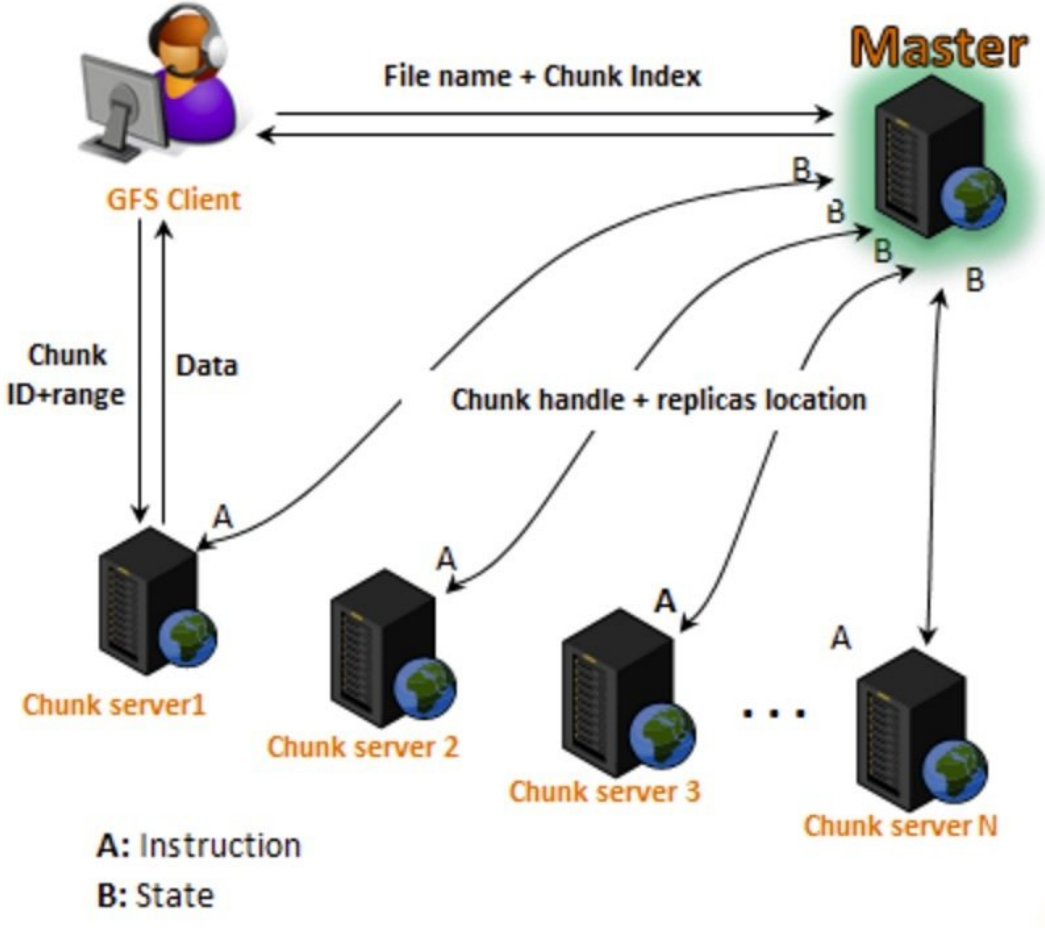
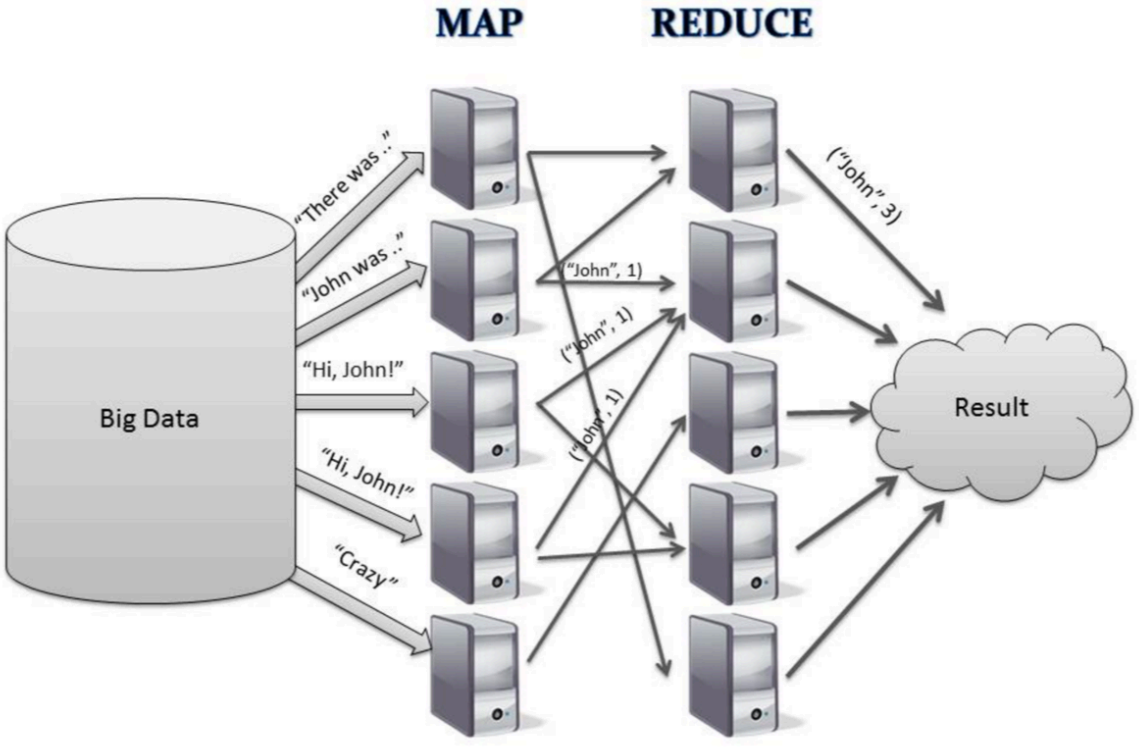
Spark Intro

