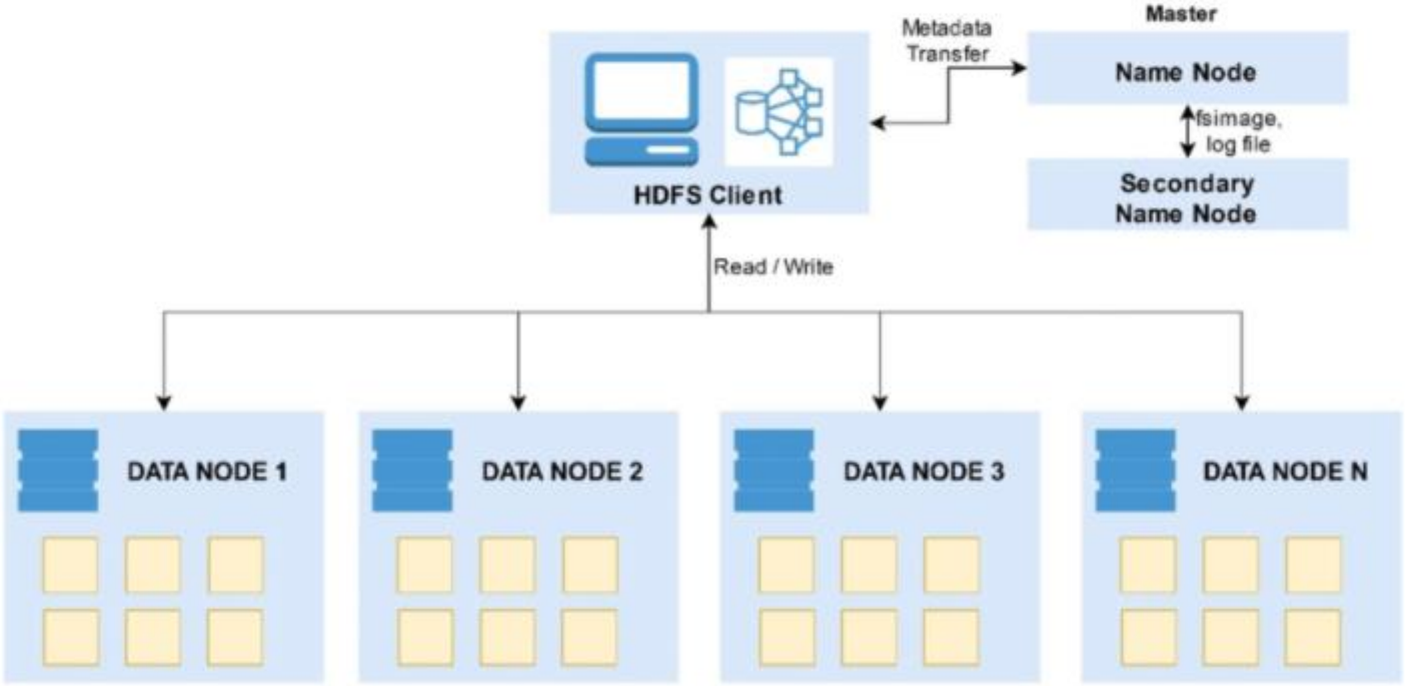
JDBC/ODBC connection (e.g. Stata)

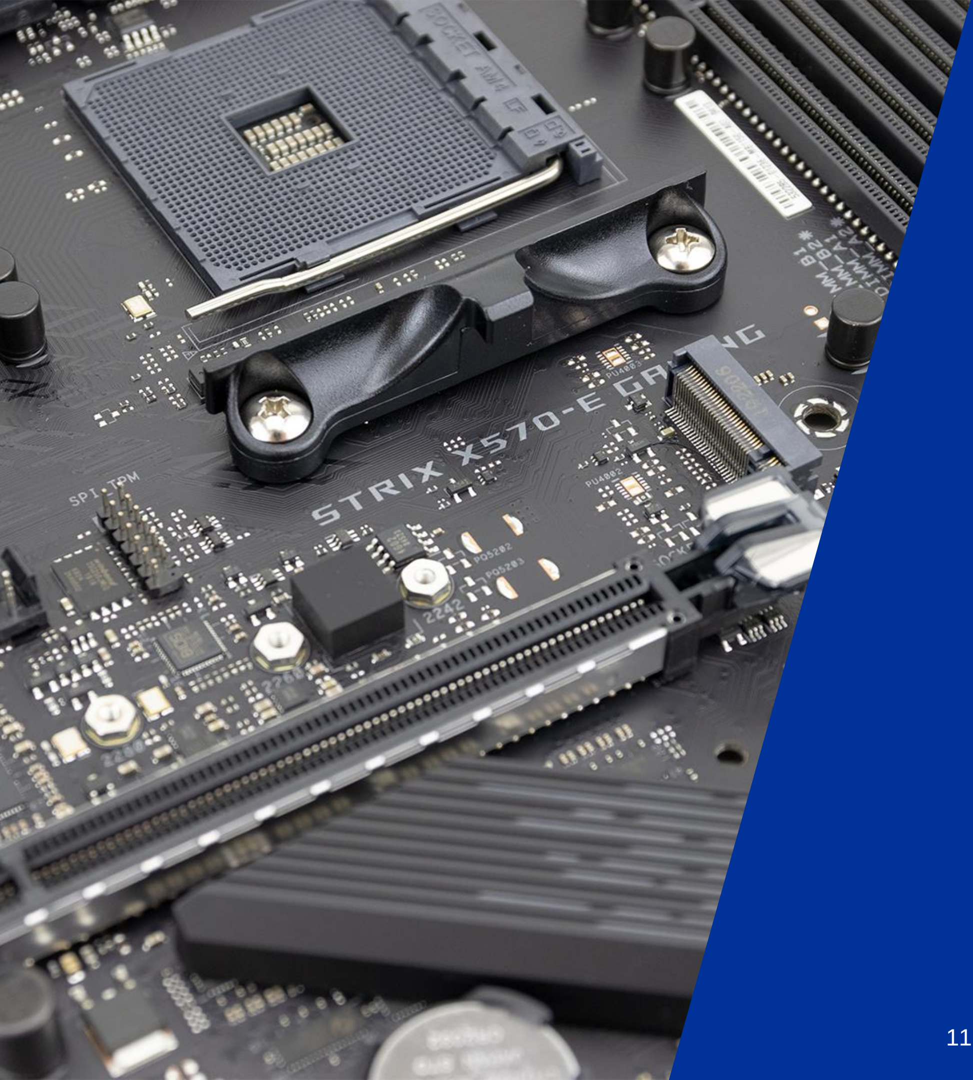
```
#delimit;
odbc load, exec("
SELECT *
FROM my_database.my_table
")
dsn("DISC DP Hive 64bit")
user("")
password("");
#delimit cr
```

What’s a Hadoop cluster?

