Università degli Studi di Messina

Data Science

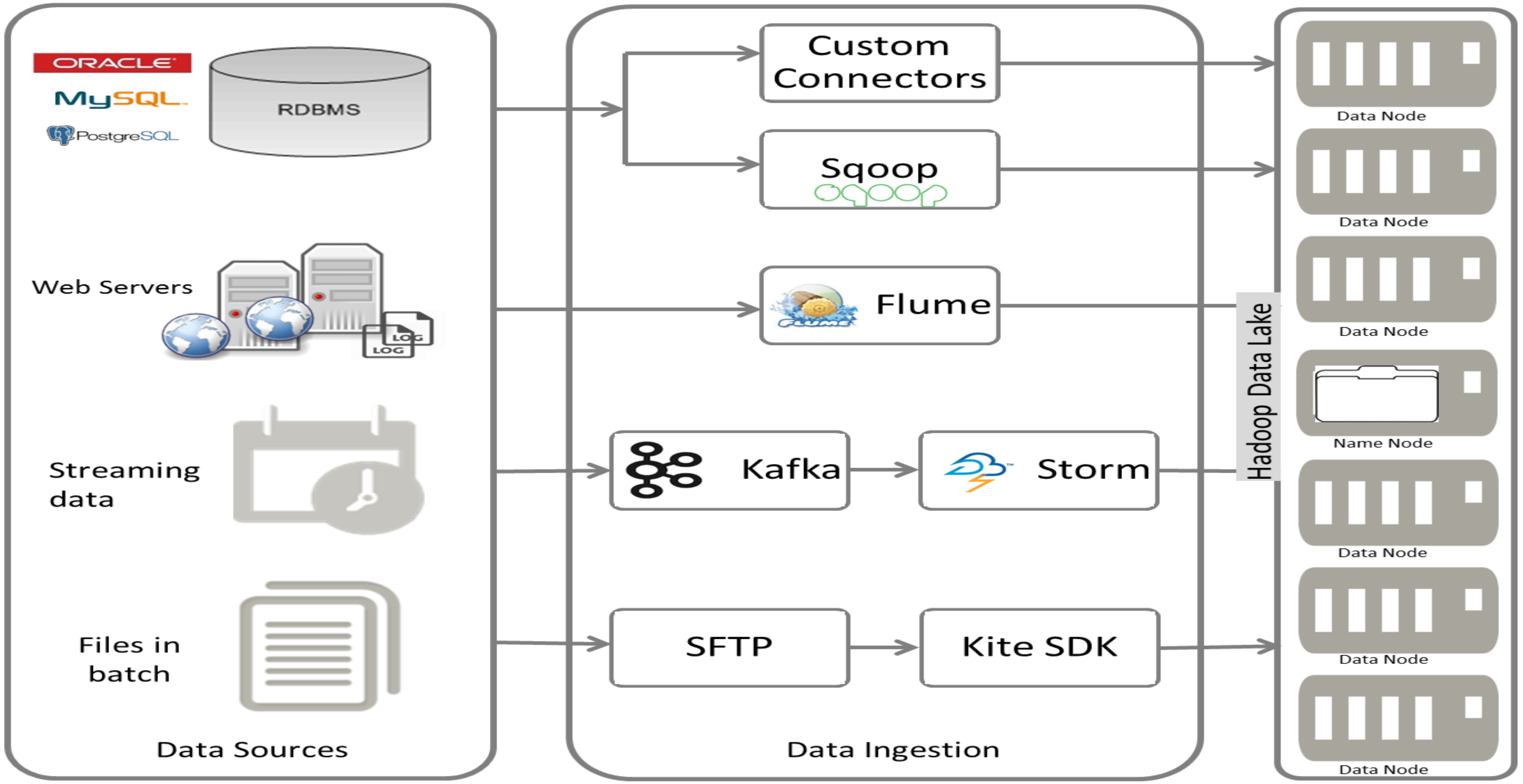
Sqoop

Prof. Daniele Ravi

Honorary Associate Professor
University College London (UCL) – UK
Department of Computer Science
d.ravi@ucl.ac.uk

