ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

Εξόρυξη γνώσης από Βάσεις Δεδομένων και τον Παγκόσμιο Ιστό

Ενότητα # 7: Introduction to Big Data

Διδάσκων: Μιχάλης Βαζιργιάννης

Τμήμα: Προπτυχιακό Πρόγραμμα Σπουδών “Πληροφορικής”

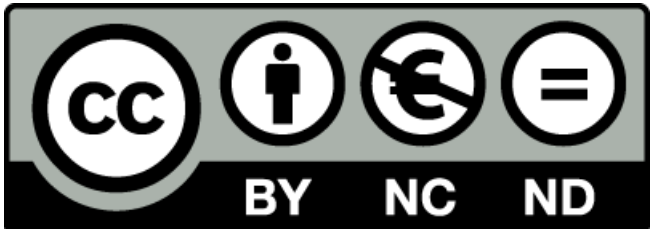
Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Οικονομικό Πανεπιστήμιο Αθηνών**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Οι εικόνες προέρχονται … .



Σκοποί ενότητας

Εισαγωγή και εξοικείωση με τις μεθόδους Map reduce - distributed processing, technologies (Hadoop, Map Reduce, NoSQL storage)

Περιεχόμενα ενότητας

- Why Big data
- Hadoop
- MapReduce
- HDFS
- Hive, HIVE QL

ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

Why Big data

Μάθημα: Εξόρυξη γνώσης από Βάσεις Δεδομένων και τον Παγκόσμιο Ιστό

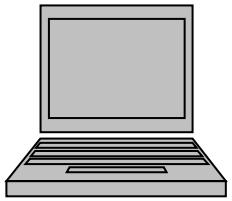
Ενότητα # 7: Introduction to Big Data

Διδάσκων: Μιχάλης Βαζιργιάννης

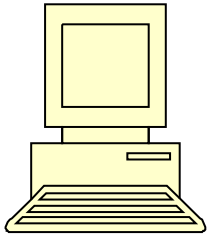
Τμήμα: Προπτυχιακό Πρόγραμμα Σπουδών “Πληροφορικής”

Scaling up…

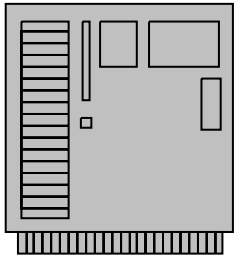
1GBB



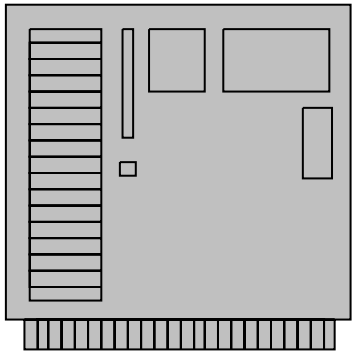
10 GB



100 GB



1 TB



100 TB ???

We are flooding with data

- The New York Stock Exchange generates about ***1TB*** new trade data / day.
- Facebook hosts approximately *10 billion photos*, taking up ***1PB*** *(1000TB)* of storage.
- *ancestry.com*, the genealogy site, stores around **2.5PB** of data.
- The Internet Archive stores around **2 PB** data, and is growing at a rate of *20TB/month*.
- The Large Hadron Collider near Geneva, Switzerland, will produce about **15 PB** data / year

Issues due to storage resources

- Storage capacity have increased massively over the years, BUT not access speeds
- 90's: capacity 1,370 MB, speed:4.4 MB/s - (read a full drive in ~5 mins)
- 2010's: 1 TB, transfer, speed ~100 MB/s,
- (**> 2.5 hours** to read a full drive)
- *writing is even slower.*
- to reduce read time -multiple disks at once.
  - Assume 100 drives, each holding 1/100 of data.
  - Working in parallel, read the data in less than *2 minutes*

Issues due to storage resources

- hardware failure: many pieces of hardware used result in fairly high fail probability
- replication: redundant copies of the data are kept
- in the event of failure another copy available.
- **Hadoop** Distributed Filesystem (HDFS) does replication.
- most analysis tasks combine the data; data read from one disk need to be combined with the data from other disk(s).
- MapReduce provides a programming model that abstracts the problem from disk reads and writes.

ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

Hadoop

Μάθημα: Εξόρυξη γνώσης από Βάσεις Δεδομένων και τον Παγκόσμιο Ιστό

Ενότητα # 7: Introduction to Big Data

Διδάσκων: Μιχάλης Βαζιργιάννης

Τμήμα: Προπτυχιακό Πρόγραμμα Σπουδών “Πληροφορικής”

Successful Big Data cases with Hadoop

- **IBM Watson**

- Watson, a super computer developed by IBM competed in the popular Question and Answer show “Jeopardy!”.
- Watson beat the two most popular players in that game.

- used input approximately 200 million pages of text using Hadoop to distribute the workload for loading this information into memory.

Successful Big Data cases with Hadoop

- **China Mobile**

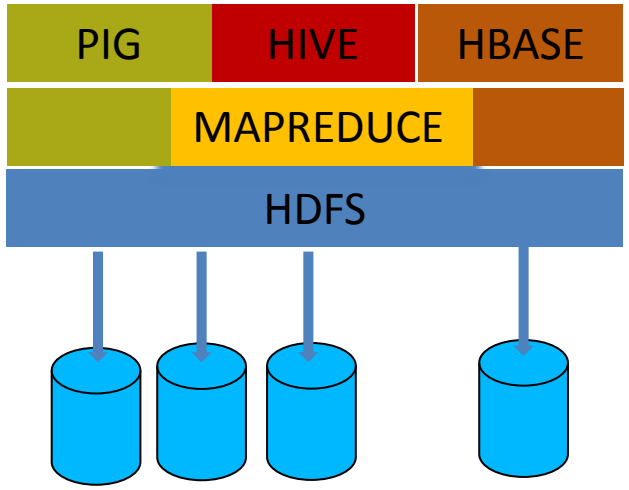
- telecom industry in China, built a Hadoop cluster to perform data mining on Call Data Records.
- China Mobile was producing 5-8TB of these records daily.
- Hadoop-based system enables process 10 times as much data as when using their old system, and at 1/5 of the cost.