- Many providers, both on prem and in cloud
- For example: CDH An on-premise **Hadoop Cluster** installed by Cloudera running Cloudera Distributed Hadoop (CDH)
  - **On-premise:** You physically own the hardware
  - **Hadoop Cluster:** A set of connected servers running the Hadoop Distributed File System HDFS
  - **Cloudera:** A company making loads of money selling big data analytics solutions
  - **Cloudera Distributed Hadoop:** The full Hadoop Ecosystem configured by Cloudera

Hadoop Cluster: A set of connected servers running the Hadoop Distributed File System HDFS



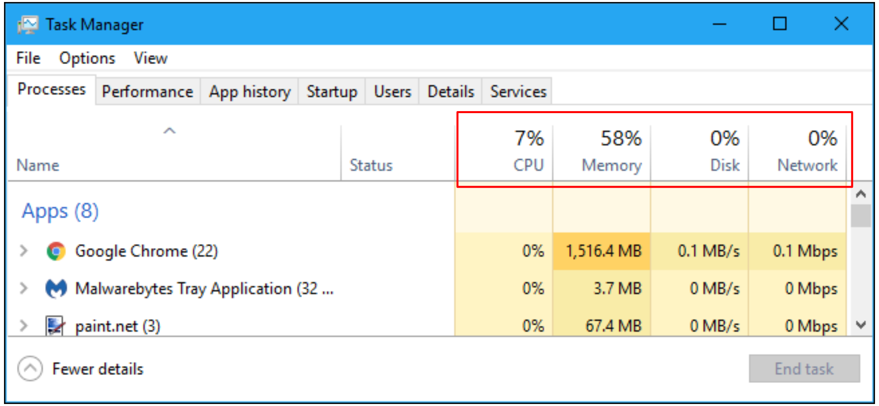


Distributed Computing

A primer on computer architecture

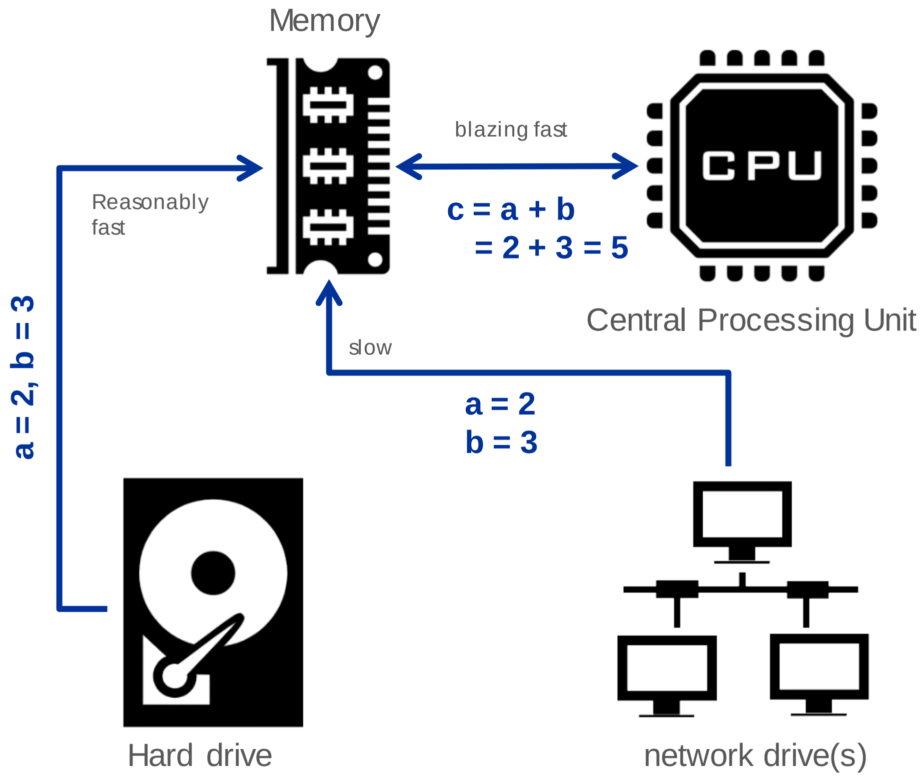
Understanding bottlenecks

- **Three bottlenecks to understand:**
  - CPU
  - RAM
  - I/O (disk & network)
- **Let's understand how those work together**



Computer Architecture

- **Problem: Sum two numbers that are stored on hard drive / network drive**
- **Input/Output IO**
  - **Hard drive**
    - Large files can take time to read
    - Less of a problem nowadays thanks to SSDs
  - **Network**
    - Files need to traffic through the network
    - Even worse from home office (have you noticed?)
- **Random Access Memory (RAM)**
  - Small (~20GB on your thinkpad)
  - „Space complexity problem“
- **Central Processing Unit (CPU)**
  - Some tasks are computationally intensive
  - „Time complexity problem“



Space complexity example

- 100,000 x 100,000 of zeros
- Add 1 to each cell. What‘s the result?
- 100,000 x 100,000 of ones
- Let python do it:

```
import numpy as np
zeros = np.zeros((100_000,100_000))
>> MemoryError: Unable to allocate 74.5 GiB for an array with shape (100000, 100000) and data type float64
```

- We would need 75GB available memory to populate the matrix
  - Note that we haven‘t computed anything (no „+1“ yet); just laying out the matrix

Time complexity example

- Consider the fibonacci sequence:
  - $F(n) = F(n-1) + F(n-2)$
  - $0, 1, 1, 2, 3, 5, 8, 13, 21, ?$
- Space complexity?
  - *Step 1*: 0, 1
  - *Step 2*: 1, 1
  - *Step 3*: 1, 2
  - *Step 4*: 2, 3
  - … need only two numbers in memory at any point in time
  - Constant space complexity: $\mathcal{O}(1)$
- Time complexity?
  - Need n computations for get the n-th Fibonacci number
  - Linear time complexity: $\mathcal{O}(n)$ -- you cannot „split the work up“ (i.e. you cannot parallelize)
  - Check the appendix for more details on space/runtime complexity of Fibonacci

When can distributed computing help you

- **You have a space complexity problem**
  - The data you want to work with does not fit in memory
  - Rule of thumb: Big data = 25-50GB or more
- **You have a time complexity problem that can be <u>parallelized</u>**
  - i.e. a long computation that can be split individual parts which can be evaluated in parallel
  - E.g. 1+2+3+4+5+6+7+8+9+10

$$\underbrace{1+2+3+4+5}_{15}+\underbrace{6+7+8+9+10}_{40}$$

- **You lose a lot of time reading large datasets from the network drive to your laptop**
  - Think: Download a large dataset, do an aggregation vs do an aggregation on the server download only the result

When can distributed computing help you: Caveats

- **You must have access to a Hadoop/Spark Cluster**
  - On prem, can be quite expensive
  - Game Changer: Public Clouds! You can spin up your own Cluster in minutes and handle terabytes of data at reasonable costs
- **You must be able express your data manipulation in (Impala/Hive) SQL or Spark**
  - Many, but not all standard features of data manipulation are implemented in Impala/Hive SQL (but no regressions…)
  - Spark offers a wide range of analytical functionality (regressions) and even more data manipulation commands
- **If not, you can likely still get some benefits from interacting through Python/R/Stata with your distributed system**
  - Distributed systems under the hood, have a file system “just like” your network drive. You can read and write files there (any file, not just the tables you see in Hue!)