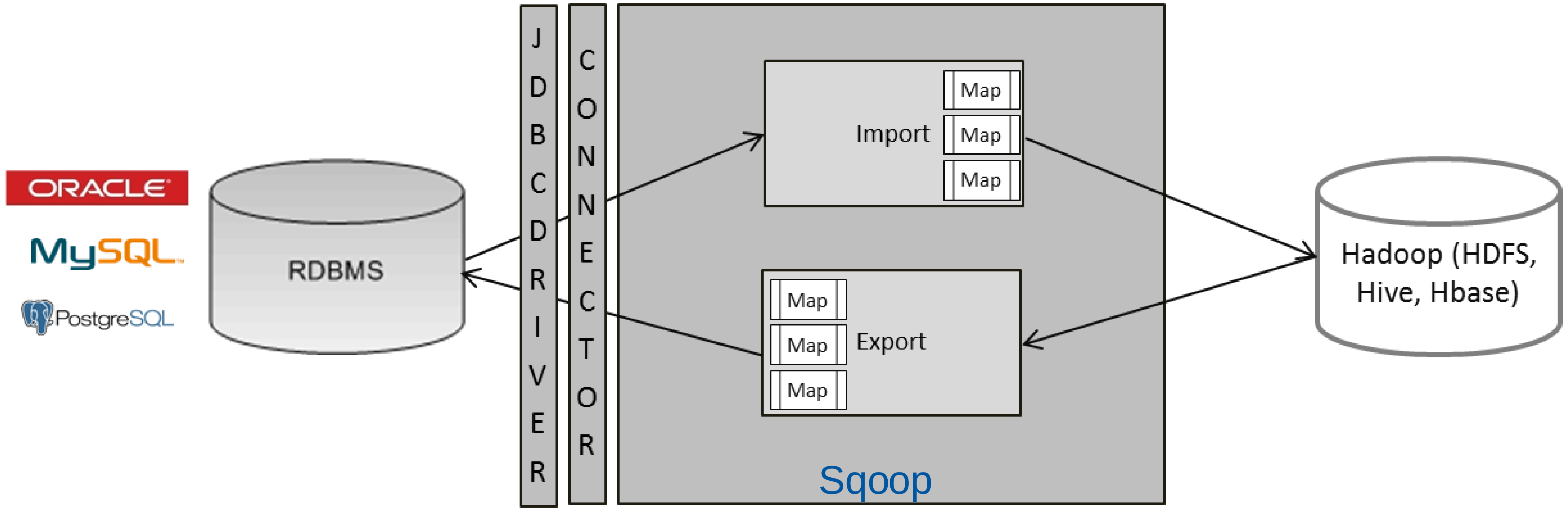
dravi@unime.it

Data Ingestion



Sqoop – SQL to Hadoop

- Open source tool to extract data from structured data store into Hadoop
- Architecture