- **New York Times**

- wanted to host on their website all public domain articles from 1851 to 1922.
- They converted articles from 11 million image files to **1.5TB** of PDF documents.
- implemented by one employee - ran a job in *24 hours* on a 100-instance Amazon EC2 Hadoop cluster at a very low cost

Basic components of Big Data management

- **HDFS:** A distributed file system that runs on large clusters of commodity machines.
- **MapReduce:** A distributed data processing model and execution environment running on large commodity machines clusters.
- **Pig:** data flow language and execution environment for exploring very large datasets. Pig runs on HDFS and MapReduce clusters.
- **Hive:** distributed data warehouse, managing data stored in HDFS. Provides a query language based on SQL (translated by the runtime engine to MapReduce jobs) for querying the data.
- **HBase:** A distributed, column-oriented database. HBase uses HDFS for its underlying storage, and supports both batch-style computations using MapReduce and point queries (random reads).



ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

MapReduce

Μάθημα: Εξόρυξη γνώσης από Βάσεις Δεδομένων και τον Παγκόσμιο Ιστό

Ενότητα # 7: Introduction to Big Data

Διδάσκων: Μιχάλης Βαζιργιάννης

Τμήμα: Προπτυχιακό Πρόγραμμα Σπουδών “Πληροφορικής”

Introduction to Map Reduce

- MapReduce - programming model developed at Google

- decomposes large data manipulation jobs into individual tasks
- executed in parallel across a cluster of servers.
- the results of the tasks can be joined together

Introduction to Map Reduce

- two fundamental data-transformation operations:

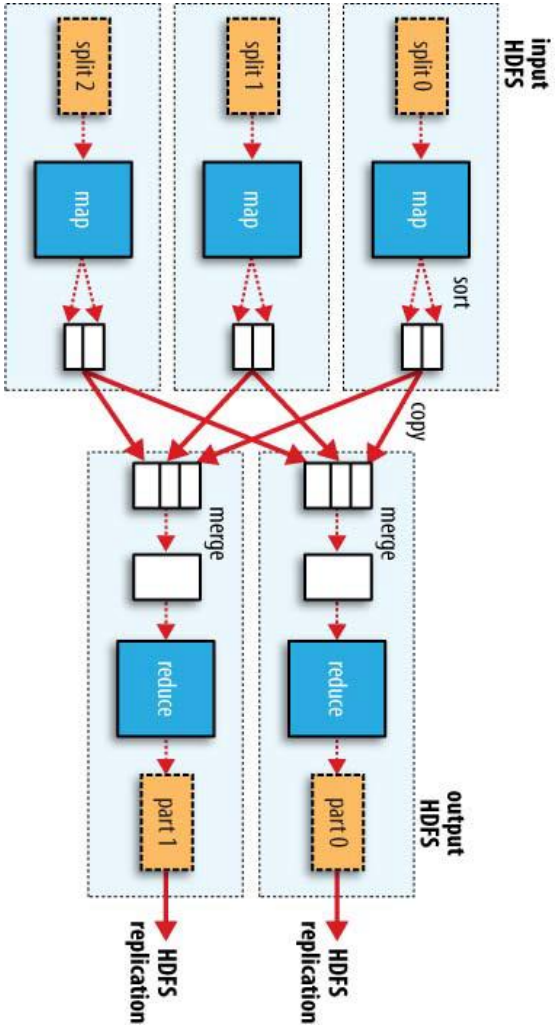
- map operation: converts the elements of a collection from one form to another - input and output values might be completely different.

- all the key- values pairs for a given key are sent to the same reduce operation - the key and a collection of the values are passed to the reducer.

- Reduce operation: convert the values corresponding to a key a value – by summing or averaging a collection of numbers, or to another collection.

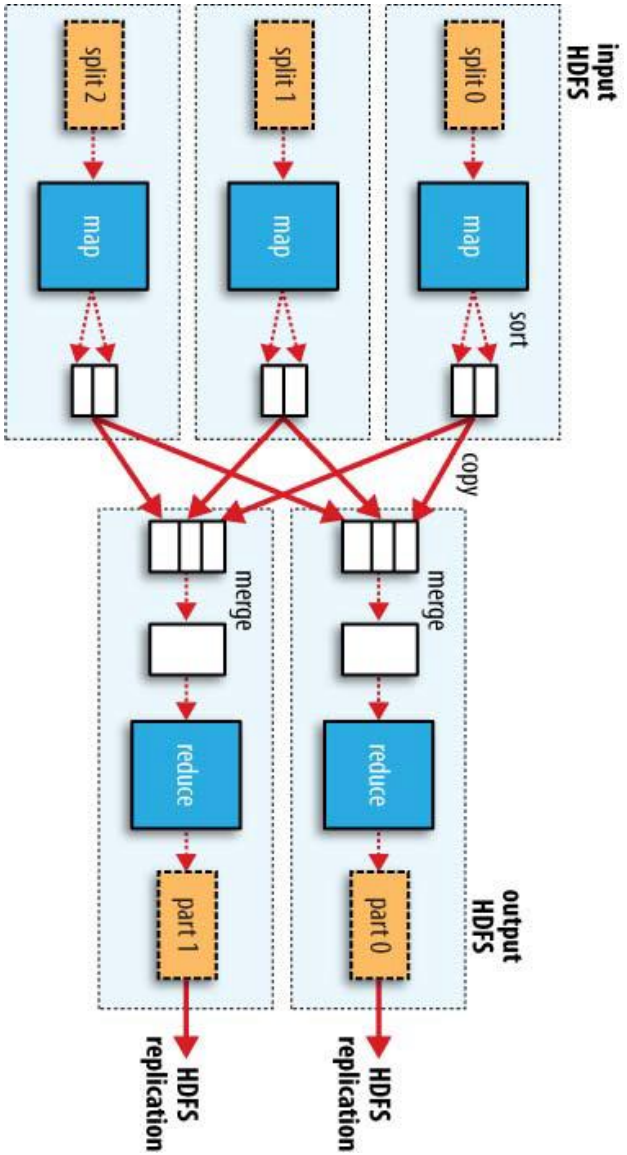
- Each reducer produces one final key-value pair

- Again, the input versus output keys and values may be different. Note that if the job requires no reduction step, then it can be skipped.