Example: Average of a huge dataset

- **Too large dataset: You want the average of all numbers.**
- **Old, bad way:**

  Transfer data from network drive/database into your computer

```
// from p drive
use P:\ECB\...\...\my_big_file.dta
// from SQL connection
odbc, exec("SELECT * FROM my_database.my_table")
```

1. **I/O:** Takes long to "download" data from network drive/sql into memory
2. **Space complexity**: Difficulty fitting it into memory
3. **Time complexity**: Takes along time to compute the mean of many numbers

```
collapse (mean) myvar
```

- **New, good way:**
  - In HUE:

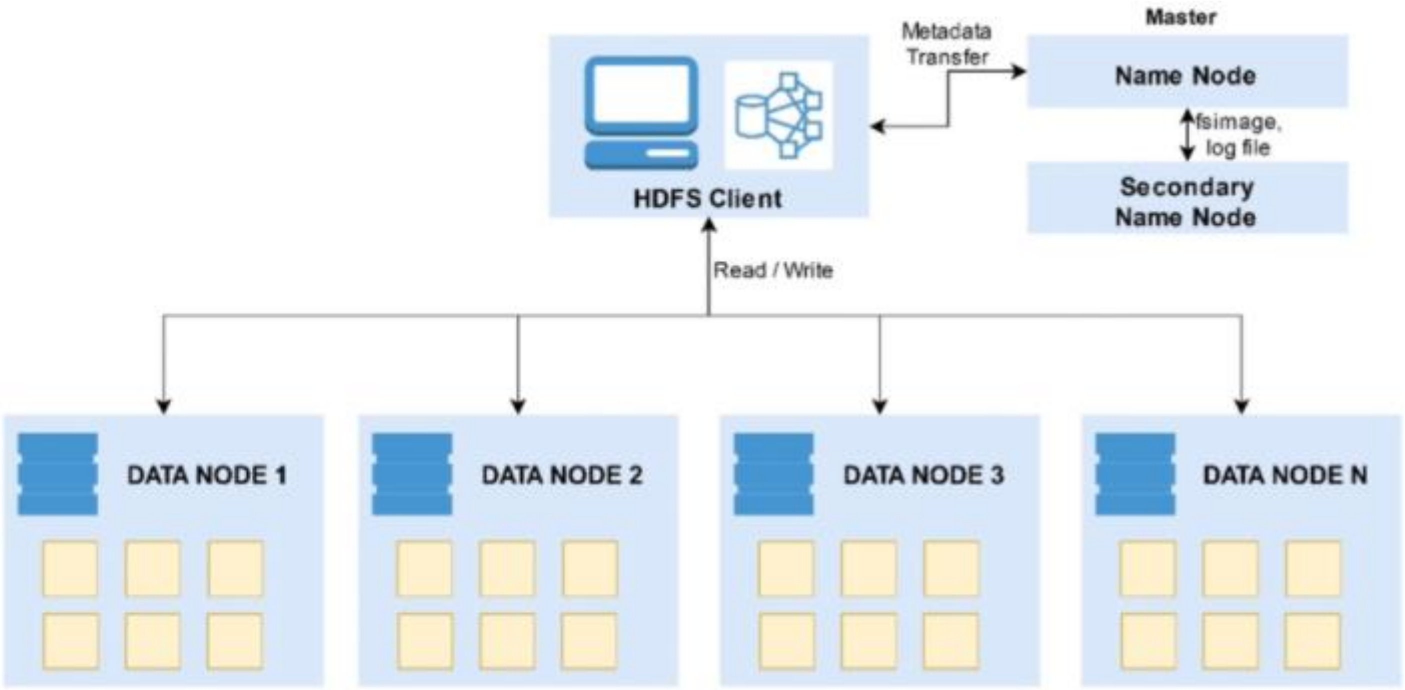
```
SELECT avg(myvar) FROM my_database.my_table
```

  - Or into Stata/Python/R/…

```
odbc, exec("SELECT avg(myvar) FROM my_database.my_table")
```

Example: Average of a huge dataset

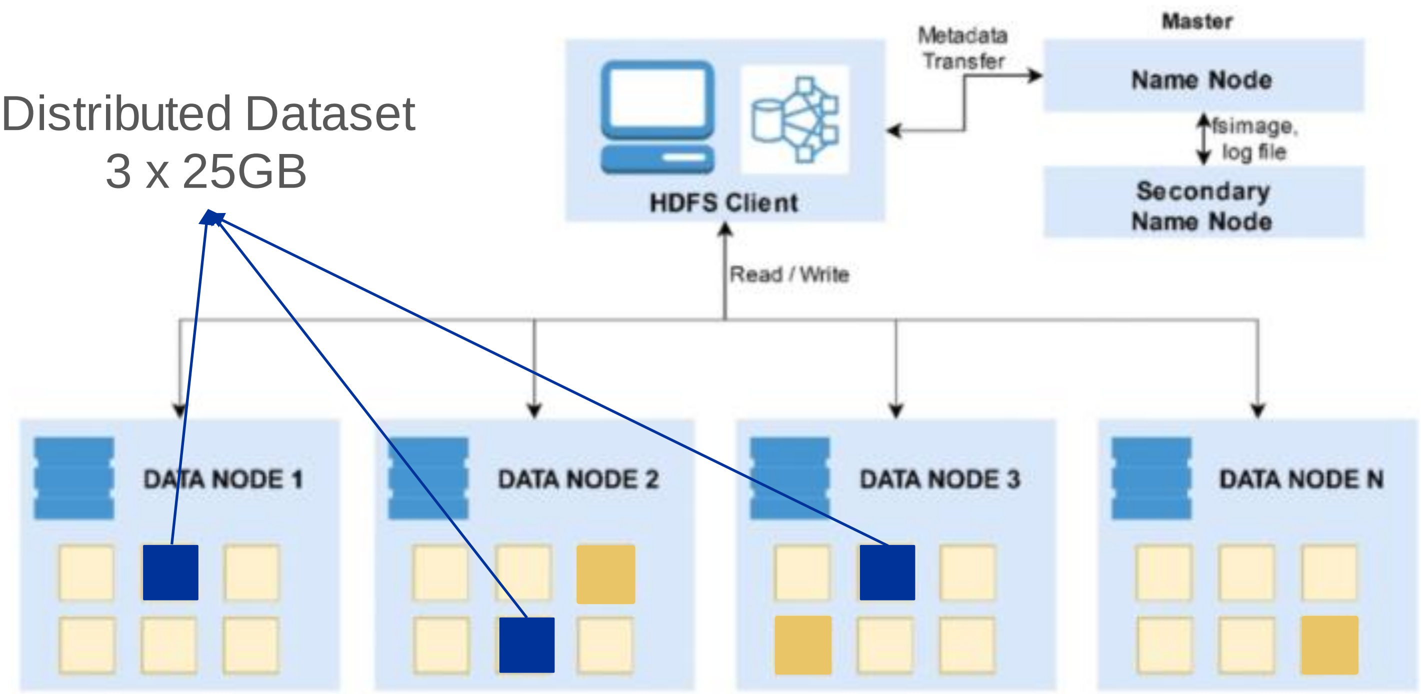
- **75GB dataset, you want the average**



Example: Average of a huge dataset

- **75GB dataset, you want the average**

Distributed Dataset
3 x 25GB



Example: Average of a huge dataset

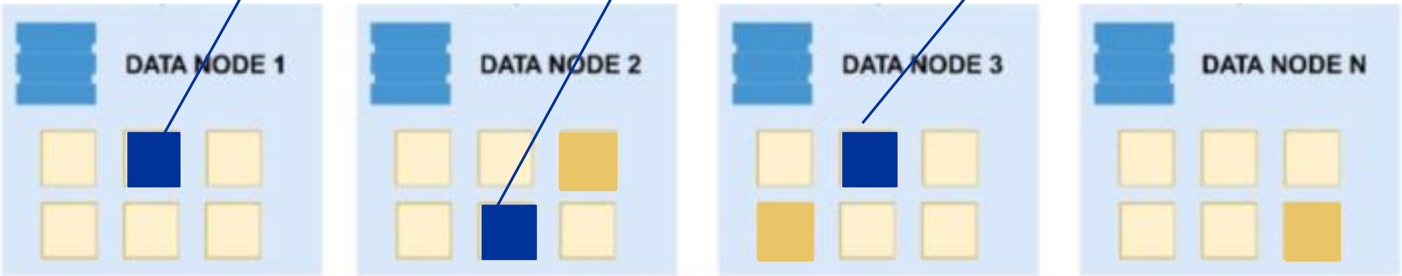
- **75GB dataset, you want the average**
- **Divide and conquer! Split the big data set into several small ones**