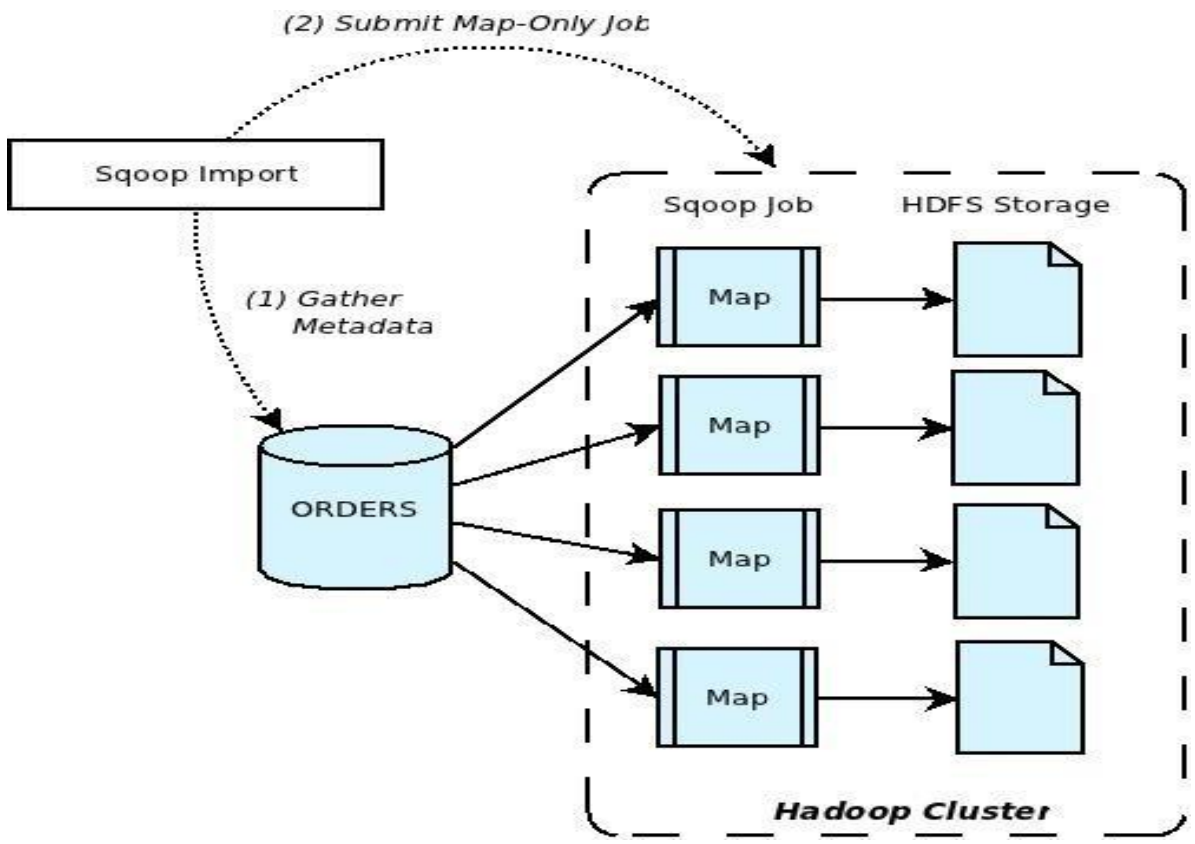
INTRODUCTION

- When Big Data storages and analyzers such as MapReduce, Hive, HBase, Cassandra, Pig, etc. of the Hadoop ecosystem came into picture.
- They required a tool to interact with the relational database servers for importing and exporting the Big Data residing in them.
- Sqoop occupies a place in the Hadoop ecosystem to provide feasible interaction between relational database server and Hadoop' s HDFS.

SQOOP- DEFINITON

- Sqoop: “SQL to Hadoop and Hadoop to SQL”.
- Tool to transfer data from relational databases Teradata, MySQL, PostgreSQL, Oracle, Netezza.
- It is provided by the Apache Software Foundation.

WORKING OF SQOOP



SQOOP IMPORT

- The import tool imports individual tables from RDBMS to HDFS.
- Each row in a table is treated as a record in HDFS.
- All records are stored as text data in text files

SQOOP EXPORT

- The export tool exports a set of files from HDFS back to an RDBMS.

- The files given as input to Sqoop contain records, which are called as rows in table.

- Those are read and parsed into a set of records and delimited with user-specified delimiter.

FEATURES OF SQOOP

- Full Load.
- Incremental Load.
- Parallel import/export.
- Import results of SQL query.
- Compression.
- Connectors for all major RDBMS Databases.
- Kerberos Security Integration.

ADVANTAGES OF SQOOP

- ✓ Allows the transfer of data with a variety of structured data stores like Postgres, Oracle, Teradata, and so on.
- ✓ Sqoop can execute the data transfer in parallel, so execution can be quick and more cost effective.
- ✓ Helps to integrate with sequential data from the mainframe.

DISADVANTAGES OF SQOOP

- ✓ It uses a JDBC connection to connect with RDBMS based data stores, and this can be inefficient and less performant.

- ✓ For performing analysis, it executes various map-reduce jobs and, at times, this can be time consuming when there are lot of joins if the data is in a denormalized fashion.

Sqoop

- Sqoop schedules map reduce jobs to effect imports and exports
- Sqoop always requires the connector and JDBC driver
- Sqoop requires JDBC drivers for specific database server, these should be copied to /usr/lib/sqoop/lib
- The command-line structure has the following structure

```
Sqoop TOOL PROPERTY_ARGS SQOOP_ARGS
```

TOOL - indicates the operation that you want to perform, e.g import, export etc

PROPERTY_ARGS - are a set of parameters that are entered as Java properties in the format -Dname=value.

SQOOP_ARGS - all the various sqoop parameters.

Sqoop – How to run sqoop

Example:

```
sqoop import \
--connect jdbc:oracle:thin:@devdb11-s.cern.ch:10121/devdb11_s.cern.ch \
--username hadoop_tutorial \
-P \
--num-mappers 1 \
--target-dir visitcount_rfidlog \
--table VISITCOUNT.RFIDLOG
```

Sqoop – how to parallelize

```
-- table table_name

-- query select * from table_name where $CONDITIONS

-- table table_name
-- split-by primary key
-- num-mappers n

-- table table_name
-- split-by primary key
-- boundary-query select range from dual
-- num-mappers n
```

Sqoop – how to parallelize

--split-by will split your data uniformly on the basis of number of mappersFor free-form query imports, you need to specify 'split-by’.

- When you are importing the result of any particular query, sqoop needs to know the column-name using which it will create splits.
- Whereas, while importing tables, if not specified, it uses the primary key of the table being imported for creating splits. In case your primary key is uneven and not consistent, you can also specify any other column using split-by.

--boundary-query: During sqoop import process, it uses this query to calculate the boundary for creating splits: select min(), max() from table_name.

- In some cases this query is not the most optimal so you can specify any arbitrary query returning two numeric columns using --boundary-query argument. This saves min(split-by) and max(split-by) operations and thereby is more efficient.

- If you know id starts from val1 and ends with val2. Then there is no point to calculate min() and max() operations. This will make sqoop command execution faster.

- In this way you can waive complex default computation to get them. You can simple hardcode them as argument to boundary queries.

Hands On – 2

Use Sqoop to copy an Oracle table to Hadoop

Step 1) Get the Oracle JDBC driver

```
sudo su -
cd /var/lib/sqoop
curl -L https://pkothuri.web.cern.ch/pkothuri/ojdbc6.jar -o ojdbc.jar
exit
```

Step 2) Run the sqoop job

```
sqoop import \
--connect jdbc:oracle:thin:@devdb11-s.cern.ch:10121/devdb11_s.cern.ch \
--username hadoop_tutorial \
-P \
--num-mappers 1 \
--target-dir visitcount_rfidlog \
--table VISITCOUNT.RFIDLOG
```

Hands On – 3

Use Sqoop to copy an Oracle table to Hadoop, multiple mappers

```
sqoop import \
--connect jdbc:oracle:thin:@devdb11-s.cern.ch:10121/devdb11_s.cern.ch \
--username hadoop_tutorial \
-P \
--num-mappers 2 \
--split-by alarm_id \
--target-dir lemontest_alarms \
--table LEMONTEST.ALARMS \
--as-parquetfile
```

Check the size and number of files

```
hdfs dfs -ls lemontest_alarms/
```

Hands On – 4

Use Sqoop to make incremental copy of a Oracle table to Hadoop

Step 1) Create a sqoop job

```
sqoop job \
--create alarms \
-- \
import \
--connect jdbc:oracle:thin:@devdb11-s.cern.ch:10121/devdb11_s.cern.ch \
--username hadoop_tutorial \
-P \
--num-mappers 1 \
--target-dir lemontest_alarms_i \
--table LEMONTEST.ALARMS \
--incremental append \
--check-column alarm_id \
--last-value 0 \
```

Incremental loads

Apache Sqoop supports **incremental loads**, which allow you to fetch only the newly added or updated rows from a database table since the last import.

This is highly useful for optimizing data transfer, especially when working with large datasets.

1. --incremental

- **Description:** Specifies the mode of incremental import.
- **append:** Imports new rows where the value in the specified column is greater than the last imported value.
- **lastmodified:** Imports rows that have been modified since the last import based on a timestamp column.

2. --check-column

- **Description:** Specifies the column to be checked for incremental conditions.
- In append mode, this column should have unique and increasing values (e.g., an ID column).
- In lastmodified mode, this column is typically a timestamp or datetime column indicating when the row was last modified.

3. --last-value

- **Description:** Specifies the last value imported during the previous incremental load.
- In append mode, this would be the maximum value of the check column from the last import.
- In lastmodified mode, this would be the most recent timestamp from the last import.

4. --merge-key

- **Description:** Specifies the key column used to merge incremental data with existing data in the Hadoop ecosystem.
- **Usage:** Used when importing to HDFS or Hive, ensuring that duplicate rows are not created during updates.

Hands On – 4 contd.

Step 2) Run the sqoop job

```
sqoop job --exec alarms
```

Step 3) Run sqoop in incremental mode

```
sqoop import \
--connect jdbc:oracle:thin:@devdb11-s.cern.ch:10121/devdb11_s.cern.ch \
--username hadoop_tutorial \
-P \
--num-mappers 1 \
--table LEMONTEST.ALARMS \
--target-dir lemontest_alarms_i \
--incremental append \
--check-column alarm_id \
--last-value 47354 \

hdfs dfs -ls lemontest_alarms_i/
```