Hadoop automates the process

- Hadoop handles most of details required to make jobs run successfully.
- - determines how to decompose the submitted job into individual map and reduce tasks to run,
- schedules those tasks given the available resources,
- decides where to send a particular task in the cluster to minimize network overhead (data locality)
- monitors each task to ensure successful completion, and it restarts tasks that fail.



MapReduce: Motivating example

- National Climatic Data Center(NCDC, http://www.ncdc.noaa.gov/).
- Rich set of meteorological elements,
- basic elements, such as temperature - fixed width
- Data files are organized by date and weather station – for 100 years (1901 – 2001)
- Assuming tens of thousands of stations, total number (small) files: $100*365*10^4 \sim 10^9$
- more efficient to process a smaller number of relatively large files, so the data concatenated into *a single file*.

- The objective is to make a list: ***{<year, max temperature>}***

Solution with Unix Tools

Example 2-2. A program for finding the maximum recorded temperature by year from NCDC weather records

```
#!/usr/bin/env bash

for year in all/*

do

        echo -ne `basename $year .gz`"\t"

        gunzip -c $year | \

                awk '{ temp = substr($0, 88, 5) + 0;

                        q = substr($0, 93, 1);

                                if (temp !=9999 && q ~ /[01459]/ && temp > max) max =
temp }

                        END { print max }'

Done
```

- **The complete run for the century took 42 minutes in one run on a single EC2 High-CPU Extra Large Instance**
- **Trying to run the task in parallel incurs several difficulties.**

Analyzing the Data with Hadoop

- MapReduce breaks the processing into two phases: **map phase** and **reduce phase**.
- Each phase has key-value pairs as input and output, the types of which may be chosen by the programmer.
- The programmer specifies two functions:
- the map function and the reduce function

Analyzing the Data with Hadoop

- map function

- We pull out the year and the air temperature from the file setting up the data in such a way that the reducer function can do its work on it: finding the maximum temperature for each year.
- The map function is also a good place to drop bad records: here we filter out temperatures that are missing, suspect, or erroneous.

Map function

- **INPUT** - the key-value pairs:
- (0, 006701199099999**1950**051507004...9999999N9+**0000**1+99999999999...)
- (106, 004301199099999**1950**051512004...9999999N9+**0022**1+99999999999...)
- (212, 004301199099999**1950**051518004...9999999N9-**0011**1+99999999999...)
- (318, 004301265099999**1949**032412004...0500001N9+**0111**1+99999999999...)
- (424, 004301265099999**1949**032418004...0500001N9+**0078**1+99999999999...)

- **MAP function:**
- **i.** extracts the year and the air temperature (indicated in bold text),
- ii. emits them as its output (the temperature values have been interpreted as
- integers):
- **OUTPUT: (1950, 0), (1950, 22), (1950, −11), (1949, 111), (1949, 78)**

Mapper maximum temperature example

```
...
public class NewMaxTemperatureMapper {
        static class NewMaxTemperatureMapper
        extends Mapper <LongWritable, Text, Text, IntWritable> {
        private static final int MISSING = 9999;
// map function parameters: input key/value, output key/value types
// in this case: input file offset/text line , output year/max-temperature
        public void map(LongWritable key, Text value, Context context) throws       IOException
InterruptedException {
                String line = value.toString(); //Retrieve text line @ key position
                String year = line.substring(15, 19); //isolate the year value
                 int airTemperature;
                if (line.charAt(87) == '+') {//parseInt doesn't like leading + signs
                        airTemperature=Integer.parseInt(line.substring(88, 92)); } else {
                        airTemperature = Integer.parseInt(line.substring(87, 92)); }
                        String quality = line.substring(92, 93);
                if (airTemperature != MISSING && quality.matches("[01459]")) {
                // context writes the to the  output: year, max_temperature
                        context.write(new Text(year), new IntWritable(airTemperature));
}
}                       } }
```

Reduce function

- **(1950, 0), (1950, 22), (1950, –11), (1949, 111), (1949, 78)**

- *iterate through the list and pick up the maximum reading:*

- **(1949, 111), (1950, 22)**

- final output: the maximum global temperature recorded in each year.

Reduce function

```
import java.io.IOException; import java.util.Iterator;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
// reducer inputs must match the output types of the mapper
// for each year: compute max temperature
Static class NewMaxTemperatureReducer
extends Reducer <Text, IntWritable, Text, IntWritable>
{
// for each key (year) compute the max temperature from values
// and write the pair key (year), value (max_temprature) to the context/output
public void reduce(Text key, Iterable<IntWritable> values,
        Context context) throws IOException {
                int maxValue = Integer.MIN_VALUE;
                for (Intwritable value : values) {
                        maxValue = Math.max(maxValue, value.get());
                }
                context.write(key, new IntWritable(maxValue)); }
}
```