Example: Average of a huge dataset

- **75GB dataset, you want the average**
- **Compute individual means and average those**

m=(m1+m2+m3)/3

m1= mean(df1) m2= mean(df2) m3= mean(df3)



Example: Average of a huge dataset

- **Too large dataset: You want the average of all numbers.**
- **Non-DISC way:**

Transfer data from DISC/P drive into memory

```
// from p drive
use P:\ECB\...\...\my_big_file.dta
// from DISC
odbc, exec(“SELECT * FROM my_database.my_table”)
```

1. **I/O:** Takes long to “download” data from P drive/disc into memory
2. **Space complexity**: Difficulty fitting it into memory
3. **Time complexity**: Takes along time to compute the mean of many numbers

```
collapse (mean) myvar
```

- **DISC way:**
  - In HUE:

```
SELECT avg(myvar) FROM my_database.my_table
```

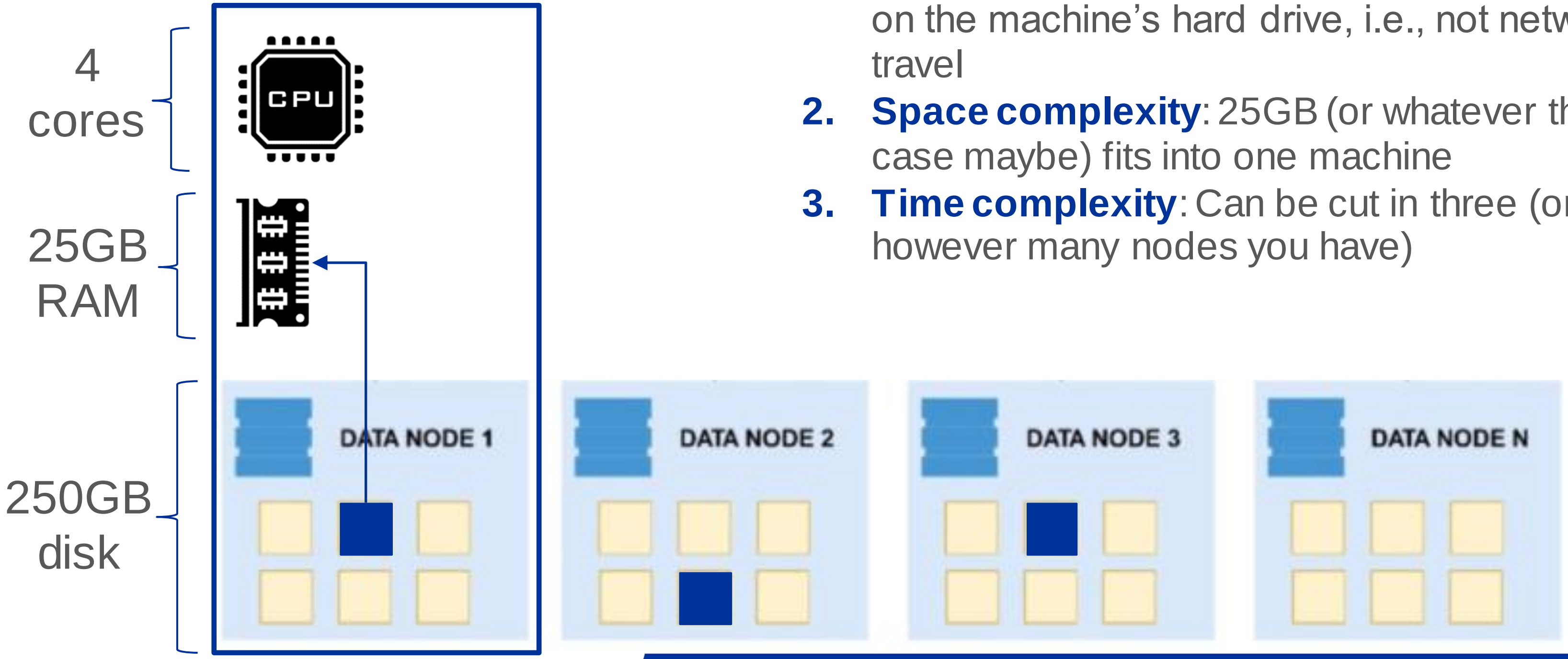
  - Or into Stata/Python/R/…

```
odbc, exec(“SELECT avg(myvar) FROM my_database.my_table”)
```

Example: Average of a huge dataset

- **What did we save?**

1. **I/O:** Takes little time to read in since data is on the machine's hard drive, i.e., not network travel
2. **Space complexity**: 25GB (or whatever the case maybe) fits into one machine
3. **Time complexity**: Can be cut in three (or however many nodes you have)



Let‘s recap…

- **Core strategy:**
  - **Bring the computation to the data!**
  - Don’t bring the data to the computation!
  - Mindset: My computer is only used to the very final plotting/tabling, no hardcore computations!
- **Be server centric**
  - (Move?) Have all your data on Hadoop („staging area“), not network drives
  - Distributed computing is the default storage solution for large datasets, it is here to stay and its peformance will continually increase
  - In 10 years no one will do computations on their laptops, only on servers/cloud.
- **Do all „expensive“ computation in Impala/Hive/Spark/Cloud**
  - Whenever you load a lot of data into your computers memory ask yourself: Couldn’t you do this on Hadoop/Spark?
  - That way you can store intermediate datasets

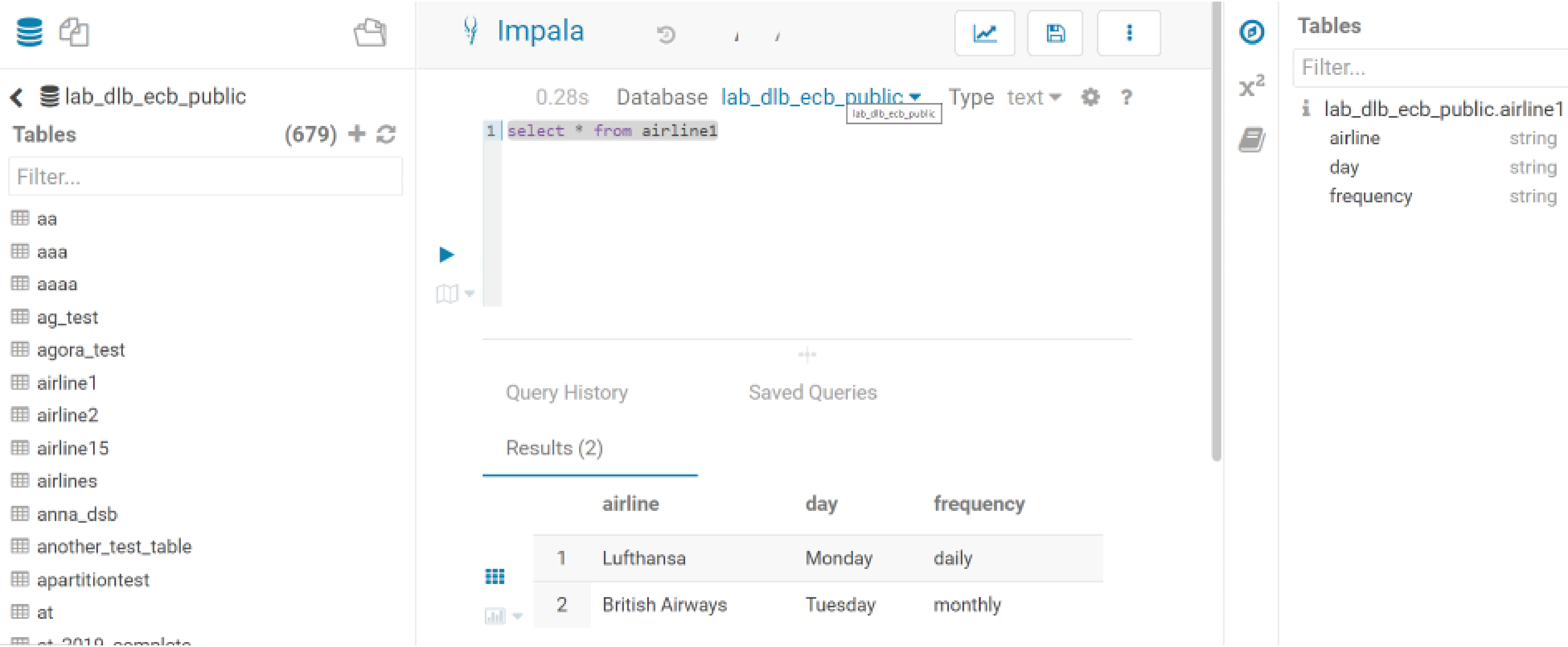
But isn‘t that just like Oracle/SQLServer?