A short history
of map-reduce systems

Google File System 2003



Google MapReduce 2004



Apache Hadoop, 2006 ---

- An open-source implementation of GFS+MapReduce
- File System: GFS -> HDFS
- Compute system: Google MapReduce -> Hadoop MapReduce
- Large eco-system: Apache Pig, Apache Hive, Apache HBase, Apache Phoenix, Apache Spark, Apache ZooKeeper, Cloudera Impala, Apache Flume, Apache Sqoop, Apache Oozie, Apache Storm.

Apache Spark 2014

- Matei Zaharia, MPLab, Berkeley (Now prof in MIT)
- Main difference from Hadoop: distributed memory instead of distributed files.

Spark, java, scala & python

- The native language of the Hadoop eco-system is Java
- Spark can be programmed in java, but code tends to be long.
- **Scala** allows the parallel programming to be abstracted. It is the core language for Spark.
  - The main problem is that it has a small user base.
- **PySpark** is an extension of Python for programming spark.
  - Does not always achieve the same efficiencies, but is much easier to learn.
  - We will use pyspark

Spark Architecture

SC and RDD

Spark Context

- The pyspark program runs on the main node.
- Control of other nodes is achieved through a special object called the **SparkContext (sc)**. A program needs only one sc.
- Initialization: **sc=SparkContext()**, use parameters for non-default configuration.
- When using pyspark through a jupyter notebook, the kernel will create an **sc** object for you when you run.
  - You can check this by typing **sc** in the first cell.

Resilient Distributed Dataset (RDD)

- A list whose elements are distributed over several computers.
- The main data structure in Spark.
- When in RDD form, the elements of the list can be manipulated only through RDD specific methods.
- RDDs can are created from a list on the master node or from a file.
- RDDs can be translated back to a local list using "collect()"

Pyspark

Some basic examples

Basic example

```
## Initialize an RDD
RDD=sc.parallelize([0,1,2])
## sum the squares of the items
RDD.map(lambda x:x*x)\
   .reduce(lambda x,y:x+y)
## 5
## = 0*0+1*1+2*2
```

Reduce generates a single item on the master node

RDD to RDD

```
## Initialize an RDD
RDD=sc.parallelize([0,1,2])
## sum the squares of the items
A=RDD.map(lambda x:x*x)
A.collect()
## [0,1,4]
```

- **collect()** Collects all of the items in the RDD into a list in the master.
- If the RDD is large, this can take a long time.

Checking the start of an RDD

```
## Initialize a largish RDD
n=10000
B=sc.parallelize(range(n))

# get the first few elements of an RDD
print 'first element=',B.first()
print 'first 5 elements = ',B.take(5)
# first element= 0
# first 5 elements = [0,1,2,3,4]
```

Sampling an RDD

```
## Initialize a largish RDD
n=10000
B=sc.parallelize(range(n))
## sample about m elements into a new RDD
m=5.
C=B.sample(False,m/n)
C.collect()
# [27, 459, 4681, 5166, 5808, 7132, 9793]
```

- Each run results in a different sample.
- Sample size varies, expected size is 5.
- Result is an RDD, need to collect to list.
- Sampling very useful for machine learning.