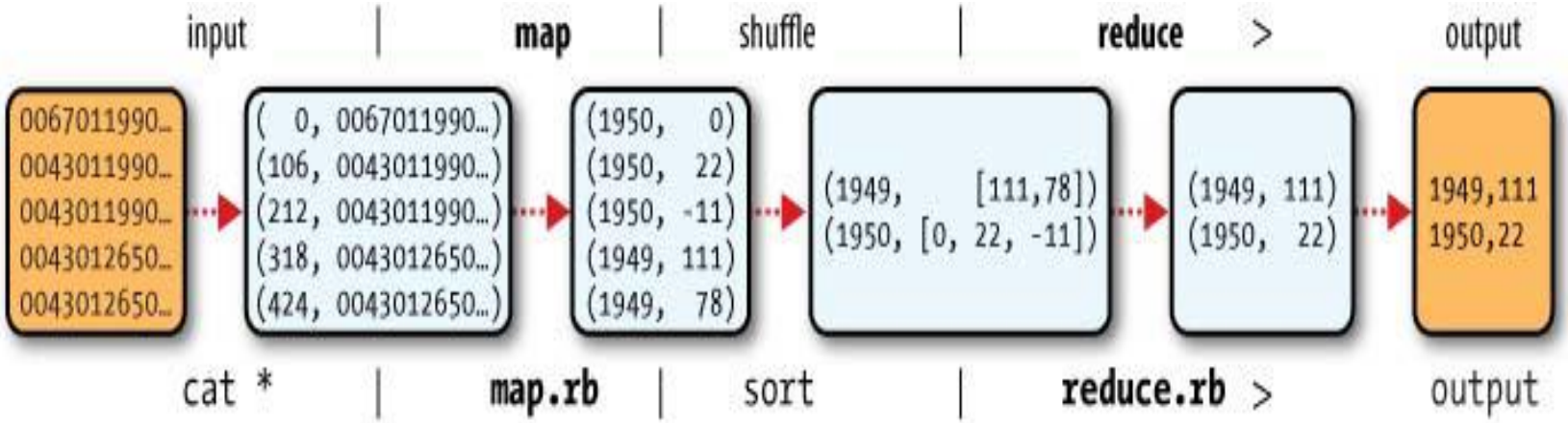
MAX TEMPERATURE APPLICATION

```
import java.io.IOException;import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.FileInputFormat; import org.apache.hadoop.mapred.FileOutputFormat; import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;



public class NewMaxTemperature {
        public static void main(String[] args) throws  Exception {
        if (args.length != 2) {
                                        System.err.println("Usage: MaxTemperature <input path> <output path>");
                                        System.exit(-1); }
        // object to run the job
        Job  job = new Job();
        Job.setJarByClass(NewMaxTemperature.class);
        FileInputFormat.addInputPath(conf, new Path(args[0])); //input data location
        FileOutputFormat.setOutputPath(conf, new Path(args[1])); //output data path
        job.setMapperClass(NewMaxTemperatureMapper.class); // set the map process
        job.setReducerClass(NewMaxTemperatureReducer.class); // set the reduce process
        job.setOutputKeyClass(Text.class); //type of the output key objects job.setOutputValueClass(IntWritable.class); //output value objects type
        JobClient.runJob(conf);
        System.exit(job.waitForCompletion(true) ? 0 : 1);
        }
```

MapReduce lifecycle



Data Flow

- A MapReduce job is a unit of work to be performed and consists:

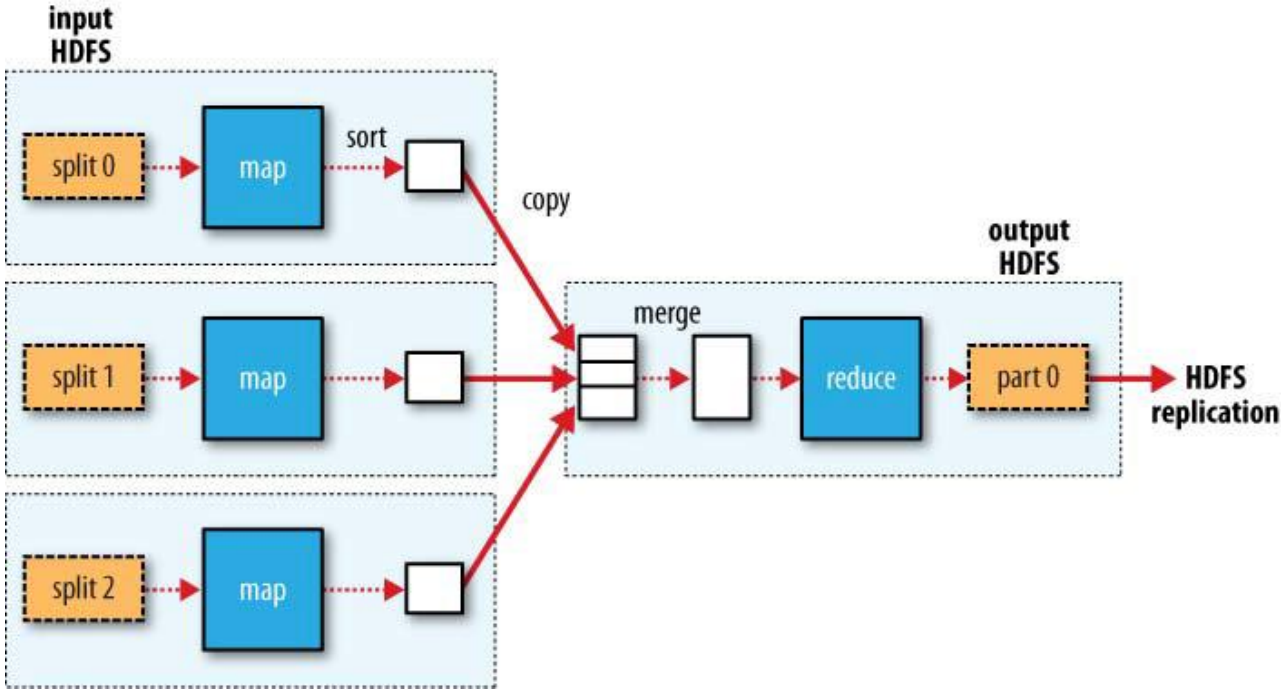
- input data, the MapReduce program, and configuration information.
- Hadoop runs the job by dividing it into tasks, of which there are two types: map tasks and reduce tasks.