- **On first sight „yes“**: Both systems would encourage you to do the heavy lifting on the system rather than your laptop
- **But when you dive deeper:** Hadoop/Spark offer considerable advantages…
  - Comparable computing power is much more expensive using a dataware house system
  - Blackbox, algorithms are secret, no file system
  - Hadoop/Spark is open source, huge developer/support community
  - Spark is the go-to solution for big data machine learning problems
  - Spark is natively supported by most data lake solutions (AWS S3, …)
  - Spark offers much more flexible computing solutions
  - All the cool kids use it!

But isn‘t that just like Oracle/SQLServer?

- **On first sight „yes“**: Both systems would encourage you to do the heavy lifting on the system rather than your laptop
- **But when you dive deeper:** Hadoop/Spark offer considerable advantages…
    - Comparable computing power is much more expensive using a dataware house system
    - **Blackbox, algorithms are secret, no file system**
    - Hadoop/Spark is open source, huge developer/support community
    - **Spark is the go-to solution for big data machine learning problems**
    - Spark is natively supported by most data lake solutions (AWS S3, …)
    - Spark offers much more flexible computing solutions
    - **All the cool kids use it!**

Let‘s have a look at HDFS



Let‘s have a look at HDFS

File Browser

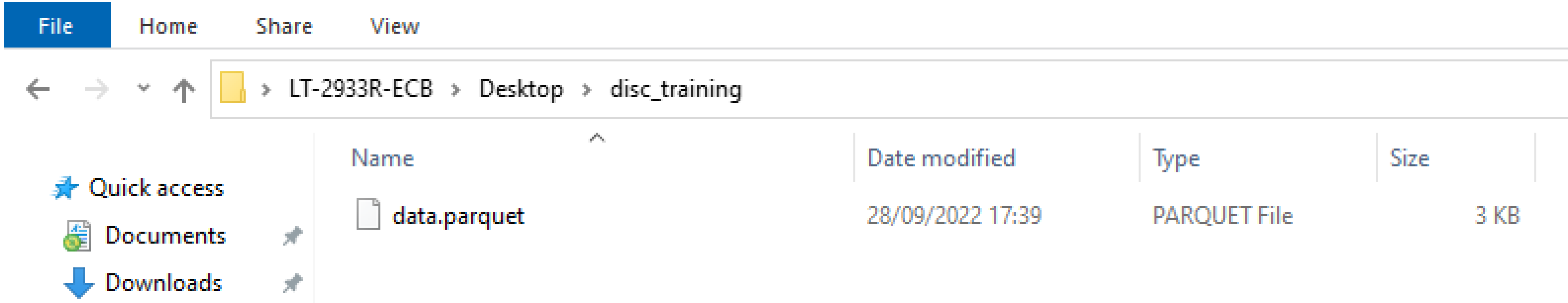
Search for file name | Actions | Move to trash | Upload | New

Home | hdfs:/ data / lab / dlb_ecb_public / **db** | Trash

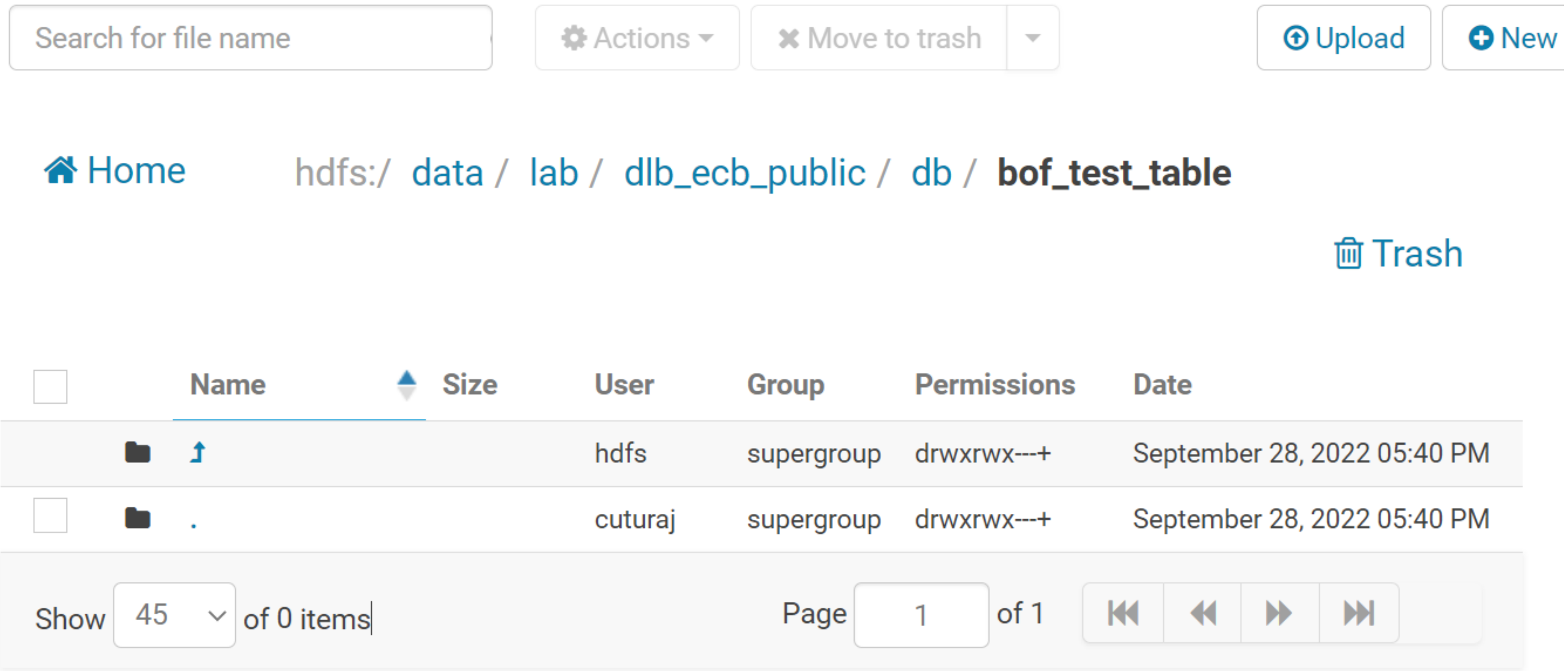
Name	Size	User	Group	Permissions	Date
↥		hdfs	supergroup	drwxrwxr-x	September 28, 2022 01:23 AM
.		hdfs	supergroup	drwxrwx---+	September 28, 2022 12:04 PM
ASampleDatabase.accdb	620.0 KB	alvareo	supergroup	-rw-rw----+	September 06, 2021 03:40 PM
GIOVANNI_temp		mingare	supergroup	drwxrwx---+	March 10, 2021 08:33 PM
Julia		mingare	supergroup	drwxrwx---+	October 18, 2021 04:13 PM
My_new_table.parq	2.1 KB	mingare	supergroup	-rwxrwxr-x+	June 07, 2022 02:27 PM
_impala_insert_staging		impala	supergroup	drwxrwx---+	April 21, 2021 06:33 PM

