Databases and Big Data Today

CS634
Class 22

Current types of Databases

- SQL using relational tables: still very important!
- NoSQL, i.e., not using relational tables: term “NoSQL” popular since about 2007. May have SQL layered on top.
  - Key-Value Stores
    - Dictionary or “Hash”: key selects “value”, which can have multiple fields, but data not typed, or homogeneous, and contains no links to other data.
  - Document Stores
    - Document here means semi-structured data, in XML, JSON, BSON (binary JSON), or YAML. Each document has a unique id and is self-contained, so no links to parts.
  - Graph Databases
    - Have nodes and edges, and both can have properties, so supports linkage.
  - Wide Column Stores
    - Have tables, but not full relational setup

Important NoSQL Systems

Key-Value Stores

- Redis: in-memory data with journal (log), can do transactions with careful programming
- Memcached: cache in that it drops data to stay within memory bound

Document Stores

- Mongodb: stores BSON, supports types, provides atomic writes, very difficult multi-document transactions now, but better system promised for summer
- Couchdb, others, less popular but in wide use

Graph Databases

- Neo4J the predominant system, supports 9 datatypes, transactions

Wide Column Stores

- Apache HBase: Part of Hadoop project, uses Hadoop's distributed filesystem (HDFS) for data (typically in a datalake)
- Apache Cassandra: also big data related. Not part of Hadoop project but supports Hadoop jobs
- Note: Hadoop is not a database, but rather is "a framework that allows for the distributed processing of large data sets across clusters of computers", i.e., the user has to program the processing rather than use a query language

Hbase queries: from cloudera docs

Scan all rows of table 't1'

hbase> scan 't1'

Specify a startrow, limit the result to 10 rows, and only return selected columns

hbase> scan 't1', {COLUMNS => ['c1', 'c2'], LIMIT => 10, STARTROW => 'xyz'}

Specify a timerange

hbase> scan 't1', {TIMERANGE => [1303668804, 1303668904]}

Specify a custom filter

hbase> scan 't1', {FILTER => org.apache.hadoop.hbase.filter.ColumnPaginationFilter.new(1, 0)}

Apache Hadoop

- Scalable fault-tolerant distributed system for Big Data:
  - Data Storage
  - Data Processing
  - Borrowed concepts/Ideas from Google; Open source under the Apache license
- Core Hadoop has two main systems:
  - **Hadoop/MapReduce**: distributed big data processing infrastructure (abstract/paradigm, fault-tolerant, schedule, execution)
  - **HDFS (Hadoop Distributed File System)**: fault-tolerant, high-bandwidth, high availability distributed storage
- More recently (since 2014): **Apache Spark** on Hadoop/HDFS and directly on HDFS ("standalone")
  - Allows more flexibility in programming than MapReduce
  - Can use memory more effectively, so can be much faster on some tasks
  - Originally developed (2011+) at the University of California, Berkeley's AMPLab, the Spark codebase was at this point donated to Apache (open source).
  - Spark supports Scala, Java, Python, and R.

Example: word counts

Millions of documents in
Word counts out:

```
brown, 2
fox, 2
how, 1
now, 1
the, 3 …
```

In practice, before MapReduce/Spark and related technologies:
The first 10 computers are easy;
The first 100 computers are hard;
The first 1000 computers are impossible;

But now with MapReduce and Spark, data scientists often use 10000 computers!

What's wrong with 1000 computers?

Some will crash while you're working…

If probability of crash = .001
Then probability of all up = $(1-.001)^{1000} = 0.37$

MapReduce and Spark systems expect crashes, tracks partial work, keep going

Typical Large-Data Problem

- Iterate over a large number of records
- Extract something of interest from each

Map

- Shuffle and sort intermediate results
- Aggregate intermediate results
- Generate final output

Reduce

Key idea: provide a functional abstraction for these two operations

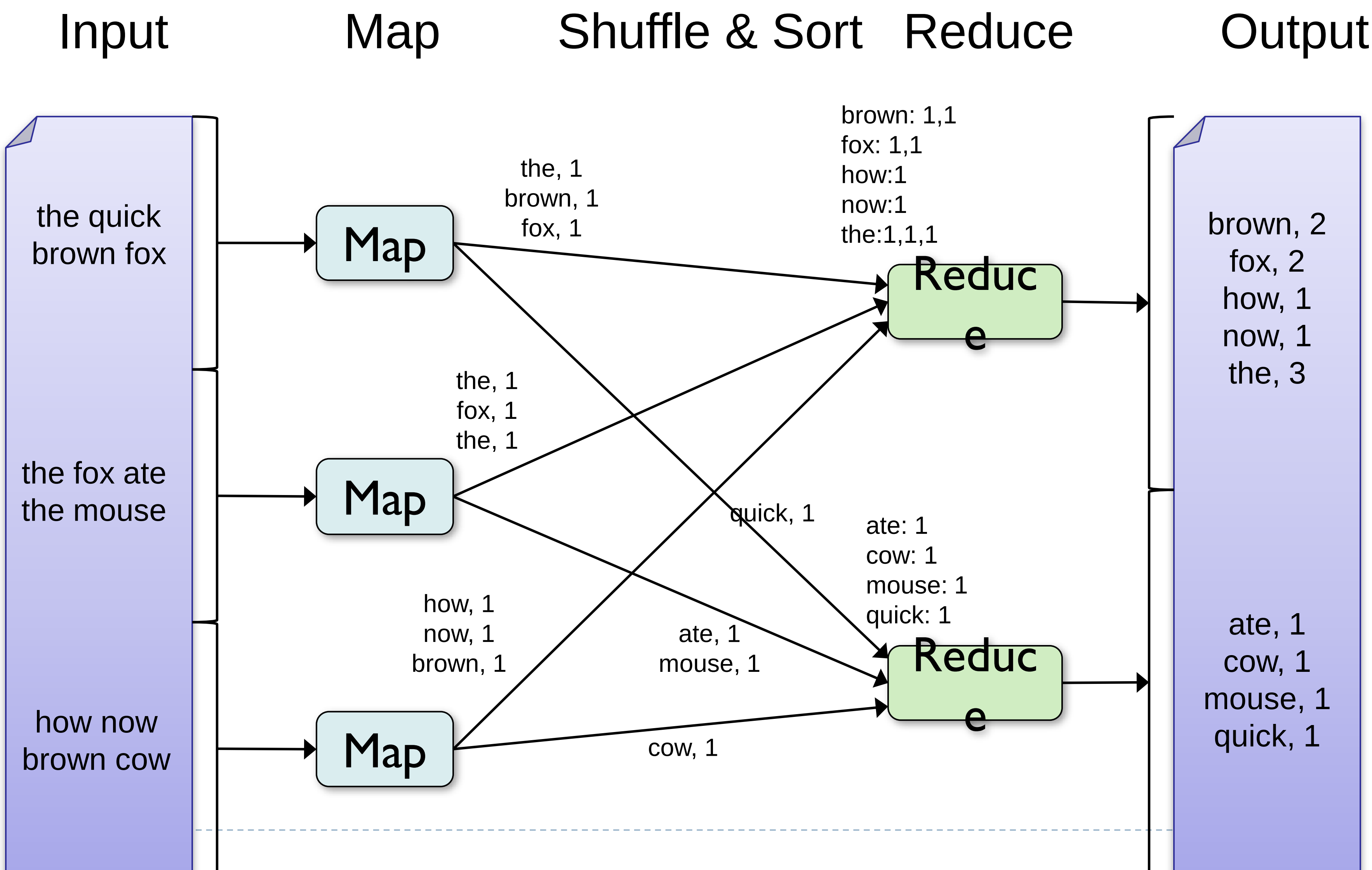
MapReduce and Spark

- MapReduce programmers specify two functions:
  map (k, v) → [(k', v')]
  reduce (k', [v']) → [(k', v'')] or simpler
  - All values with the same key (k') are sent to the same reducer, in k' order for each reducer
  - Here [] means a sequence
- The execution framework handles everything else…
- Spark: has map, reduce as operations, plus others.
- Spark program (words.scala in Scala) for wordcount, from Wikipedia's Spark page (assuming vars conf and sc are already set up)

```
val data = sc.textFile("gs://…some file")
val tokens = data.flatMap(_.split(" "))
val wordFreq = tokens.map((_, 1)).reduceByKey(_ + _)
  wordFreq.sortBy(s => -s._2).map(x => (x._2, x._1)).top(10)
```

- See "map" and "reduceByKey" here, so this Spark program is just using map/reduce programming.

Word Count Execution



Google Cloud has ProcData (under BigData)

- Spin up a Hadoop cluster in 2 minutes!
- Just as easy as creating a VM (easier, because you already have the billing account set up)
- Look in Home>BigData>Procdata
- Can try out Spark on Hadoop.
- See https://cloud.google.com/dataproc/docs/quickstarts/quickstart-console

Running words.scala on Google Procdata

Running words.scala, the Spark word-count program just seen, using a Python hello-world source as input file:

```
eoneil@cluster-eon-m:~$ spark-shell -i words.scala
Loading words.scala...
res0: Array[(Int, String)] = Array((3,=), (1,words),
(1,sorted(rdd.collect())), (1,sc.parallelize(['Hello,',), (1,sc),
(1,rdd), (1,pyspark.SparkContext()), (1,pyspark),
(1,print(words)), (1,import))  ← the results
```

The input file: see 3 "=" words, etc.:

```
#!/usr/bin/python
import pyspark
sc = pyspark.SparkContext()
rdd = sc.parallelize(['Hello,', 'world!'])
words = sorted(rdd.collect())
print(words)
```

Running this python file: Use "submit-spark hello-world.py" or paste into an interactive session started with "pyspark".

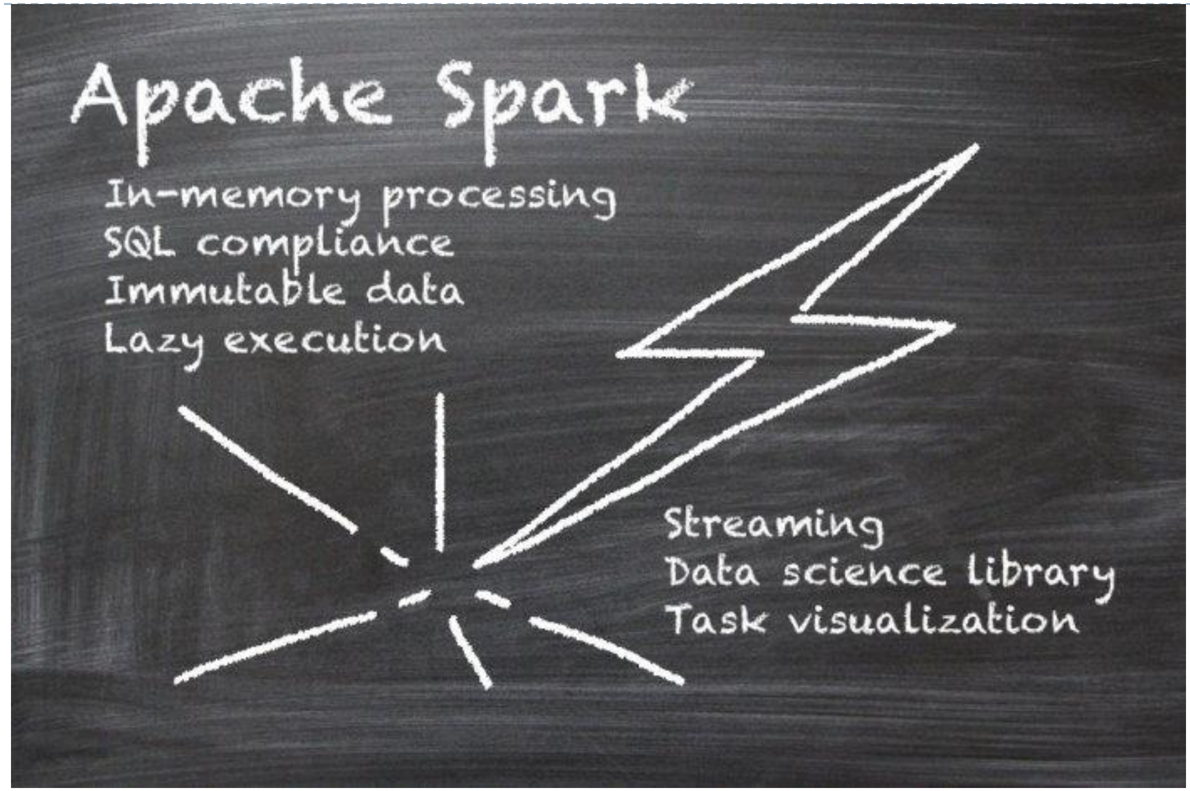
Spark can access RDBs too

Sqltest.scala, using JDBC to access table in my VM's mysql:

```
import org.apache.spark.sql.SQLContext
val url = "jdbc:mysql://10.142.0.2:3306/firstdb" // JDBC URL  sqlContext =
new org.apache.spark.sql.SQLContext(sc)
val df = sqlContext.read.format("jdbc").     // DataFrame object
  option("url", url).option("user", "user").
  option("password","pass123").option("dbtable", "Persons").
  load()
val countsByCity = df.groupBy("City").count()
countsByCity.show
+------------+-----+
|        City|count|
+------------+-----+
|Johannesburg|    1|
+------------+-----+
```

- Spark's ability to access both unstructured data from the data lake and structured data from the RDBs make it a powerful tool
- Tutorial on Spark SQL
- It can access its data using SQL 2003, a more complete SQL than mysql has.

From Infoworld Article (Oct., 2017)



Initially open-sourced in 2012 and followed by its first stable release two years later, Apache Spark quickly became a prominent player in the big data space. Since then, its adoption by big data companies has been on the rise at an eye-catching rate.