- two types of nodes control the job execution process:

- a jobtracker: i. coordinates all the jobs run on the system by scheduling tasks to run on tasktrackerswhich ii. keeps a record of the overall progress of each job. If a task fails, the jobtracker can reschedule it on a different tasktracker.

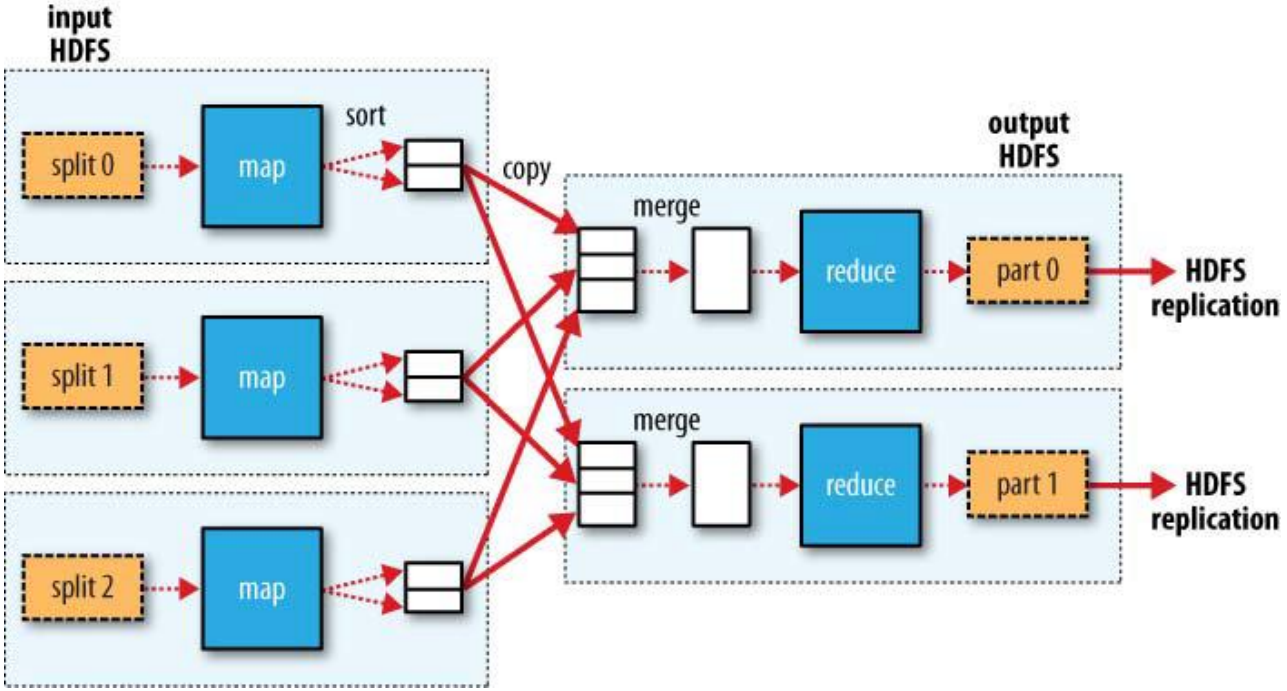
Map task

- Hadoop splits input in fixed size splits
- Map task runs on the same node



Data flow with multiple reduce tasks

- When there are multiple reducers, the map tasks *partition* their output, each creating one partition for each reduce task.



Another example of an application for MapReduce

- Inverted index creation of a search engine:
  - Dataset of 1TB, 25 million Web pages
    - (GOV2 - http://ir.dcs.gla.ac.uk/test_collections/access_to_data.html)
  - Cluster of 5 machines (Intel Core 2, 4GB of RAM)
  - 10 hours needed instead of 72 hours

ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

HDFS

Μάθημα: Εξόρυξη γνώσης από Βάσεις Δεδομένων και τον Παγκόσμιο Ιστό

Ενότητα # 7: Introduction to Big Data

Διδάσκων: Μιχάλης Βαζιργιάννης

Τμήμα: Προπτυχιακό Πρόγραμμα Σπουδών “Πληροφορικής”

Hadoop Distributed File System - HDFS

- Node: a single – commodity - computer
- Rack: a collection of 30 or 40 nodes physically stored close together all connected to the same network switch). Network bandwidth between any two nodes in rack is greater than bandwidth between two nodes on different racks.
- Hadoop Cluster is a collection of racks

- Hadoop major components:

•- Hadoop Distributed File System (HDFS)

- - MapReduce component, which is a framework for performing computations on the data in the distributed file system.

HDFS Design

- HDFS designed for
- storing very large files (currently handles up to Petabyte files)
- streaming data access patterns
  - analysis tasks on large files involve a large proportion, if not all, of the dataset,
  - time to read/scan the whole dataset is more important than the latency to seek the first record.
- running on clusters of commodity hardware.
  - Can handle failures via replication

HDFS design

- HDFS is NOT designed for:

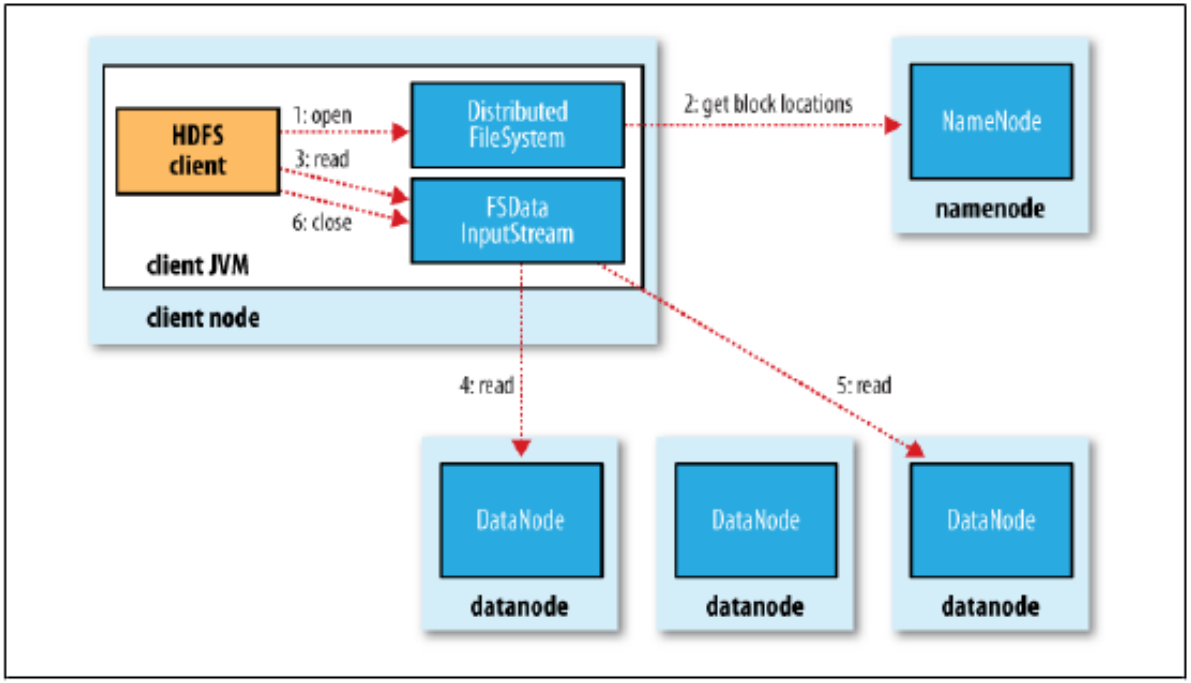
- Direct access to files (Hbase does..)
- Lots of small files
  - namenode holds file system metadata in memory,
  - number of files in a filesystem constrained by the memory on the namenode.
    - a file or directory name ~150 bytes. For 1M files, need ~300 MB of memory.
    - handling millions of files feasible, billions is beyond current hardware capacities
- Multiple and arbitrary file modifications
  - Files in HDFS may be written to by a single writer and always at the end of the file.

Large Blocks in HDFS …

- HDFS blocks are large compared to disk blocks (few KB)
  - to minimize the cost of seeks. By making a block large enough, the time to transfer the data from the disk can be made to be significantly larger than the time to seek to the start of the block.
  - thus time to transfer a large file made of multiple blocks operates at the disk transfer rate – few seeks
- Example:
- seek time ~10 ms, transfer ~100 MB/s,
- seek time <=1% transfer time -> block size ~100 MB.
- the default ~ 64 MB, although many HDFS installations use 128 MB blocks.
- Block size is expected to grow as new generations of disk drives provide faster transfer rates.

HDFS design

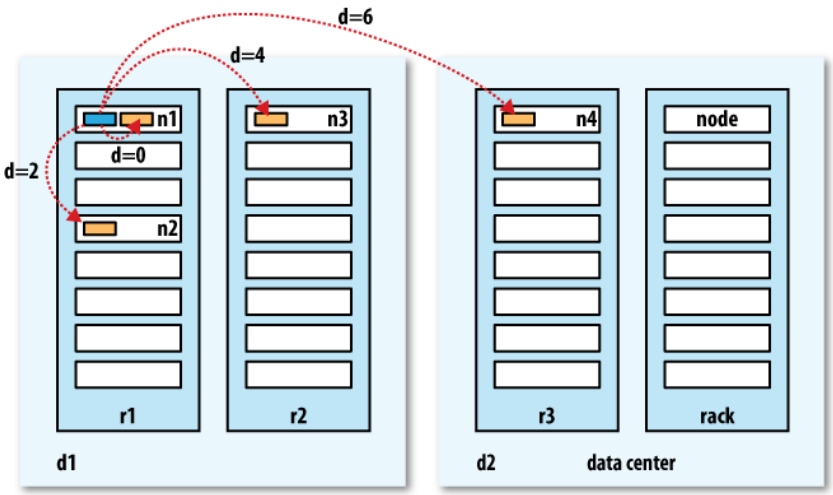
- Name node (master) – data nodes (the workers)
- Data nodes store and retrieve blocks when they are told to (by clients or the name node), and report back to the name node periodically with lists of blocks that they are storing.



HDFS – network topology

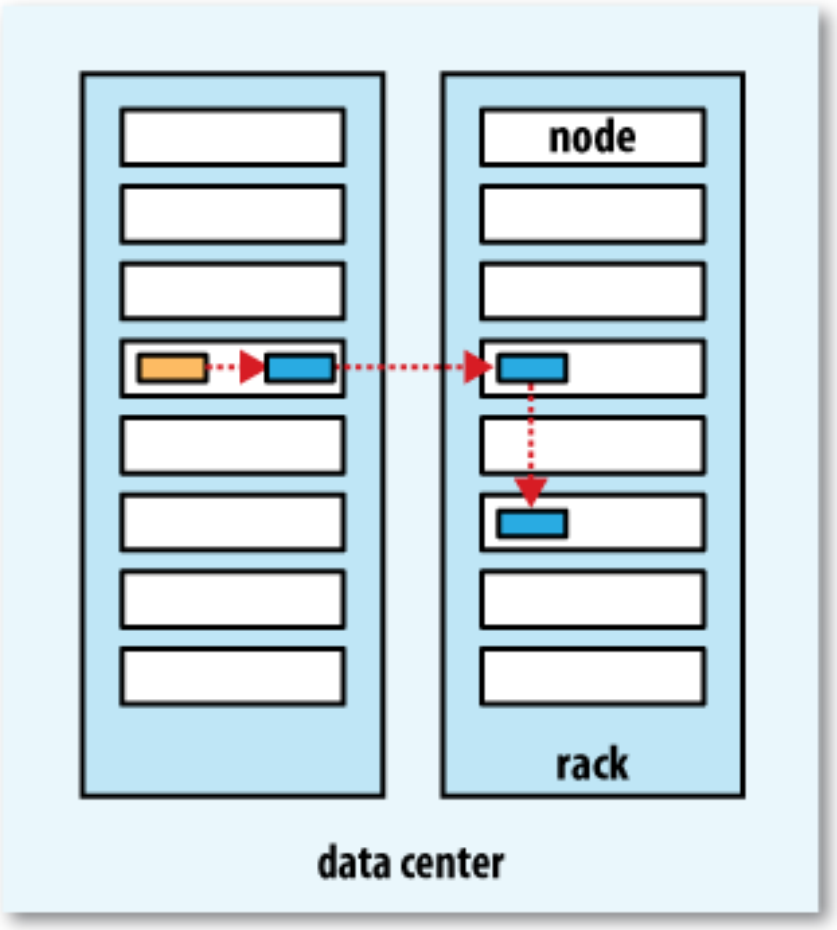
Need to place data such that bandwidth between nodes for an analysis is optimal