Let‘s upload a parquet dataset and create a table

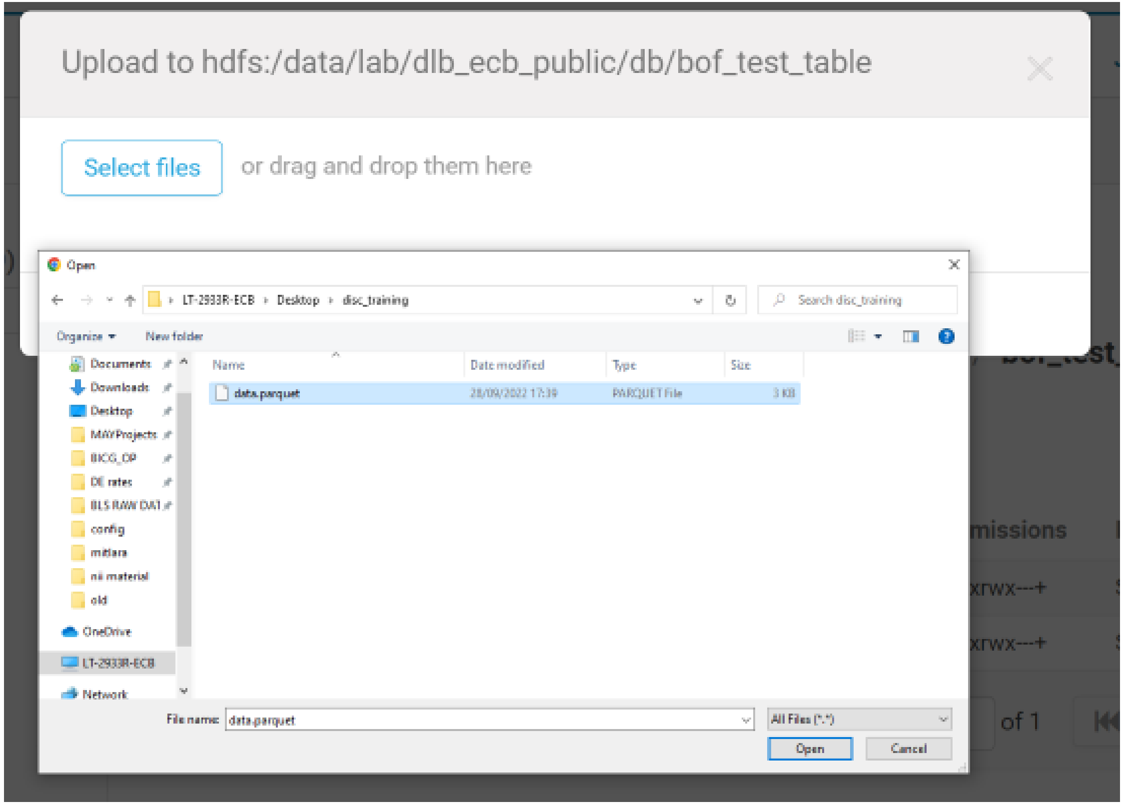
```python
import pandas as pd

# some dataset you have
df = pd.DataFrame(data={'col1':[1,2,3,4], 'col2':['a','b','c','d']})

# save to disk
df.to_parquet("disc_training/data.parquet")
```



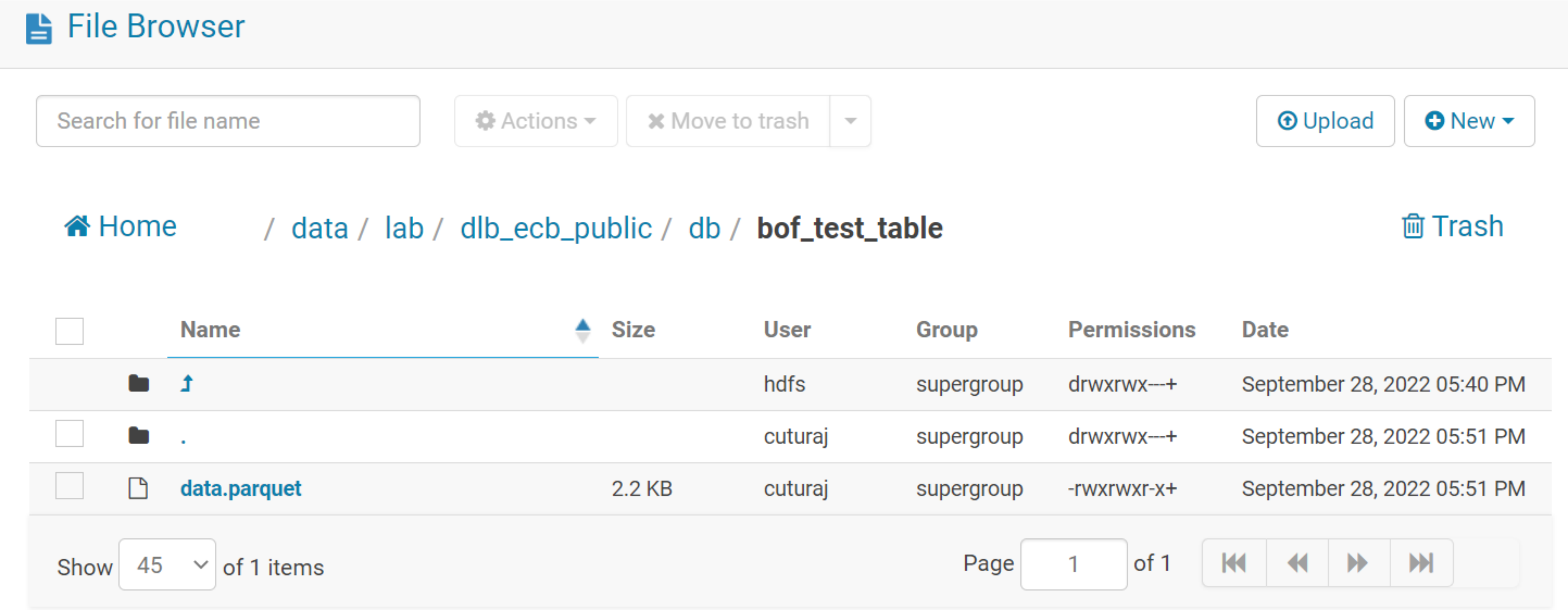
Let‘s upload a parquet dataset and create a table