- distance(/d1/r1/n1, /d1/r1/n1) = 0 (processes on the same node)
- distance(/d1/r1/n1, /d1/r1/n2) = 2 (different nodes on the same rack)
- distance(/d1/r1/n1, /d1/r2/n3) = 4 (nodes on different racks in the same data center)
- distance(/d1/r1/n1, /d2/r3/n4) = 6 (nodes in different data centers)



HDFS - replication

- 3 copies:
- Same node
- Random node n_1 in another rack r_j
- Random node n_2 in same rack r_j



ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

Hive, HIVE QL

Μάθημα: Εξόρυξη γνώσης από Βάσεις Δεδομένων και τον Παγκόσμιο Ιστό

Ενότητα # 7: Introduction to Big Data

Διδάσκων: Μιχάλης Βαζιργιάννης

Τμήμα: Προπτυχιακό Πρόγραμμα Σπουδών “Πληροφορικής”

Introduction to Hive

- more and more data becoming available every day, the need to deploy a distributed framework becomes apparent.
- need to keep things simple and allow users to (continue to) interact with the data using SQL (which is easy and intuitive)
- avoid having developers writing long and difficult programs in map-reduce

Hive

Hive (by facebook) is a database

- provides an abstraction layer between the user and tha data stored in hadoop
- allows quering and analyzing that data using a SQL dialect (quite similar to MySQL)

Suited for applications that require a data warehouse with

- relatively static data
- no need for a fast response time.

Hive abilities

Hive can facilitate:

- developers in porting current code using traditional databases to hadoop.
- Ease users in the transition as it uses a SQL dialect. (no new languages-or tools required)
- Be extended by new user-defined functions(UDF)
- Integrated with Amazon's Services (i.e.*Elastic Mapreduce*)

Hive is not made for:

- transactions management (record-level, updates,inserts or deletions)
- Providing “realtime” usage due to the mapreduce overhead and the amount of data.

Hive Interaction

Hive can be accessed the command line interface (CLI)

Using the CLI :

$ hive -e “select id from users limit 10”

(displays at most 10 ids)

$ hive -S -f query.sql > query_results

$ cat query_results

(-S doesn't display mapreduce messages and -f selects a sql file)

$ hive --help (for more)

HiveQL

Primitive Data Types :

- TINYINT
- SMALLINT
- INT
- BIGINT
- BOOLEAN
- FLOAT
- DOUBLE
- STRING
- TIMESTAMP
- BINARY

Collection Data Types :

- Struct : like a c struct
- Map : collection of key-value tuples
- Array: arrays for a column

Using collections in fields may appear wrong as it violates normalization. But in our case (Big Data), we have huge speed gains due to the minimal “head seeks” we need to find our data.

SQL vs. HiveQL

Feature	SQL	HiveQL
Updates	UPDATE, INSERT, DELETE	INSERT OVERWRITE TABLE (populates whole table or partition)
Transactions	Supported	Not supported
Indexes	Supported	Not supported
Latency	Sub-second	Minutes
Data types	Integral, floating point, fixed point, text and binary strings, temporal	Integral, floating point, boolean, string, array, map, struct
Functions	Hundreds of built-in functions	Dozens of built-in functions
Multitable inserts	Not supported	Supported
Create table as select	Not valid SQL-92, but found in some databases	Supported
Select	SQL-92	Single table or view in the FROM clause. SORT BY for partial ordering. LIMIT to limit number of rows returned. HAVING not supported.
Joins	SQL-92 or variants (join tables in the FROM clause, join condition in the WHERE clause)	Inner joins, outer joins, semi joins, map joins. SQL-92 syntax, with hinting.
Subqueries	In any clause. Correlated or noncorrelated.	Only in the FROM clause. Correlated subqueries not supported
Views	Updatable. Materialized or nonmaterialized.	Read-only. Materialized views not supported
Extension points	User-defined functions. Stored procedures.	User-defined functions. MapReduce scripts.