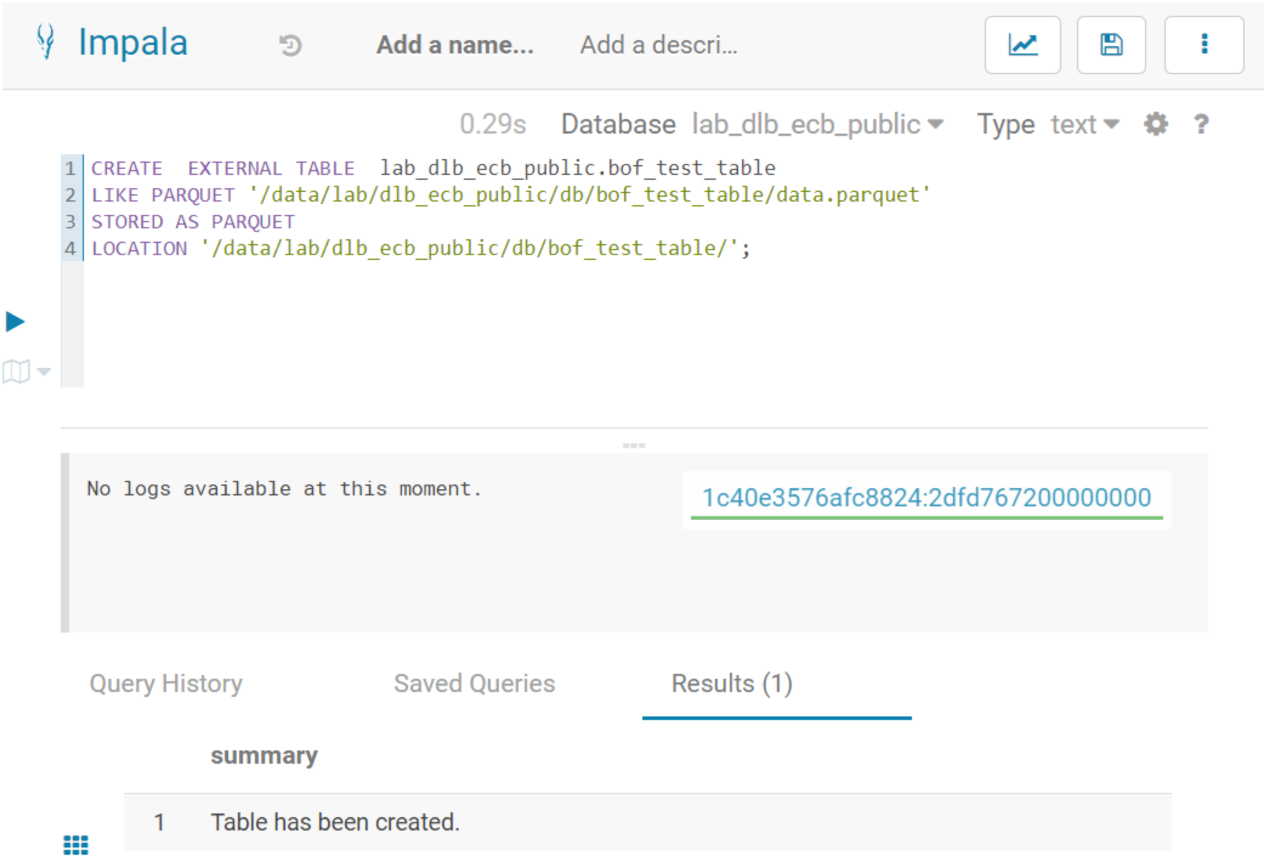
Let‘s upload a parquet dataset and create a table



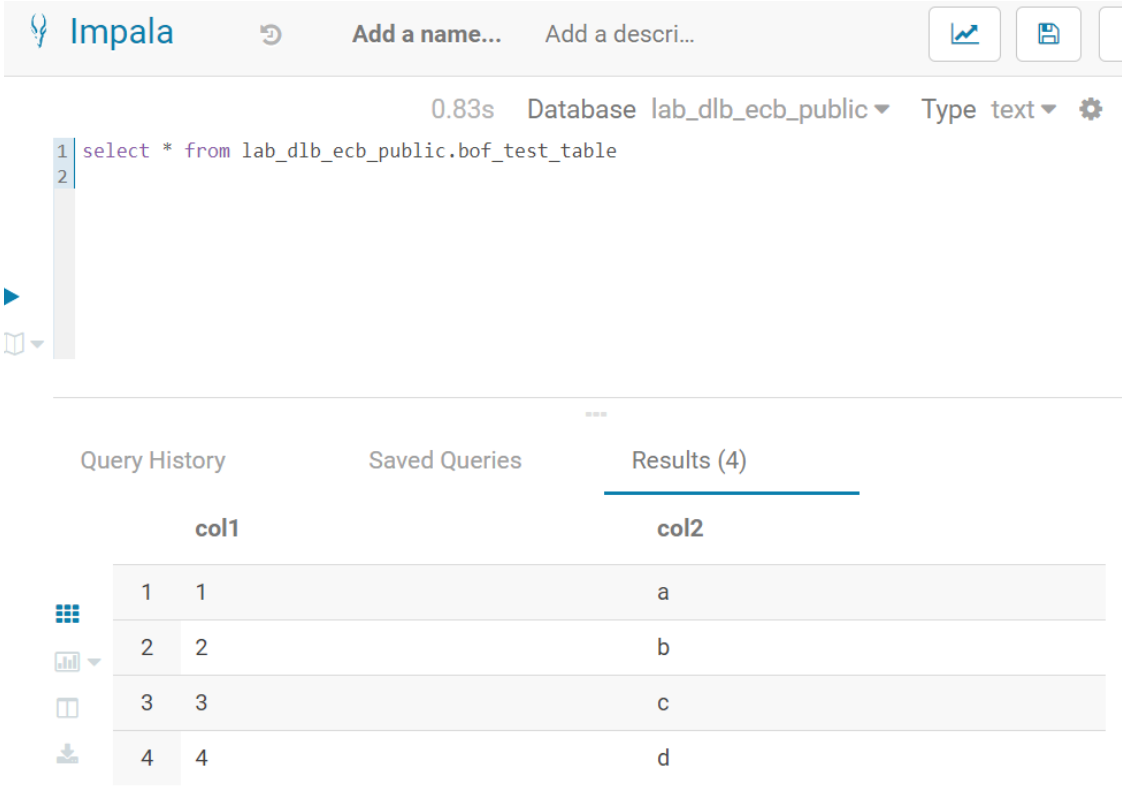
Let‘s upload a parquet dataset and create a table



Let‘s upload a parquet dataset and create a table

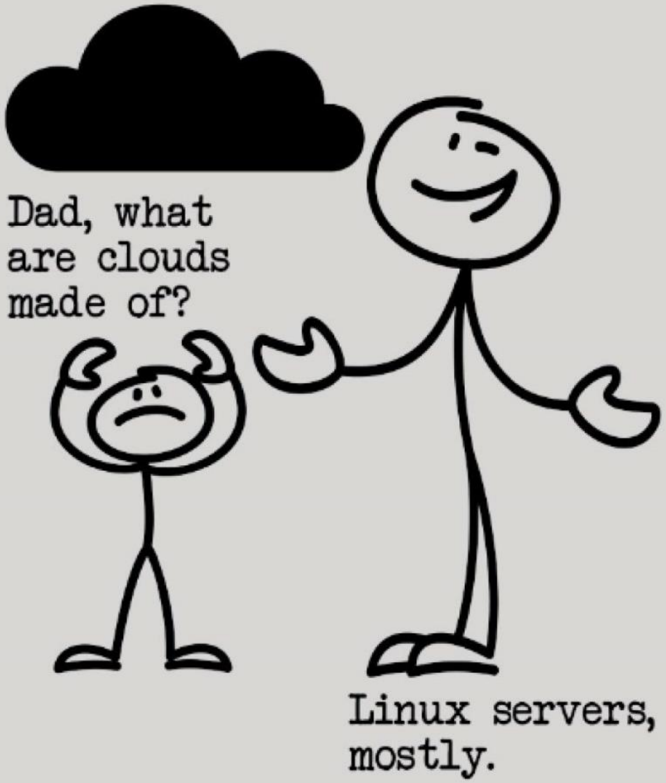


Let‘s upload a parquet dataset and create a table



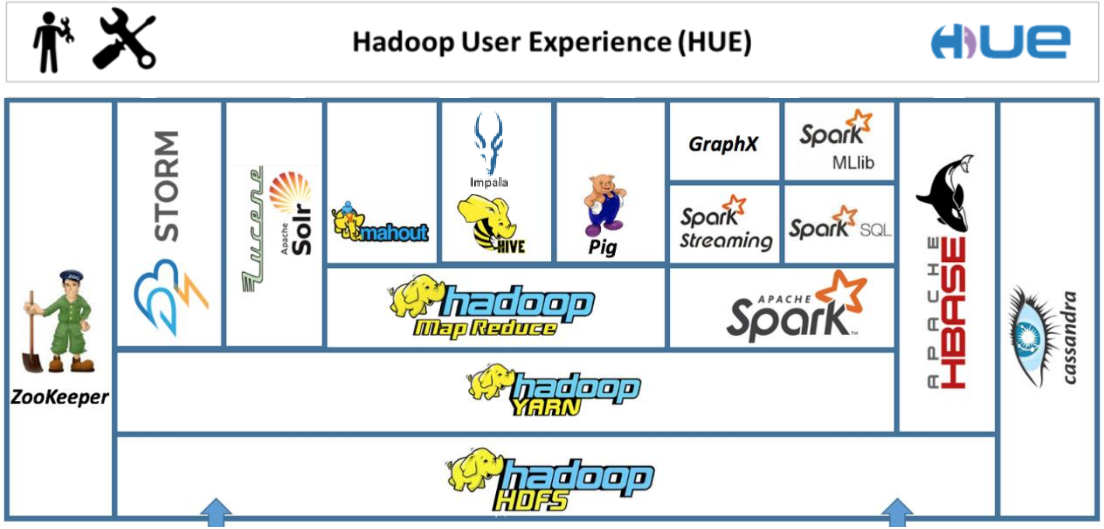
Let‘s automate that!

```python
import pandas as pd

# some dataset you have
df = pd.DataFrame(data={'col1':[1,2,3,4], 'col2':['a','b','c','d']})

from connectors import disc
disc.create_table(df,
                  path='/data/lab/dlb_ecb_public/db/bof_test_table/',
                  lab='lab_dlb_ecb_public')
```



A quick tour through the Hadoop stack

The Hadoop Stack („zoo“)



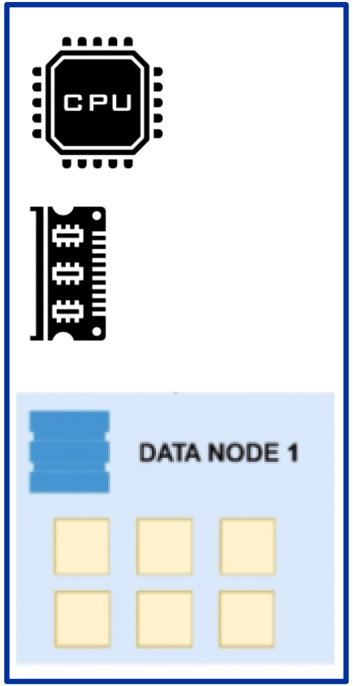
- **H**adoop **U**ser **E**xperience (**HUE**) is a graphical interface to the HDFS system
- **Pig** is a SQL-styled scripting Language
- **Impala** super fast SQL engine
- **Hive** SQL engine & metastore
- **Spark** is an in-memory map-reduce technology
- **Hbase** and **Cassandra** are No-SQL type databases
- **Mahout** is a machine learning library
- **Y**et **A**nother **R**esource **N**egotiation (**YARN**) plans the execution of computation across the cluster

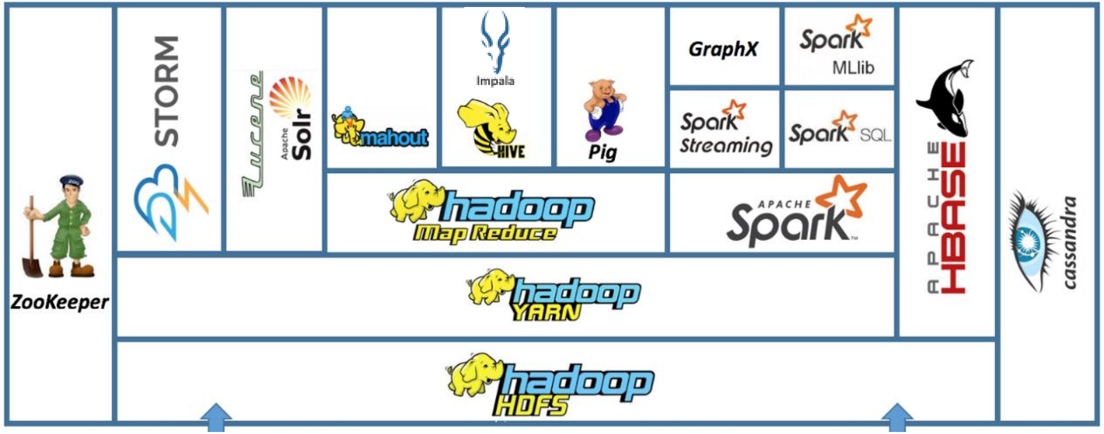
The Hadoop Stack („zoo“): Who does what?

4 cores

25GB disk

250GB disk



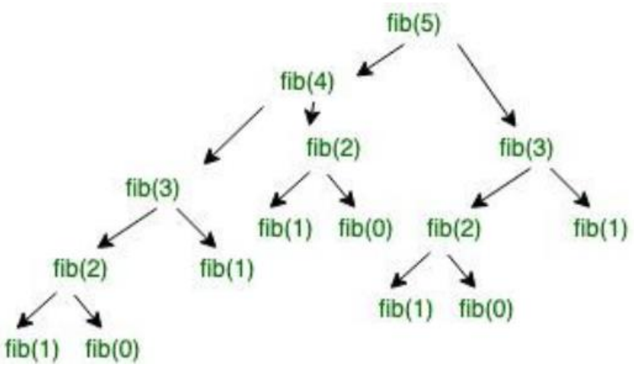


Appendix

Fibonacci revisited

- Different implementations of the Fibonacci sequence are a great way to understand runtime complexity.
- The naïve, recursive implementation takes exponential time since the necessary computations grow exponentially

```python
def fibonacci_recursive(n):
    """"Recursive implementation, Exponential time"""
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci_recursive(n - 1) + fibonacci_recursive(n - 2)
```



Source: https://www.geeksforgeeks.org/introduction-to-recursion-data-structure-and-algorithm-tutorials/