HiveQL

```
Hive> CREATE DATABASE ecole;

Hive> USE ecole;

Hive> CREATE TABLE students (

  name STRUCT<f:STRING; l:STRING>,
  grade MAP<STRING,INT>,
  age INT);

Hive> select “Big Data”, avg(grade[“Big
  Data”])                  from students where age
  > 23 limit 1;
```

Complex Types

- Hive has three complex types: ARRAY, MAP, and STRUCT.
- ARRAY and MAP are like their respective in Java
- STRUCT is a record type which encapsulates a set of named fields.
- Complex types permit an arbitrary level of nesting.

```
CREATE TABLE complex (
                col1 ARRAY<INT>,
                col2 MAP<STRING, INT>,
                col3 STRUCT<a:STRING, b:INT, c:DOUBLE>
            );
```

- Sample Query:

```
hive> SELECT col1[0], col2['b'], col3.c FROM complex;
1 2 1.0
```

Multitable Insert

```
FROM records2
INSERT OVERWRITE TABLE stations_by_year
      SELECT year, COUNT(DISTINCT station)
      GROUP BY year
INSERT OVERWRITE TABLE records_by_year
      SELECT year, COUNT(1)
      GROUP BY year
INSERT OVERWRITE TABLE good_records_by_year
      SELECT year, COUNT(1)
      WHERE temperature != 9999
      AND (quality = 0 OR quality = 1 OR quality = 4 OR quality = 5 OR quality = 9)
      GROUP BY year;
```

multitable insert is more efficient than multiple INSERT statements, since the source table need only be scanned once to produce the multiple, disjoint outputs.

Querying Data

- **Sorting data in Hive**: ORDER BY clause, but there is a catch. ORDER BY produces a result that is *totally sorted* - sets the number *of reducers to one*, thus *very inefficient for large datasets*.
- When a globally sorted result is not Hive's nonstandard extension, **SORT BY** producing a *sorted file per reducer.*
- to control which reducer a particular row goes to, to perform some subsequent aggregation: **DISTRIBUTE BY**.
- Example: sort the weather dataset by year and temperature - ensure all the rows for a given year are sent to the same reducer:

```
hive> FROM records2
> SELECT year, temperature
> DISTRIBUTE BY year
> SORT BY year ASC, temperature DESC;
1949 111
1949 78
1950 22
1950 0
1950 -11
```

Joins

- hive> **SELECT name,prod_id FROM sales;**
- Joe 2
- Hank 4
- Ali 0
- Eve 3
- Hank 2

- hive> **SELECT prod_id, prod_name FROM things;**
- 2 Tie
- 4 Coat
- 3 Hat
- 1 Scarf

- hive> **SELECT sales.*, things.***
- > **FROM sales JOIN things ON (sales.id = things.id);**
- Joe 2 2 Tie
- Hank 2 2 Tie
- Eve 3 3 Hat
- Hank 4 4 Coat
- This is because HIVE allows only ONE table in the FROM

HIVE - QL

- CREATE TABLE records (year STRING, temperature INT, quality INT)
- ROW FORMAT DELIMITED
- FIELDS TERMINATED BY '\t';
- ----------
- LOAD DATA LOCAL INPATH 'input/ncdc/micro-tab/sample.txt'
- OVERWRITE INTO TABLE records;
- ----------
- % ls /user/hive/warehouse/record/
- sample.txt

HIVE - QL

- `hive> SELECT year, MAX(temperature)`
- `> FROM records`
- `> WHERE temperature != 9999`
- `> AND (quality = 0 OR quality = 1 OR quality = 4 OR quality = 5 OR quality = 9)`
- `> GROUP BY year;`
- Hive transforms this query into a MapReduce job, then prints the results to the console:
- `1949 111`
- `1950 22`
- `----------`

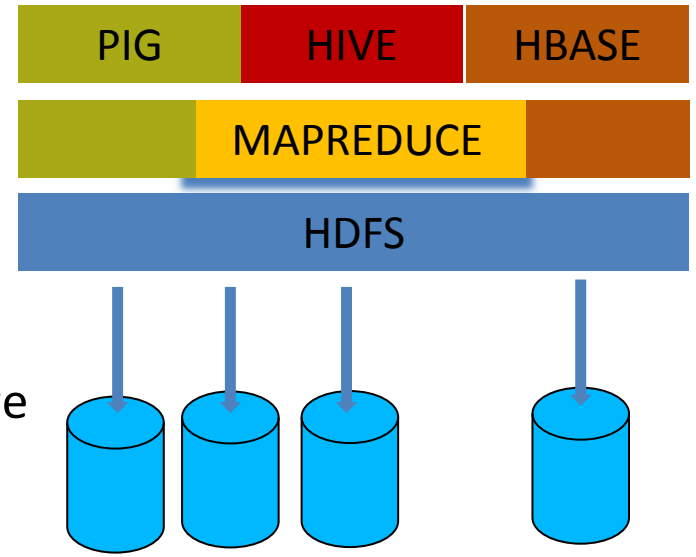
Alternative approaches

HIVE is easy to learn and use but not suited for *Big Data real time applications*.

- **Pig**
  - suitable for various consecutive transformations to input data to produce a new set of output data.
  - *data flow language* rather than a query language

- much easier to think of transformations as a flow instead of queries.

- **Hbase**
  provides features that Hive doesn't:
  - row-level updates,
  - rapid query times and transactions.
  - It is also distributed.
  - **But** it doesn't offer an SQL like query language



References and useful links

- “Hadoop the Definitive Guide”, T. White, O-Reilly media
- The architecture of HDFS is described in “The Hadoop Distributed File System” by Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler (Proceedings of MSST2010, May 2010, *http://storageconference.org/2010/Papers/MSST/Shvachko.pdf*).
- “Programming Hive, Data Warehouse and Query Language for Hadoop”, O'Reilly Media
- https://cwiki.apache.org/Hive/home.html
- http://hbase.apache.org/
- http://pig.apache.org/

ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

Τέλος Ενότητας # 7

Μάθημα: Εξόρυξη γνώσης από Βάσεις Δεδομένων και τον Παγκόσμιο Ιστό, **Ενότητα # 7:** Introduction to Big Data

Διδάσκων: Μιχάλης Βαζιργιάννης**, Τμήμα:** Προπτυχιακό Πρόγραμμα Σπουδών "Πληροφορικής"

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ & ΘΡΗΣΚΕΥΜΑΤΩΝ, ΠΟΛΙΤΙΣΜΟΥ & ΑΘΛΗΤΙΣΜΟΥ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ

ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης