Cloud Computing
Map Reduce Paradigm

Themis Palpanas
University of Paris

Data Intensive and Knowledge Oriented Systems

1

- thanks for slides to
  - Christoph Freytag
  - Brian Cooper
  - Mike Franklin
  - Roberto Trasarti

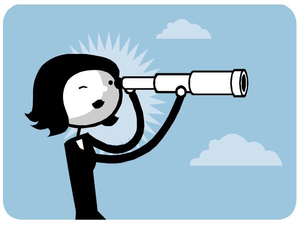
2.2

2

Overview

☐ Cloud Computing

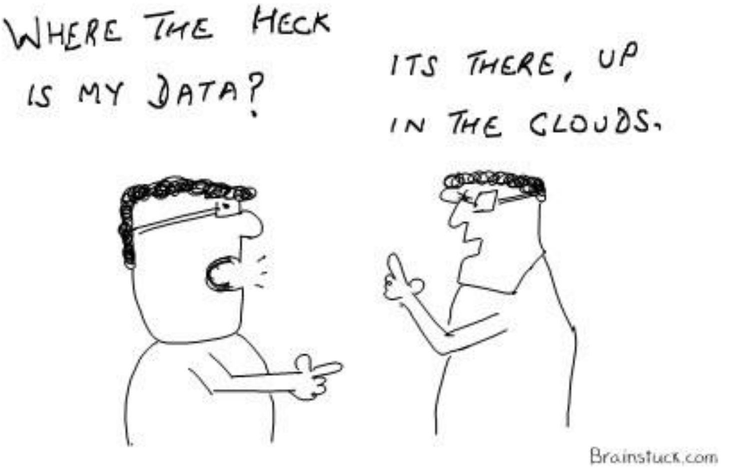
☐ The MapReduce paradigm



5. 3

3

What is Cloud Computing?



5. 4

4

My favorite reference…


Above the Clouds: A Berkeley View of Cloud Computing

Michael Armbrust
Armando Fox
Rean Griffith
Anthony D. Joseph
Randy H. Katz
Andrew Konwinski
Gunho Lee
David A. Patterson
Ariel Rabkin
Ion Stoica
Matei Zaharia

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2009-28
http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html

February 10, 2009

Definition 1: Cloud Computing

- **The illusion of <u>infinite computing resources</u> available on demand,**
  - thereby eliminating the need for Cloud Computing users to plan far ahead for provisioning.
- **The <u>elimination of an up-front commitment</u> by Cloud users**
  - thereby allowing companies to start small and increase hardware resources only when there is an increase in their needs.
- **The ability <u>to pay for use of computing</u> resources on a short-term basis**
  - as needed and release them as needed, thereby rewarding conservation by letting machines and storage go when they are no longer useful.

http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf

Definition 2: Cloud Computing

- ... **is a model for**
  - enabling convenient, on-demand network access
  - to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services)
  - that can be rapidly provisioned and released with minimal management effort or service provider interaction.
- ... **promotes**
  - **availability** and
  - is composed of five essential **characteristics,**
  - and four **deployment models**,
  - three **service models**.

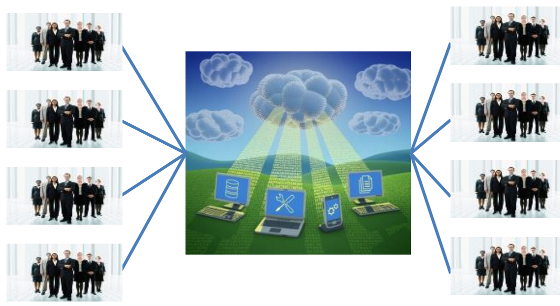
NIST (National Institute of Standards and Technology)

Cloud Usage model (Deployment)

- Private Cloud (Internal Cloud)
  - Organization unit (i.e. company) uses/manages the cloud infrastructure

Cloud Usage model (Deployment)

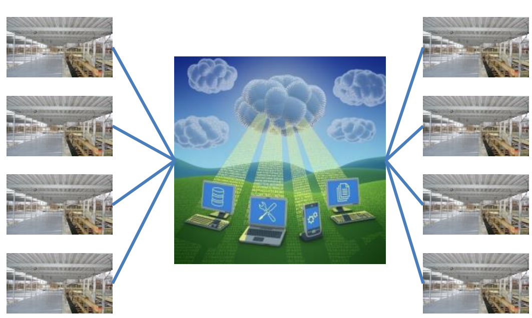
- Community Cloud (External Cloud)
  - Common Infrastructure for specific community with similar or the same needs
  - Example: BioScience/BioInformatics



9

Cloud Usage model (Deployment)

- Public Cloud (Externe Cloud)
  - Offering Cloud Services to customers
  - Example: Google, Amazon, Microsoft, …



10

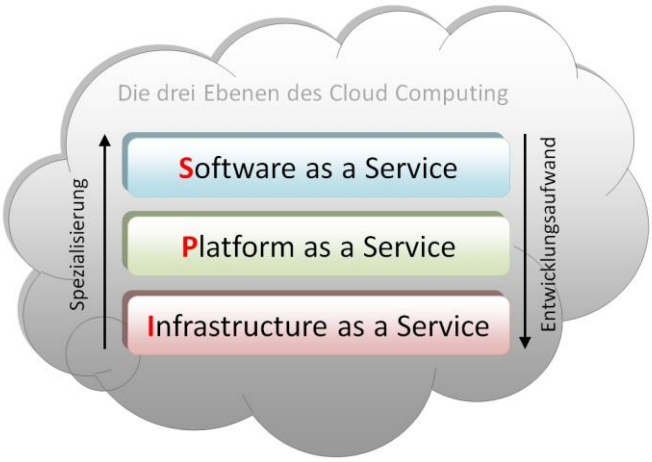
Cloud Usage model (Deployment)

- Hybride Cloud
  - Composition of two or more basic models

What is a Service?

- **A ~~(Web)~~ Service** is …
  - A program with a specified functionality that is accessible over the **Internet**
  - With a well defined interfaces and execution (semantic)
  - **Without knowledge** about its realization
- **Cloud Services** are…
  - … services that are executed over a Cloud platform

Using the Cloud



Cloud Service Model (1)

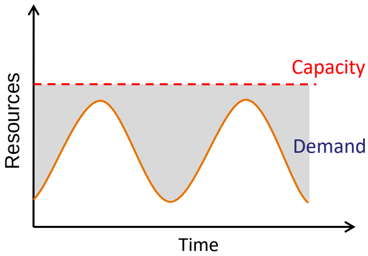
- **Cloud Infrastructure as a Service (IaaS)**
  - Requesting resources such as CPU time, memory, network bandwidth, or other basic resources for pay
- **Cloud Platform as a Service (PaaS)**
  - Developing and executing customer applications on a Cloud infrastructure
- **Cloud Software as a Service (SaaS)**
  - Using installed applications

Service Model - Example

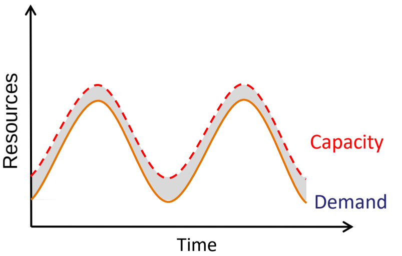
	Amazon	Google	Microsoft	Salesforce	Apple
SaaS	amazon web services Flexible Payment	Google Apps		Salesforce	
PaaS	amazon web services Elastic Beanstalk	Google App Engine	Windows Azure	force.com	
IaaS	amazon web services EC2, S3				iCloud

Economics of Cloud Users

- Pay by use instead of provisioning for peak



Static data center

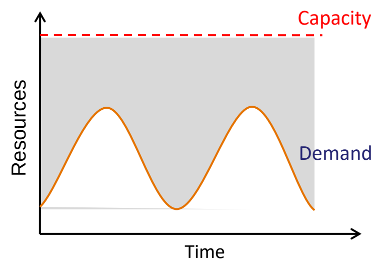


Data center in the cloud

Unused resources

Economics of Cloud Users

- Risk of over-provisioning: underutilization

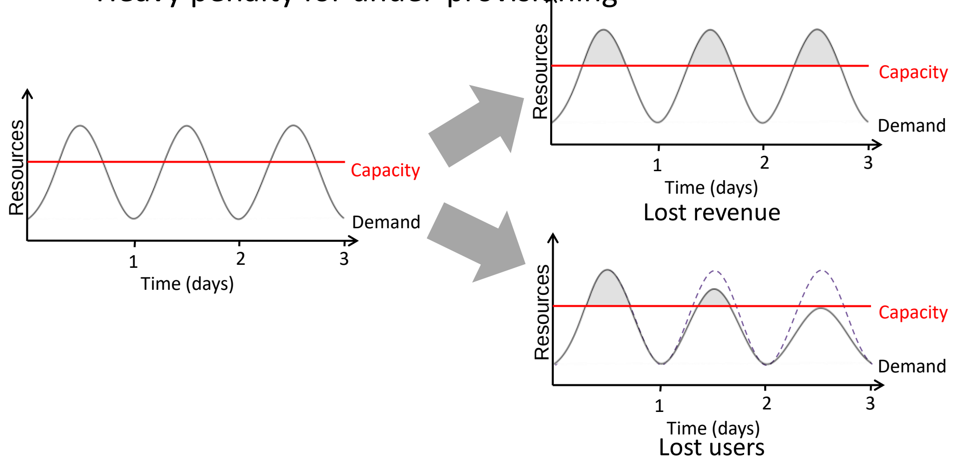


Static data center

Unused resources

Economics of Cloud Users

- Heavy penalty for under-provisioning



Lost revenue

Lost users

Cost Reduction

- „Economy of scale“ – better usage
  - The larger the datacenter (DC), the lower the cost
  - User & Cloud providers both benefit from lower cost

Ressource	Cost for medium size DC	Cost for large DC	Ratio
Network	$95 / Mbps / month	$13 / Mbps / month	~7x
Storage	$2.20 / GB / month	$0.40 / GB / month	~6x
Administration	≈140 Servers/admin	>1000 Servers/admin	~7x

Example Amazon pricing

	USA	Europe	Hongkong & Singapure	Japan	South America
First 10 TB per month	$0,120 / GB	$0,120 / GB	$0,190 / GB	$0,201 / GB	$0,250 / GB
Next 40 TB per month	$0,080 / GB	$0,080 / GB	$0,140 / GB	$0,148 / GB	$0,200 / GB
Next 100 TB per month	$0,060 / GB	$0,060 / GB	$0,120 / GB	$0,127 / GB	$0,180 / GB
Next 350 TB per month	$0,040 / GB	$0,040 / GB	$0,100 / GB	$0,106 / GB	$0,160 / GB
Next 524 TB per month	$0,030 / GB	$0,030 / GB	$0,080 / GB	$0,085 / GB	$0,140 / GB
Next 4 PB per month	$0,025 / GB	$0,025 / GB	$0,070 / GB	$0,075 / GB	$0,130 / GB
Over 5 PB per month	$0,020 / GB	$0,020 / GB	$0,060 / GB	$0,065 / GB	$0,125 / GB

http://aws.amazon.com/en/cloudfront/pricing/

dbis

Example Amazon pricing

	Linux/UNIX	Windows
Small (Standard)	$0,095 per hour	$0,12 per hour
Large	$0,38 per hour	$0,48 per hour
Extra Large	$0,76 per hour	$0,96 per hour
Micro On-Demand Instances		
Micro	$0,025 per hour	$0,035 per hour
High-Memory On-Demand Instances		
Extra Large	$0,57 per hour	$0,62 per hour
Double Extra Large	$1,14 per hour	$1,24 per hour
Quadruple Extra Large	$2,28 per hour	$2,48 per hour
High-CPU On-Demand Instances		
Medium	$0,19 per hour	$0,29 per hour
Extra Large	$0,76 per hour	$1,16 per hour

http://aws.amazon.com/de/ec2/pricing/

dbis

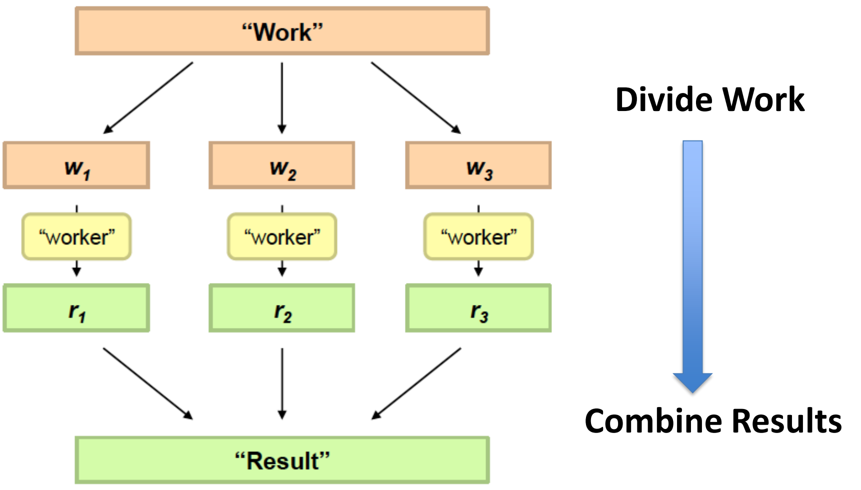
Cloud “killer” applications

- **Batch computation**
  - Washington Post: 200 EC2 instances (1,407 server hours), to analyze 17481 pages of Hillary Clinton‘s travel documents (time: 9 hours)
  - The New York Times uses 100 Amazon EC2 instances + Hadoop to transform 4TB of TIFF pictures into 1.1M pdf files within 24 hours ($240)
- **Scientific computing**
  - Simulations (Weather, Physics …)
  - Evaluation & Analysis
    - Click data
    - Log data from web accesses

- Philosophy to Scale for Big Data?

24

Divide and Conquer



25

Distributed processing is non-trivial

- How to assign tasks to different workers in an efficient way?
- What happens if tasks fail?
- How do workers exchange results?
- How to synchronize distributed tasks allocated to different workers?

Image courtesy of Master isolated images at FreeDigitalPhotos.net

26

Big data storage is challenging

- Data Volumes are massive
- Reliability of Storing PBs of data is challenging
- All kinds of failures: Disk/Hardware/Network Failures
- Probability of failures simply increase with the number of machines …

27

One popular solution: Hadoop

Hadoop Cluster at Yahoo! (Credit: Yahoo)

28

Hadoop offers

- Redundant, Fault-tolerant data storage
- Parallel computation framework
- Job coordination

29

Hadoop offers

- Redundant, Fault-tolerant data storage
- Parallel computation framework
- Job coordination

Programmers

No longer need to worry about

Q: Where file is located?

Q: How to handle failures & data lost?

Q: How to divide computation?

Q: How to program for scaling?

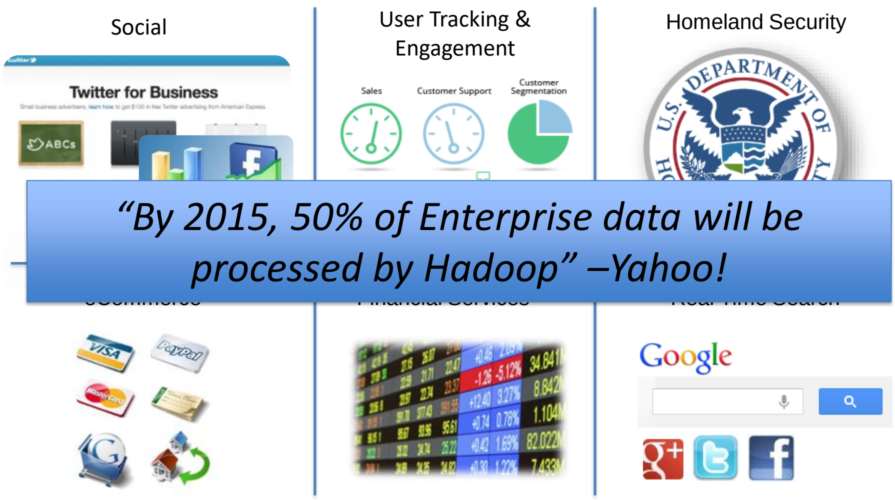
30

A little history on Hadoop

- Hadoop is an open-source implementation based on **Google File System** (GFS) and **MapReduce** from Google
- Hadoop was created by **Doug Cutting** and **Mike Cafarella** in 2005
- Hadoop was donated to **Apache** in 2006

33

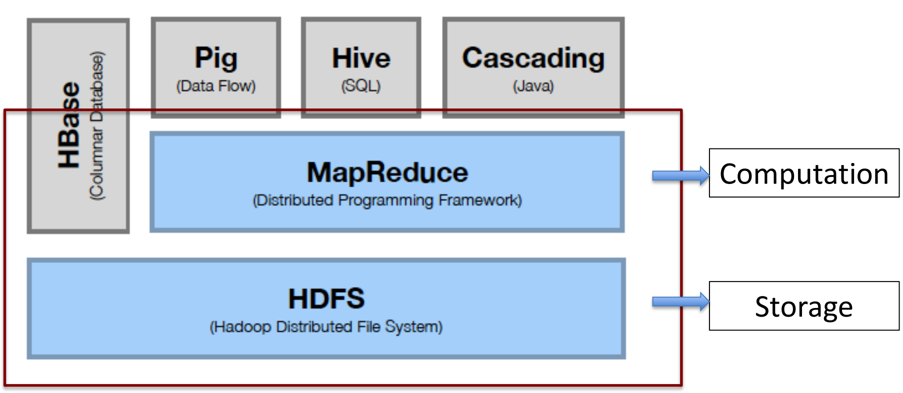
Who are using Hadoop?



34

34

Hadoop Stack



35

Hadoop Resources

- Hadoop at ND:
http://ccl.cse.nd.edu/operations/hadoop/
- Apache Hadoop Documentation:
http://hadoop.apache.org/docs/current/
- Data Intensive Text Processing with Map-Reduce
http://lintool.github.io/MapReduceAlgorithms/
- Hadoop Definitive Guide:
http://www.amazon.com/Hadoop-Definitive-Guide-Tom-White/dp/1449311520

36

HDFS

Hadoop Distributed File System

37

Motivation Questions

- **Problem 1:** Data is too big to store on one machine.

- **HDFS:** Store the data on multiple machines!

38

Motivation Questions

- **Problem 2:** Very high end machines are too expensive

- **HDFS:** Run on commodity hardware!

39

Motivation Questions

- **Problem 3:** Commodity hardware will fail!

- **HDFS:** Software is intelligent enough to handle hardware failure!

40

Motivation Questions

- **Problem 4:** What happens to the data if the machine stores the data fails?

- **HDFS:** Replicate the data!

41

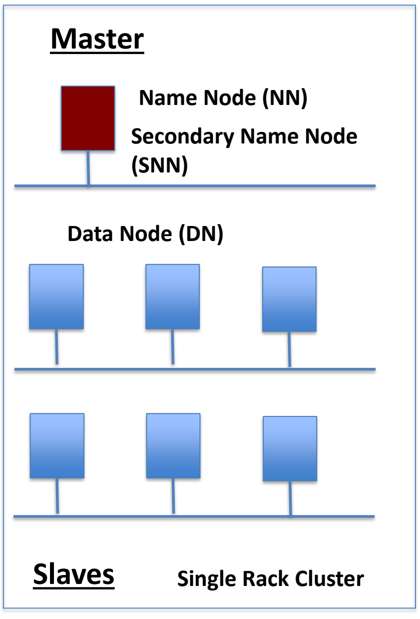
Motivation Questions

- **Problem 5:** How can distributed machines organize the data in a coordinated way?

- **HDFS:** Master-Slave Architecture!

42

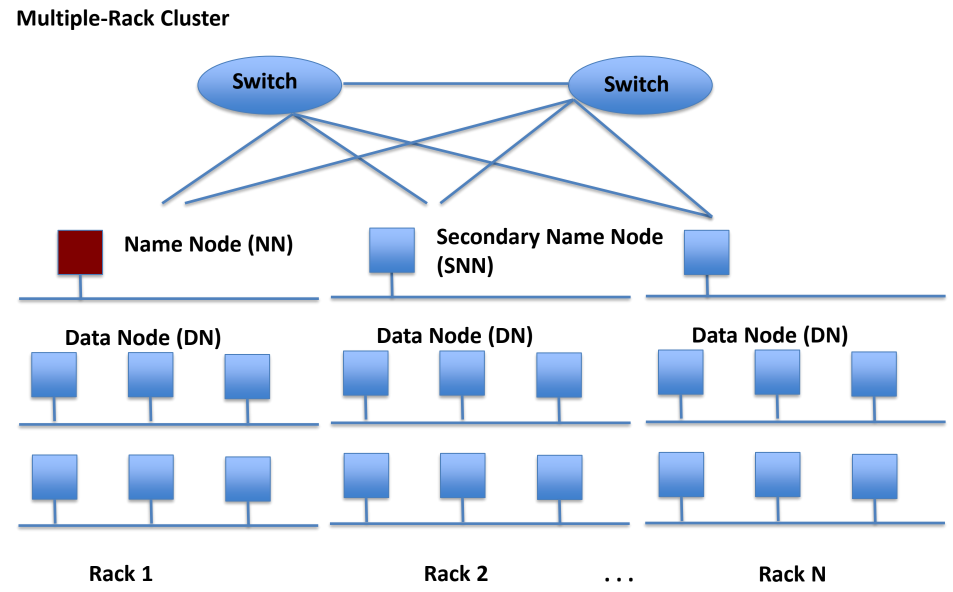
HDFS Architecture: Master-Slave



- Name Node: Controller
  - File System Name Space Management
  - Block Mappings
- Data Node: Work Horses
  - Block Operations
  - Replication
- Secondary Name Node:
  - Checkpoint node

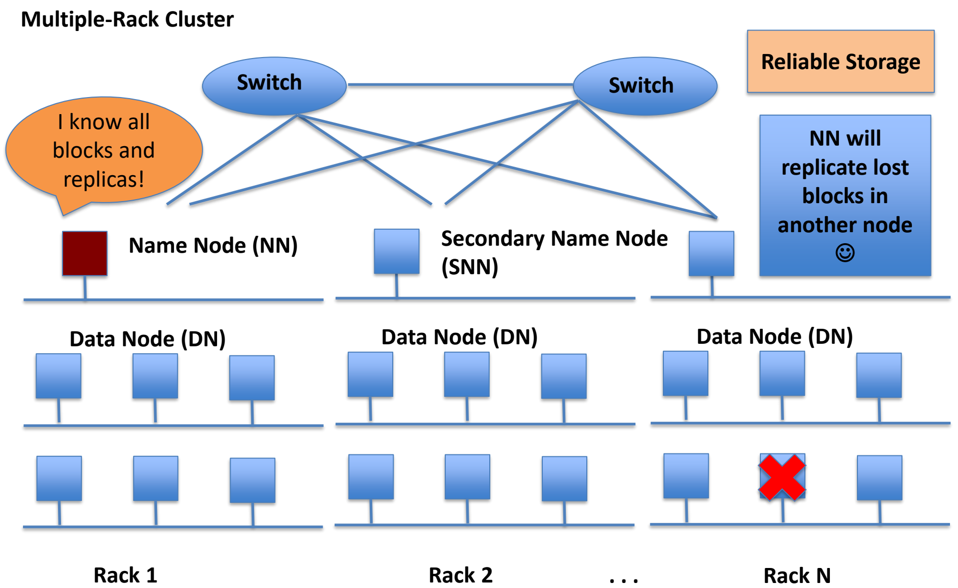
43

HDFS Architecture: Master-Slave



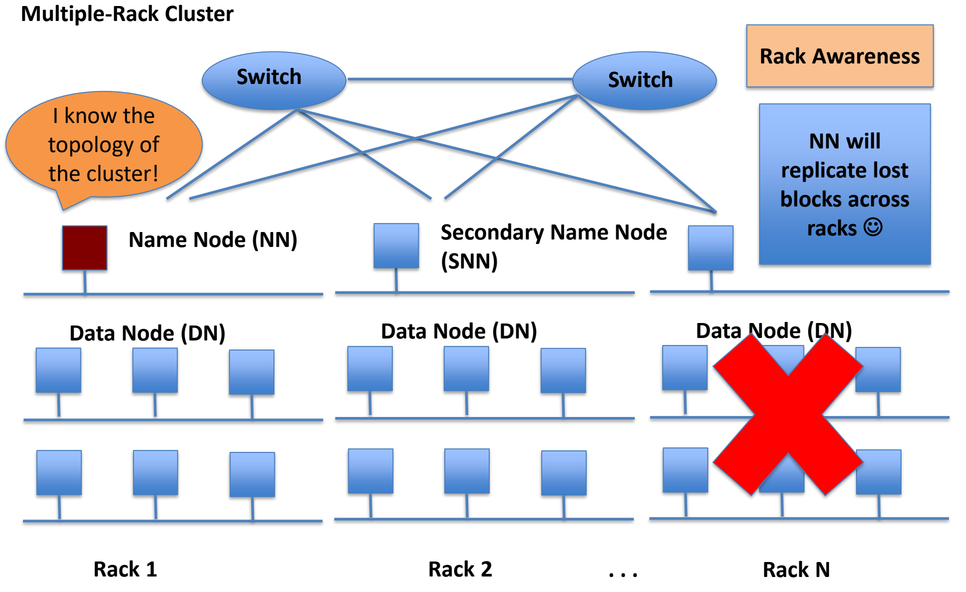
44

HDFS Architecture: Master-Slave



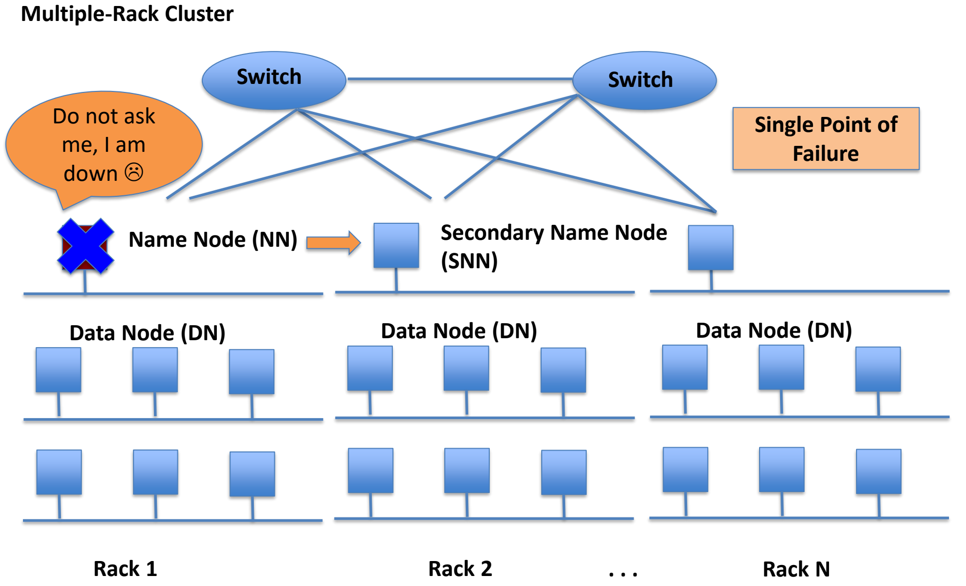
45

HDFS Architecture: Master-Slave



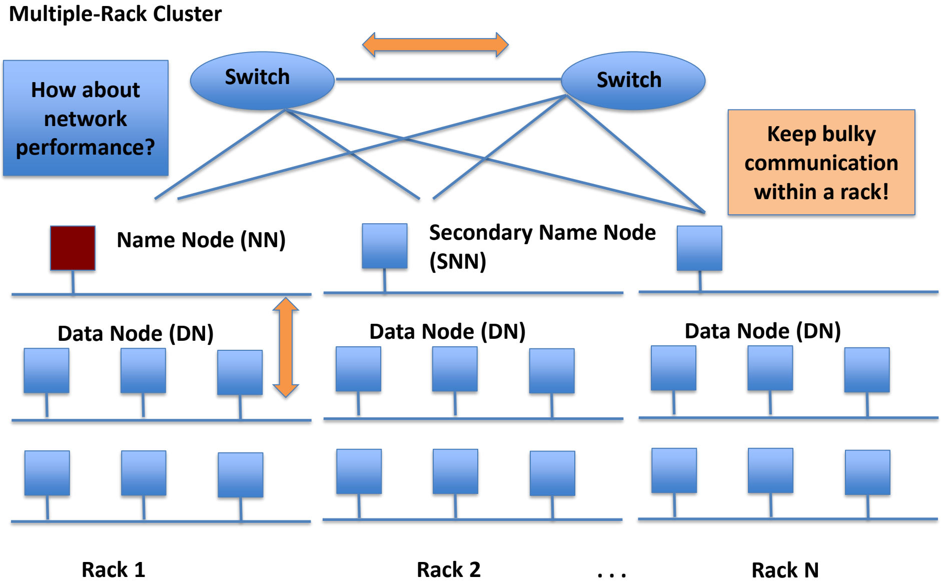
46

HDFS Architecture: Master-Slave



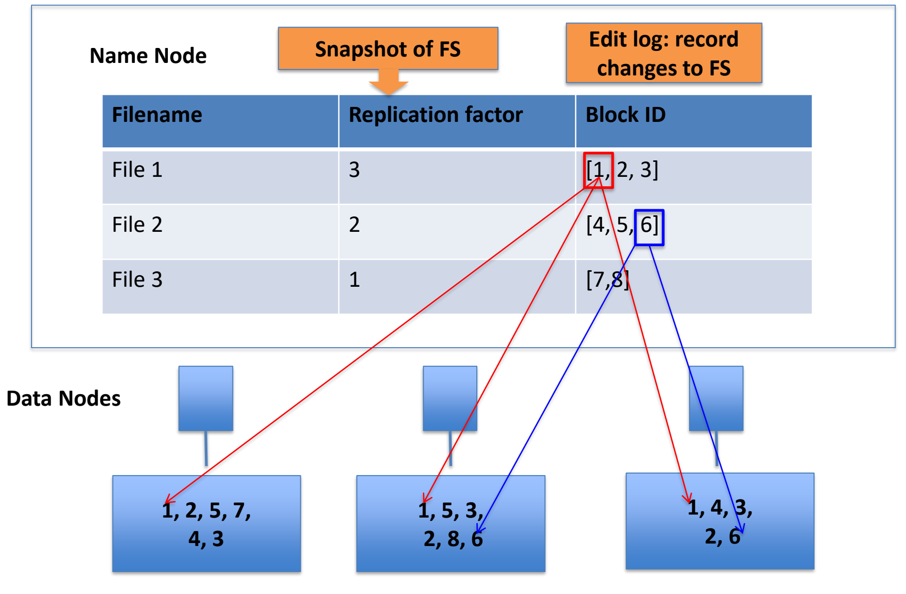
47

HDFS Architecture: Master-Slave



48

HDFS Inside: Name Node



Filename	Replication factor	Block ID
File 1	3	[1, 2, 3]
File 2	2	[4, 5, 6]
File 3	1	[7,8]

49

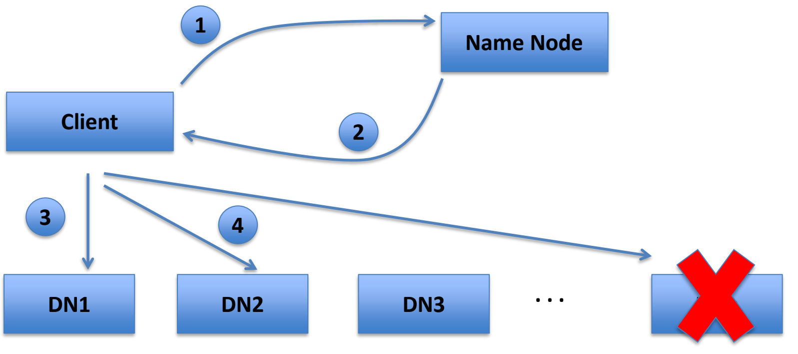
HDFS Inside: Blocks

- Q: Why do we need the abstraction "Blocks" in addition to "Files"?
- Reasons:
  - File can be larger than a single disk
  - Block is of fixed size, easy to manage and manipulate
  - Easy to replicate and do more fine grained load balancing

HDFS Inside: Blocks

- HDFS Block size is by default **64 MB**, why it is much larger than regular file system block?
- Reasons:
  - Minimize overhead: disk seek time is almost constant

HDFS Inside: Read



1. Client connects to NN to read data
2. NN tells client where to find the data blocks
3. Client reads blocks directly from data nodes (without going through NN)
4. In case of node failures, client connects to another node that serves the missing block

HDFS Inside: Read

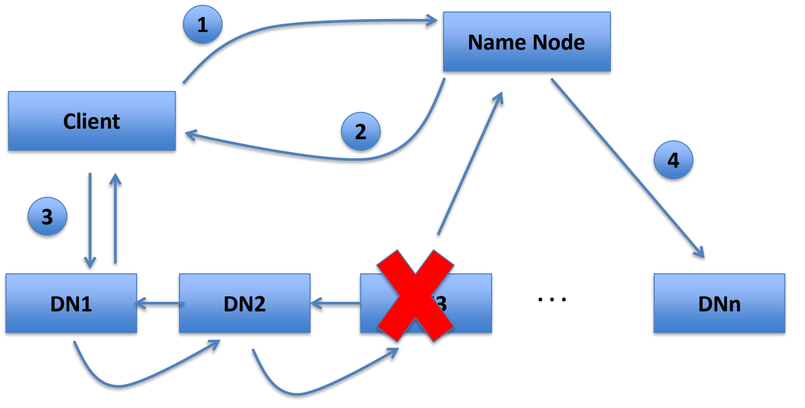
- Q: Why does HDFS choose such a design for read? Why not ask client to read blocks through NN?
- Reasons:
  - Prevent NN from being the bottleneck of the cluster
  - Allow HDFS to scale to large number of concurrent clients
  - Spread the data traffic across the cluster

HDFS Inside: Read

- Q: Given multiple replicas of the same block, how does NN decide which replica the client should read?
- HDFS Solution:
  - Rack awareness based on network topology

54

HDFS Inside: Write



1. Client connects to NN to write data
2. NN tells client write these data nodes
3. Client writes blocks directly to data nodes with desired replication factor
4. In case of node failures, NN will figure it out and replicate the missing blocks

55

HDFS Inside: Write

- Q: Where should HDFS put the three replicas of a block? What tradeoffs we need to consider?
- Tradeoffs:
  - Reliability
  - Write Bandwidth
  - Read Bandwidth

Q: What are some possible strategies?

56

HDFS Inside: Write

- Replication Strategy vs Tradeoffs

	Reliability	Write Bandwidth	Read Bandwidth
Put all replicas on one node	☹	☺	☹
Put all replicas on different racks	☺	☹	☹

57

HDFS Inside: Write

- Replication Strategy vs Tradeoffs

	Reliability	Write Bandwidth	Read Bandwidth
Put all replicas on one node			
Put all replicas on different racks			
HDFS: 1-> same node as client 2-> a node on different rack 3-> a different node on the same rack as 2			

58

HDFS

- Copy file `foo.txt` from local disk to the user's directory in HDFS

```
$ hadoop fs -put foo.txt foo.txt
```

  - This will copy the file to `/user/username/foo.txt`

- Get a directory listing of the user's home directory in HDFS

```
$ hadoop fs -ls
```

- Get a directory listing of the HDFS root directory

```
$ hadoop fs -ls /
```

59

HDFS

- Display the contents of the HDFS file `/user/fred/bar.txt`

```
$ hadoop fs -cat /user/fred/bar.txt
```

- Copy that file to the local disk, named as `baz.txt`

```
$ hadoop fs -get /user/fred/bar.txt baz.txt
```

60

HDFS

- Create a directory called `input` under the user's home directory

```
$ hadoop fs -mkdir input
```

- Delete the directory `input_old` and all its contents

```
$ hadoop fs -rm -r input_old
```

61

The Map Reduce Paradigm

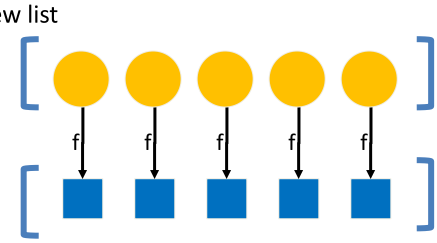
What is MapReduce?

- A programming model and an associated implementation (library) for processing and generating large data sets (on large clusters).
- A new abstraction allowing us to express the simple computations we were trying to perform but hides the messy details of parallelization, fault-tolerance, data distribution and load balancing in a library.
- Initiated by Google:
  - Jeffrey Dean , Sanjay Ghemawat: MapReduce: simplified data processing on large clusters, In OSDI'04
  - Public Domain Version: Hadoop

A short intro to Map/Reduce (Functional Programming)

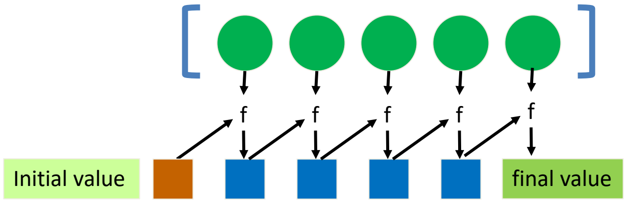
Map

- Map is a higher-order function
  - map: f x list ⇒ list
  - map (f, list) = f(first(list)) • map(f, rest(list))
- How map works:
  - Function is applied to every element in a list
  - Result is a new list



Fold

- Fold (Reduce) is also a higher-order function
  - fold: *f x I x list* ⇒ *value*
  - fold (f, list) = f(first(list), fold (f, rest(list))
- How fold works:
  1. Accumulator set to initial value *I*
  2. Repeated for every item in the list
     1. Function *f* applied to list element and the accumulator
     2. Result stored in the accumulator
  3. Result is the final value in the accumulator



66

Map/Fold in action

- Simple map example:

```
(map square '(1 2 3 4 5)) → '(1 4 9 16 25)
```

- Fold examples:

```
(fold + 0 '(1 2 3 4 5)) → 15
(fold * 1 '(1 2 3 4 5)) → 120
```

- Sum of squares:

```
define (sum-of-squares list) =
             (fold + 0 (map square list))
(sum-of-squares '(1 2 3 4 5)) → 55
```


67

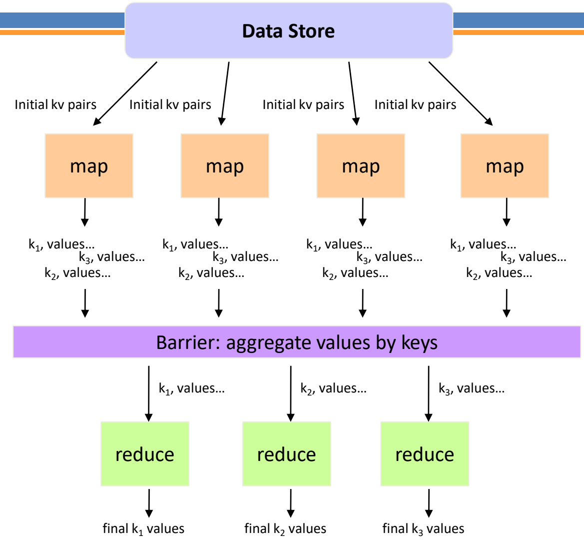
Hadoop:
a MapReduce Implementation

Hadoop… and more

- Open Source Project (http://hadoop.apache.org/)
  - Implements Map/Reduce paradigm
  - Many additional components
    - Hive: SQL like language for expressing aggregate queries & joins
    - Pig: A „data flow“ language describing „atomic“ steps
    - …
  - Introduce only the „core“ engine
    - Major properties
    - Architectural overview

dbis

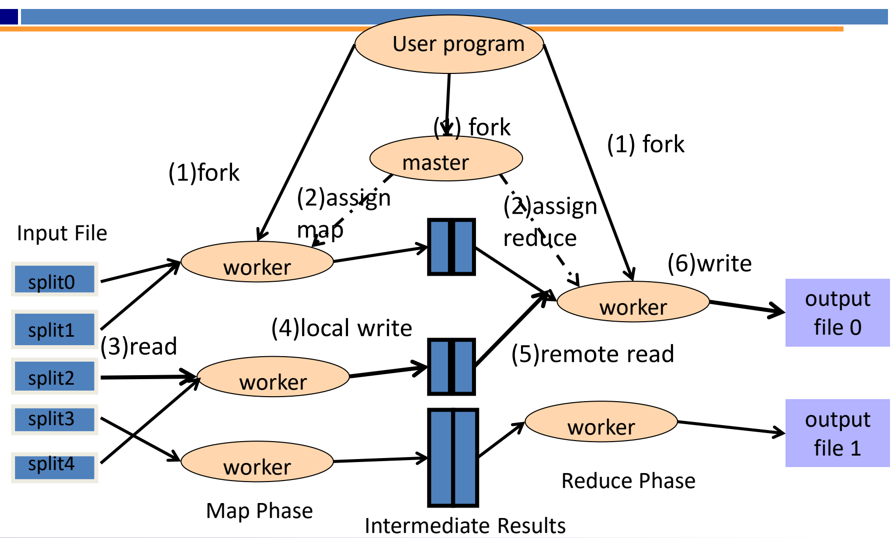
MapReduce: It's just divide and conquer!



70

dbis

Execution overview



71

dbis

Example: Word Counting

map(key=url, val=contents):
For each word *w* in contents, emit (w, "1")

reduce(key=word, values=uniq_counts):
Sum all "1"s in values list
Emit result "(word, sum)"

see bob throw
see spot run

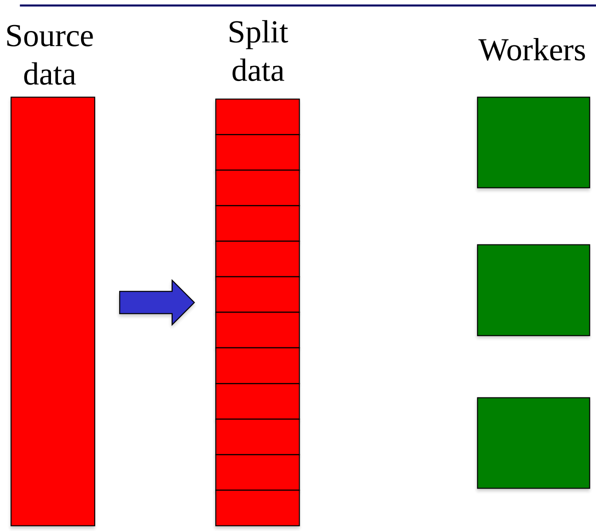
see	1
bob	1
run	1
see	1
spot	1
throw	1

bob	1
run	1
see	2
spot	1
throw	1

5. 72

72

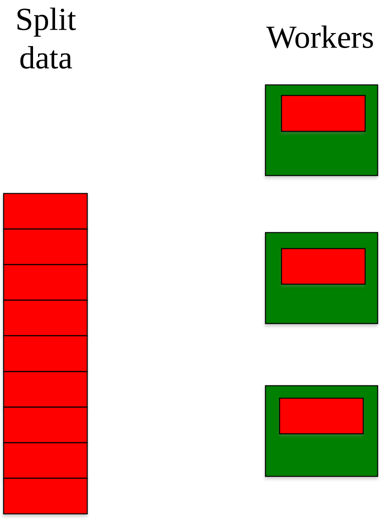
Mappers



73

73

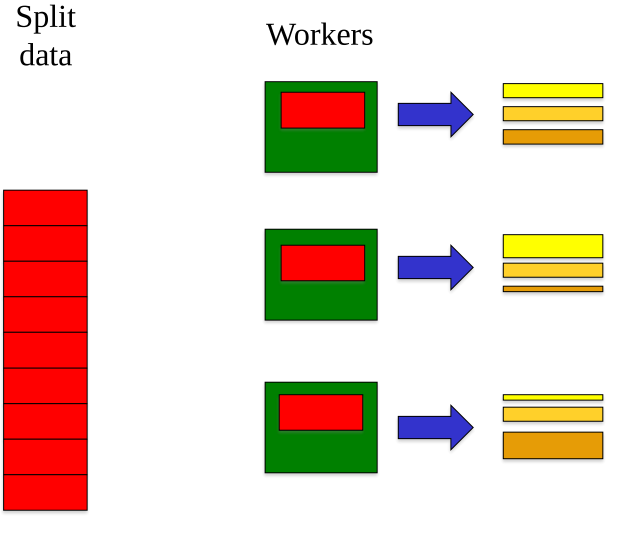
Mappers



74

74

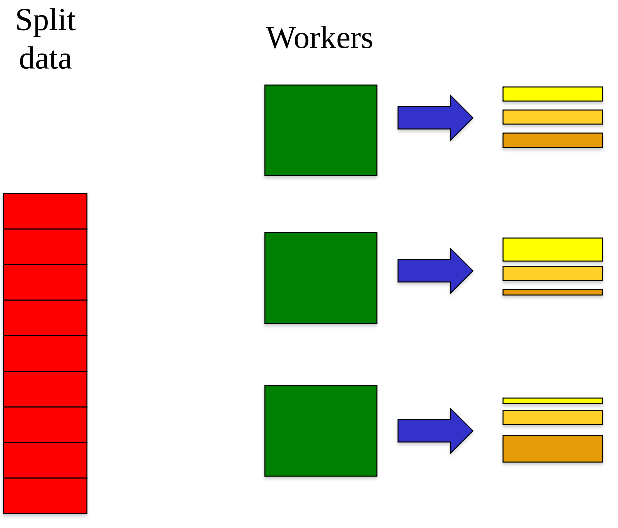
Mappers



75

75

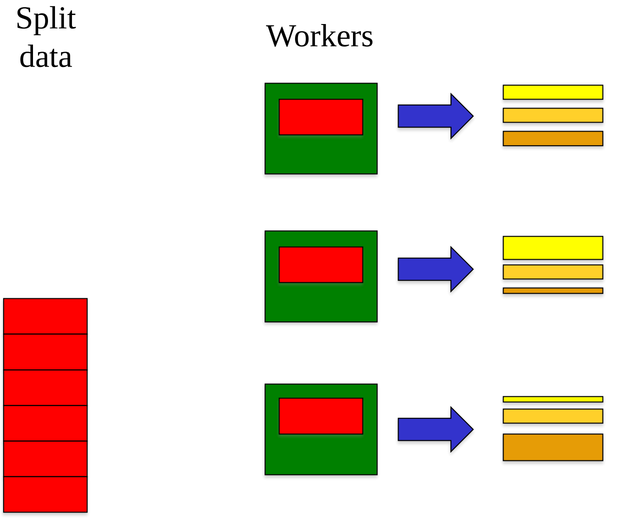
Mappers



76

76

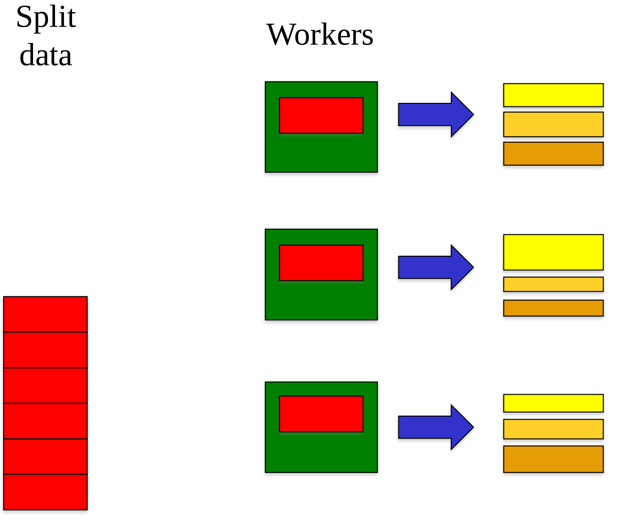
Mappers



77

77

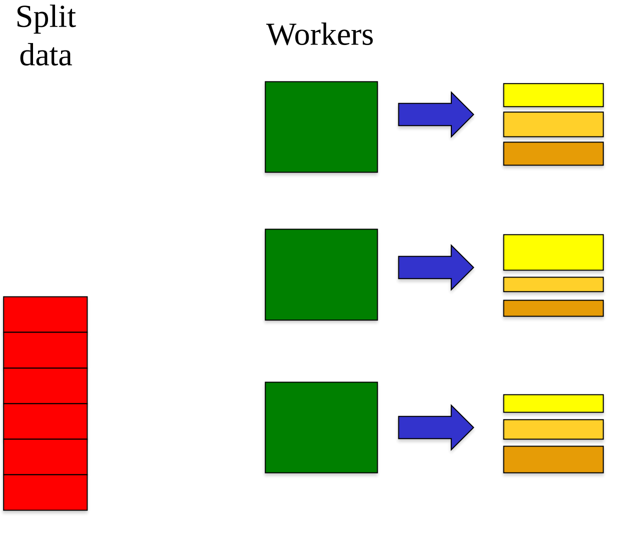
Mappers



78

78

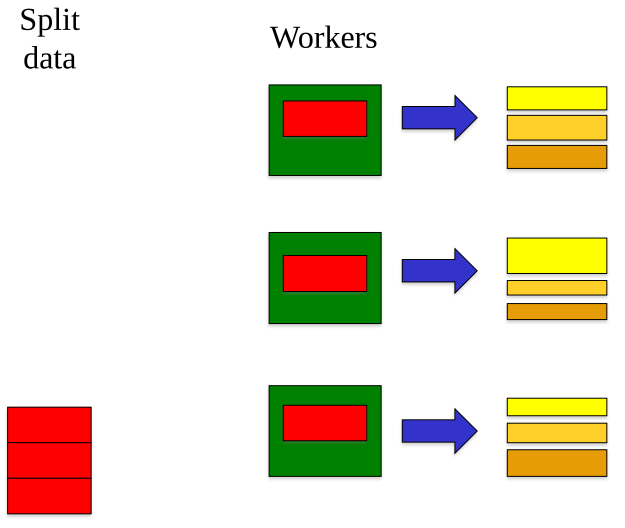
Mappers



79

79

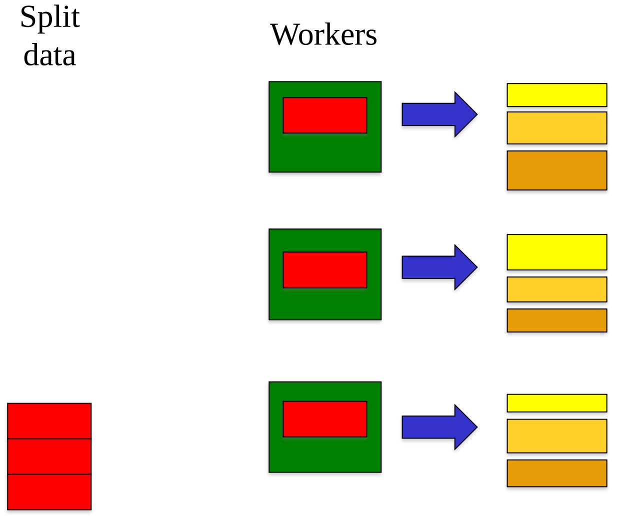
Mappers



80

80

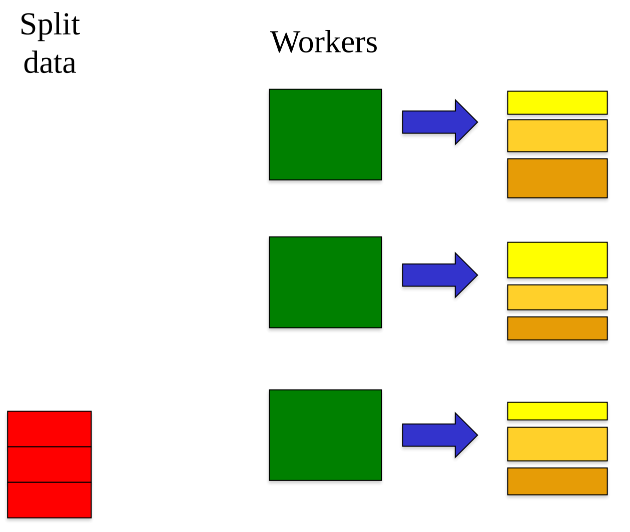
Mappers



81

81

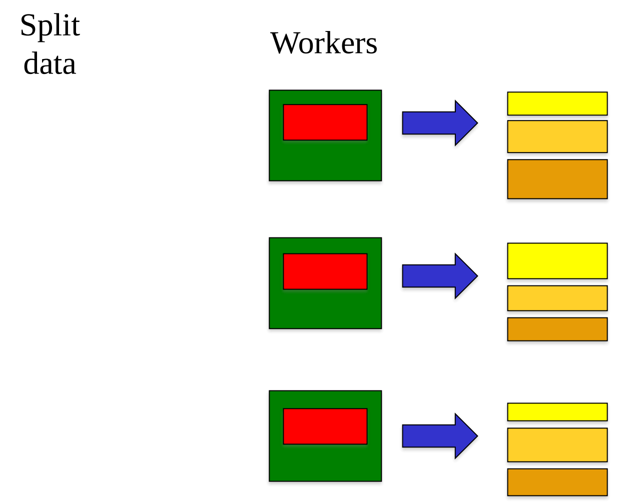
Mappers



82

82

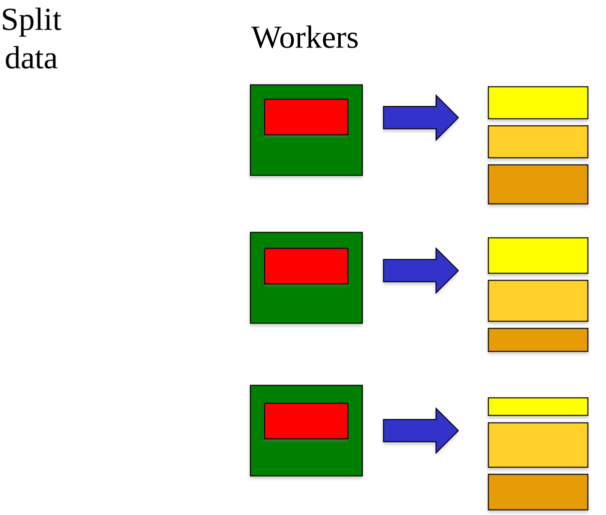
Mappers



83

83

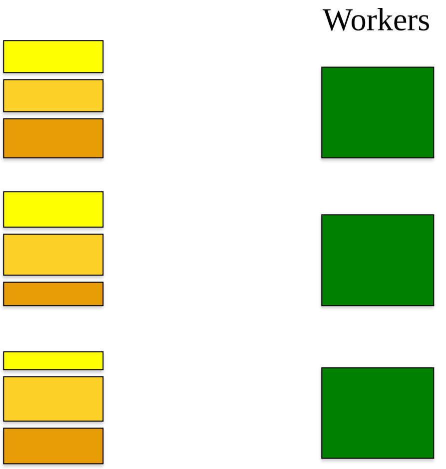
Mappers



84

84

Shuffle

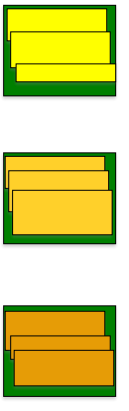


85

85

Shuffle

Workers

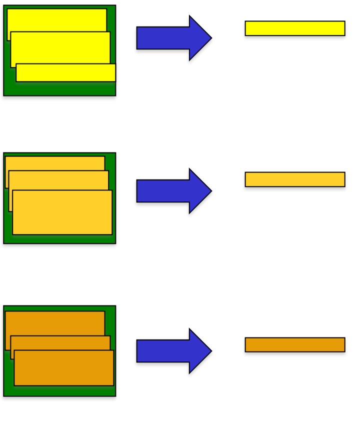


86

86

Reduce

Workers



87

87

dbis

Example: Word counting in documents

```
map(String key, String value):
  // key: document name
  // value: document contents
  for each word w in value:
    EmitIntermediate(w, "1");

reduce(String key, Iterator values):
  // key: a word
  // values: a list of counts
  int result = 0;
  for each v in values:
    result += ParseInt(v);
  Emit(AsString(result));
```


DB Thinking Meets Systems Thinking?

MapReduce: A major step backwards

By David DeWitt on January 17, 2008

[Note: Although the system attributes this post to a single author, it was written by David J. DeWitt and Michael Stonebraker]

Mike Franklin

DB Thinking Meets Systems Thinking?

"MapReduce may be a good idea for writing certain types of general-purpose computations, but to the database community, it is:

1. **A giant step backward** in the programming paradigm for large-scale data intensive applications
2. **A sub-optimal implementation**, in that it uses brute force instead of indexing
3. **Not novel at all** - it represents a specific implementation of well known techniques developed **nearly 25 years ago**
4. **Missing most of the features** that are routinely included in current DBMS
5. **Incompatible with all of the tools** DBMS users have come to depend on"

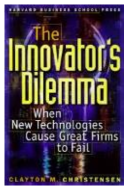
Mike Franklin

99

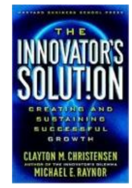
AT THE TIME, MANY IN THE DB CAMP AGREED

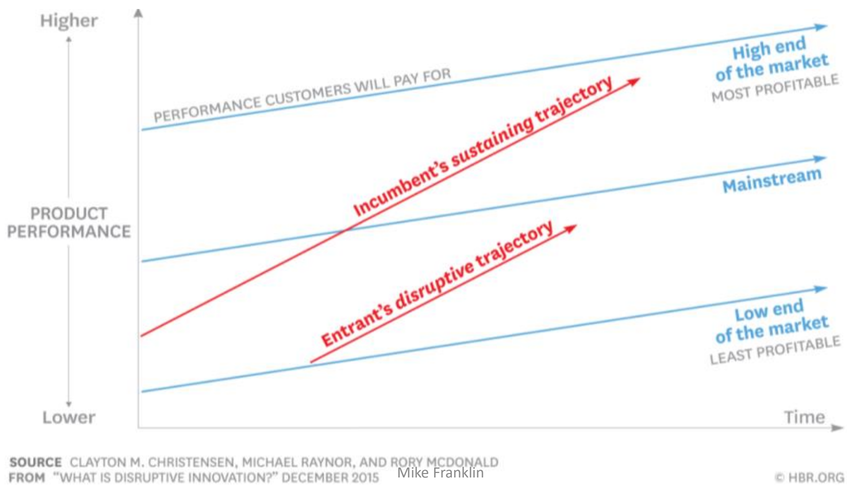
Mike Franklin

100



Disruptive Technology (low end/new market)





Mike Franklin

101

DB Thinking Meets Systems Thinking?

"MapReduce may be a good idea for writing certain types of general-purpose computations, but to the database community, it is:

1. **A giant step backward** in the programming paradigm for large-scale data intensive applications
2. **A sub-optimal implementation**, in that it uses brute force instead of indexing
3. **Not novel at all** - it represents a specific implementation of well known techniques developed **nearly 25 years ago**
4. **Missing most of the features** that are routinely included in current DBMS
5. **Incompatible with all of the tools** DBMS users have come to depend on"

Mike Franklin

102

BUT "DATABASE THINKING" IS DRIVING THE IMPROVEMENT PROCESS

Mike Franklin

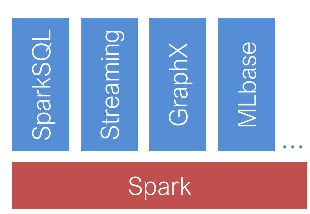
103

Spark's Philosphy

- Specializing MapReduce leads to stovepiped systems
- Instead, **generalize** MapReduce:

  1. Richer Programming Model
     ➔ Fewer Systems to Master

  2. Memory Management
     ➔ Less data movement leads to better performance for complex analytics



Mike Franklin 104

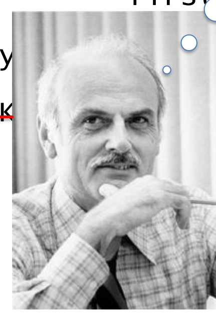
104

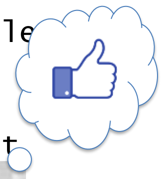
Abstraction: *Dataflow Operators*

- map
- filter
- groupBy
- sort
- union
- join
- leftOuterJoin
- rightOuterJoin

- reduce
- count
- fold
- reduceBy
- groupByK
- cogroup
- cross
- zip

sample
take
first
tionBy
th

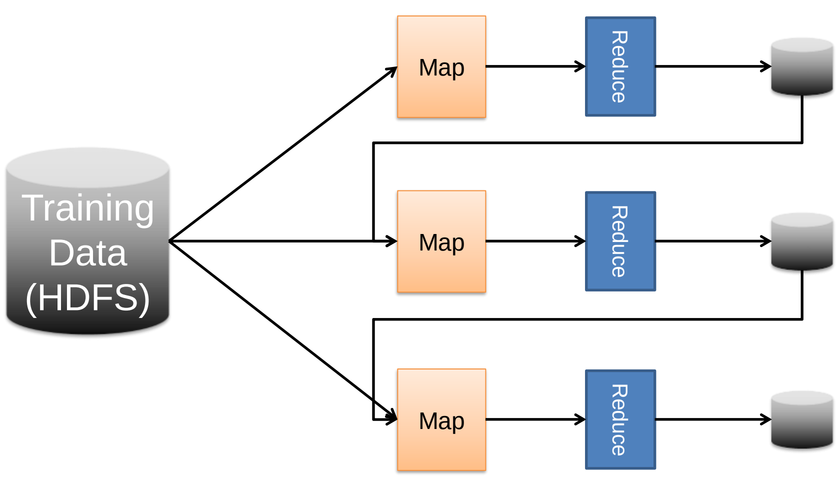




Mike Franklin 105

105

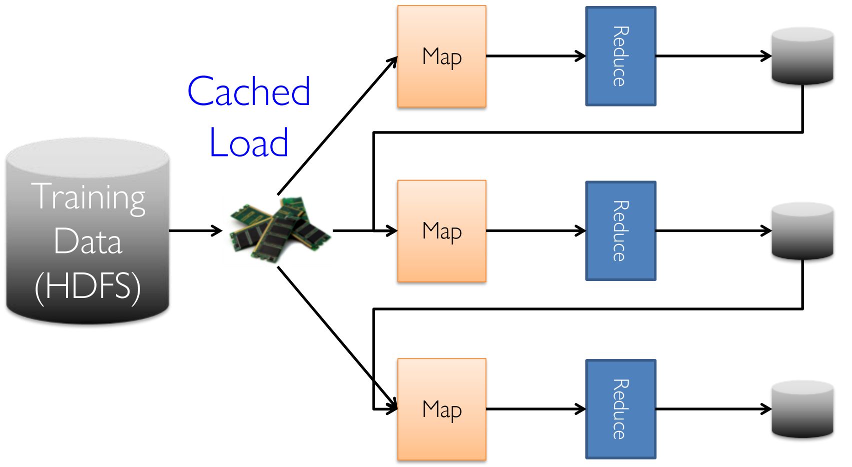
Memory Mgmt in Hadoop MR



Mike Franklin 106

106

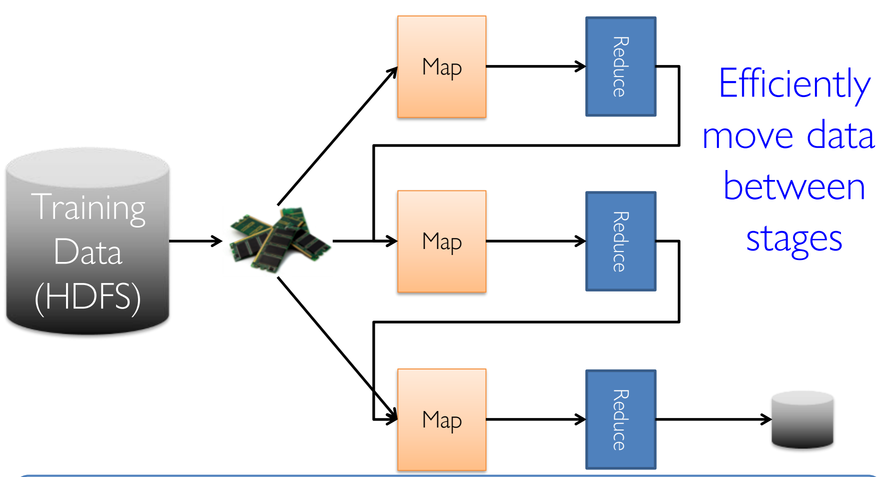
Memory Management in Spark



Mike Franklin 107

107

Memory Management in Spark



10-100× speed up vs.Hadoop MapReduce with no HDFS data migration needed

Mike Franklin 108

108

RESILIENT DISTRIBUTED DATASET (RDD)

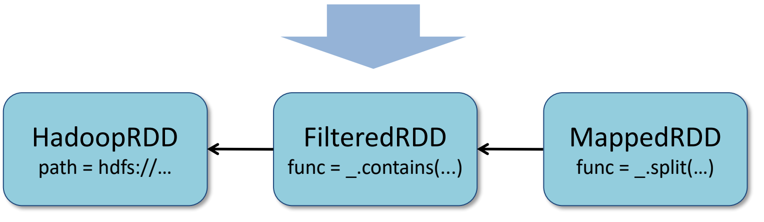
- main Spark ingredient
- Spark supports massive parallel computations based on RDD
- RDD is where you put your data
- an RDD can be viewed as a vector/list of data stored on several machines
- each machine can *access* and *process* part of it
- an RDD is fault tolerant: if a machine fails, the RDD is not "broken"

109

Lineage (aka Logical Logging)

- **RDDs: Immutable** collections of objects that can be stored in memory or disk across a cluster
  – Built via parallel transformations (map, filter, …)
  – Automatically rebuilt on (partial) failure

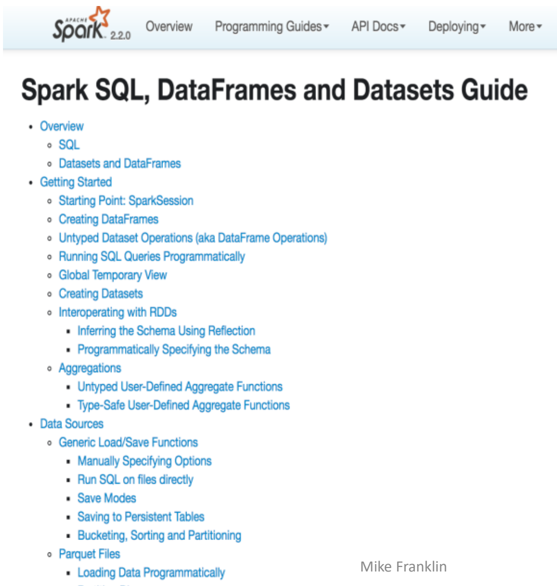
```
messages = textFile(...).filter(_.contains("error"))
                        .map(_.split('\t')(2))
```



M. Zaharia, et al, Resilient Distributed Datasets: A fault-tolerant abstraction for in-memory cluster computing, NSDI 2012.

110

Spark Native SQL Support



Mike Franklin

111

DataFrames
(main abstraction in Spark 2.0)

```
employees
.join(dept, employees("deptId") === dept("id"))
.where(employees("gender") === "female")
.groupBy(dept("id"), dept("name"))
.agg(count("name"))
```

Notes:

1) Some people prefer this to SQL ☺
2) Dataframes can be typed (called "Datasets")

Mike Franklin 112

112

Catalyst Optimizer

- Typical DB optimizations across SQL and Dataframes
  - Extensibility via Optimization Rules written in Scala
  - Open Source optimizer evolution!
- Code generation for inner-loops, iterator removal
- Extensible Data Sources: CSV, Avro, Parquet, JDBC, …

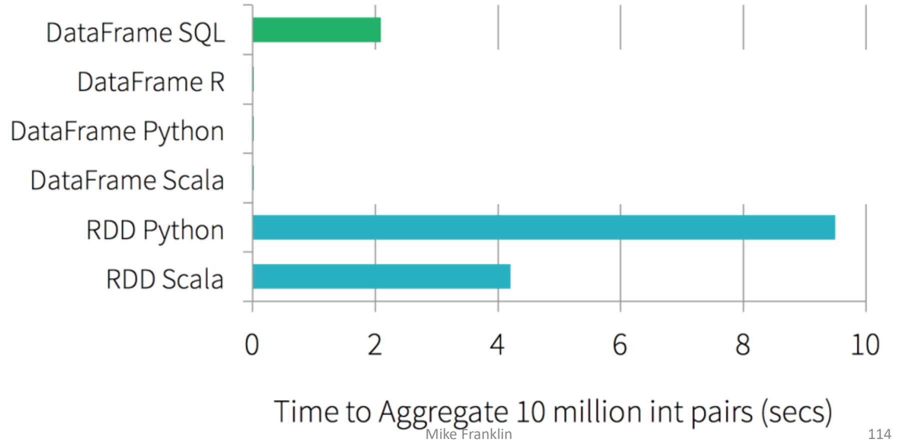
  via TableScan (all cols), PrunedScan (project),
  FilteredPrunedScan(push advisory selects and projects)
  CatalystScan (push advisory full Catalyst expression trees)
- Extensible (User Defined) Types
- Cost-based (as of v2.2)

M. Armbrust, et al, Spark SQL: Relational Data Processing in Spark, SIGMOD 2015.

Mike Franklin 113

113

An interesting thing about SparkSQL Performance



Mike Franklin 114

114

Spark Structured Streams (unified)

Batch Analytics

```
// Read data once from an S3 location
val inputDF = spark.read.json("s3://logs")

// Do operations using the standard DataFrame API and write to MySQL
inputDF.groupBy($"action", window($"time", "1 hour")).count()
       .write.format("jdbc")
       .save("jdbc:mysql//...")
```

Streaming Analytics

```
// Read data continuously from an S3 location
val inputDF = spark.readStream.json("s3://logs")

// Do operations using the standard DataFrame API and write to MySQL
inputDF.groupBy($"action", window($"time", "1 hour")).count()
       .writeStream.format("jdbc")
       .start("jdbc:mysql//...")
```


Putting it all Together: Multi-modal Analytics

SQL

```
// Load historical data as an RDD using Spark SQL
val trainingData = sql(
  "SELECT location, language FROM old_tweets")
```

Machine Learning

```
// Train a K-means model using MLlib
val model = new KMeans()
  .setFeaturesCol("location")
  .setPredictionCol("language")
  .fit(trainingData)
```

Streaming

```
// Apply the model to new tweets in a stream
TwitterUtils.createStream(...)
  .map(tweet => model.predict(tweet.location))
```

Current release has similar support for Deep Learning models as well

dbis

Beyond Map Reduce

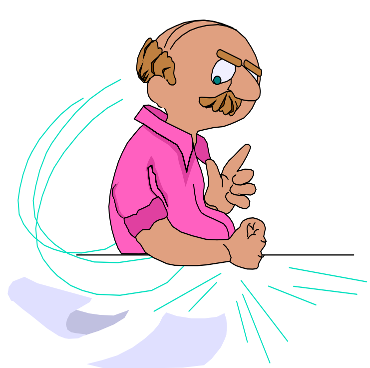
Engine comparison

	Hadoop MapReduce	TEZ	Spark	Flink
API	MapReduce on k/v pairs	k/v pair Readers/Writers	Transformations on k/v pair collections	Iterative transformations on collections
Paradigm	MapReduce	DAG	RDD	Cyclic dataflows
Optimization	none	none	Optimization of SQL queries	Optimization in all APIs
Execution	Batch sorting	Batch sorting and partitioning	Batch with memory pinning	Stream with out-of-core algorithms

117

dbis

Questions??



5. 118

118

PRELIMINARIES

"At a high level, every Spark application consists of a driver program that launches various parallel operations on a cluster. The driver program contains your application's *main function* and *defines distributed datasets* on the cluster, then *applies operations* to them.“

In the following examples, the driver program was the Spark shell. "Driver programs access Spark through a SparkContext object, which represents a connection to a computing cluster. In the shell, a SparkContext is automatically created for you as the variable called sc ."

119

RESILIENT DISTRIBUTED DATASET (RDD)

This is the main Spark ingredient. Spark supports massive parallel computations based on RDD. RDD is where you put your data. An RDD can be viewed as a vector/list of data stored on several machine, where each machine can *access* and *process* part of it. An RDD is fault tolerant: if a machine fails, the RDD is not "broken".

Your job is to build an RDD with your data, and then ask Spark to run operations on that data.

120

TRANSFORMATIONS AND ACTIONS

Spark allows to work on RDD with transformations and actions:

- transformations create a new RDD from an input RDD (e.g., filtering out some unwanted elements).
- actions return a (list of) value(s) to the driver program or write to the (distributed) file system.

Spark exploits a *lazy evaluation* of transformations. Transformation are not applied immediately, but they are *accumulated*. When an action occurs, i.e., some output is required, the Spark framework re-organizes transformations and actions into a parallel execution plan.

121

SPARK SHELL (INTERACTIVE SHELL)

- From terminal launch the command pyspark . This will open a python shell with a spark
  environment.

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.2.0-SNAPSHOT
      /_/

Using Python version 2.7.3 (default, Dec 18 2014 19:10:20)
SparkContext available as sc.
```

122

FIRST RDD

Let's create our first RDD

>>> data = range(20)
>>> myrdd = sc.parallelize(data)
>>> data
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]

>>> myrdd
ParallelCollectionRDD[0] at parallelize at PythonRDD.scala:364

As expected myrdd is *ParallelCollectionRDD*, which allows us to run Spark parallel operations on it.

123

TRANSFORMATIONS

- transformations return a new RDD.
- transformations are not executed immediately, but only when Sparks decides so. (This is called *lazy evaluation*).

124

MAP

Returns a new RDD by applying a given function.
Let's compute the square of each value in myrdd:

```
>>> def sq(x): return x*x
>>> squared = myrdd.map(sq)
>>> print squared.collect()          #THIS IS AN ACTION…
```

….

BLA BLA BLA HADOOP STUFF!

…

[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324, 361]

125

MAP - LAMBDA

For simple functions it is convenient to use python *lambda functions*. Roughly, they are a fast function declaration that defines the function input variable and output value.

```
>>> squared = myrdd.map(lambda x: x*x)
>>> print squared.collect()
```

126

FLAT MAP

Useful when the *map* function generates more then one output.
Let's compute and for any . Compare the two.

```
>>> withmap = myrdd.map(lambda x: (x*x, x*x*x))
>>> print withmap.collect()
[(0, 0), (1, 1), (4, 8), (9, 27), (16, 64), (25, 125), (36, 216), (49, 343), (64, 512), (…

>>> withflatmap = myrdd.flatMap(lambda x: (x*x, x*x*x))
>>> print withflatmap.collect()
[0, 0, 1, 1, 4, 8, 9, 27, 16, 64, 25, 125, 36, 216, 49, 343, 64, 512, 81, 729, 100, 1000
```

127

FLAT MAP – TEXT PROCESSING

This is useful with text processing. Let's find the words of the following three lines of text.

```
>>> divina = [ "Nel mezzo del cammin di nostra vita", "mi ritrovai per una selva oscura",
"ché la diritta via era smarrita."]
>>> divinardd = sc.parallelize(divina)
>>> words = divinardd.flatMap(lambda x:x.split())
>>> print words.collect()

['Nel', 'mezzo', 'del', 'cammin', 'di', 'nostra', 'vita', 'mi', 'ritrovai', 'per', 'una', 'selva',
'oscura', 'ch\xc3\xa9', 'la', 'diritta', 'via', 'era', 'smarrita.']
```

128

FILTER AND SAMPLE

Filter selects a subset of the data. Let's take only even numbers.

```
>>> even = myrdd.filter(lambda x: x%2==0)
>>> print even.collect()

[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

Sample draw a random sample of the data, with or without replacement. Let's take 20% of the data.

```
>>> sample = myrdd.sample(False, 0.20) # false means without replacement
>>> print sample.collect()

[5, 11, 15, 19]
```

129

DISTINCT

Remove duplicates. Let's try on a toy dataset.

```
>>> distinct = sc.parallelize([1,2,2,3,3,3,4,4,4,4]).distinct()
>>> print distinct.collect()

[1, 2, 3, 4]
```

130

UNION AND INTERSECTION

Union.

```
>>> myrdd2 = sc.parallelize(range(10,30))
>>> union = myrdd.union(myrdd2)
>>> print union.collect()
```

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29]

Intersection.

```
>>> intersection = myrdd.intersection(myrdd2)
>>> print intersection.collect()
```

[10, 11, 12, 13, 14, 15, 16, 17, 18, 19]

131

SUBTRACTION AND CARTESIAN PRODUCT

Subtraction.

```
>>> subtraction = myrdd.subtract(myrdd2)
>>> print subtraction.collect()
```

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

Cartesian (every pair-wise combination).

```
>>> cartesian = myrdd.cartesian(myrdd2)
>>> print cartesian.collect()
```

[(0, 10), (0, 11), (0, 12), ..., (0, 28), (0, 29), (1, 10), (1, 11), (1, 12), ...

132

JOINS (STILL TRANSFORMATIONS)

Keys can be used for database-like join operation with the usual semantic:

- join : performs the usual join based on keys
- rightOuterJoin and leftOuterJoin : allows for missing values

The following example is taken from the textbook:

```
>>> left = sc.parallelize( [(1, "red"), (3, "blue"), (3, "green")] )
>>> right = sc.parallelize( [(3, "apples")] )
>>> left.join(right).collect()

[(3, ('blue', 'apples')), (3, ('green', 'apples'))]
```

133

OTHER TRANSFORMATIONS

- keys : returns the list of keys
- values : returns the list of values
- mapValues and flatMapValues : applies the given function to values leaving keys unchanged.
- sortByKey : guess this!
- groupByKey : creates pairs key and list of values associated to the key.
- combineBy : similar to aggregate

134

ACTIONS

They are the operations that return a final value to the driver program or write data to an external storage system. For this reason, the Spark computation is actually triggered when an action is invoked.

The one we used for the previous example is the **collect**: Returns, or better materializes the RDD at the driver program. The collect action returns the whole RDD to the driver program. The RDD is potentially very large and the data transfer may be very expensive.

```
>>> print myrdd.collect()

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

135

COUNT AND COUNT BY VALUE

Count returns the number of elements in the RDD.

```
>> print myrdd.cartesian(myrdd2).count()
400
```

Count by value returns the number of occurrences of each distinct value in the RDD.

```
>>> print squared.countByValue()

{0: 2, 1: 2, 4: 1, 4096: 1, 8: 1, 9: 1, 256: 1, 16: 1, 2197: 1, 512: 1, 25: 1, 27: 1, 289: 1, 36: 1, 4913: 1, 169: 1, 3375: 1, 49: 1, 1331: 1, 2744: 1, 1728: 1, 64: 2, 196: 1, 225: 1, 5832: 1, 6859: 1, 324: 1, 81: 1, 343: 1, 216: 1, 729: 1, 144: 1, 100: 1, 1000: 1, 361: 1, 121: 1, 125: 1}
```

Note 64 occurs twice since it is equal to 8^2 and to 4^3.

136

TAKE AND TOP

Take a few elements from the RDD. It's not sorted and it is not random.

```
>>> print myrdd.take(3)

[0, 1, 2]
```

Top returns the top (largest) elements in descending order.

```
>>> print myrdd.top(3)

[19, 18, 17]
```

137

REDUCE

Aggregates elements according to the provided reduce function.
Let's compute the sum of the elements in a small vector small = [1,2,3,4] . The sum can be computed step-by-step in several ways:
sum = (((1+2)+3)+4)
sum = ((1+2)+(3+4))
etc.
Any function can be used instead of sum with the following constraints. The function should accept *left* and *right* parameters and return the aggregated value. Function should be *associative and distributive*. (Input and return type should not change.

```
>>> def sum(x,y): return x+y                    # first define the function to be used
>>> print myrdd.reduce( sum )                   # apply the function
>>> print myrdd.reduce( lambda x,y: x+y )
190
```

Note: the reduce is executed in parallel, e.g., sum of the elements managed by each machine are computed independently and then merged. Therefore, there is no guarantee about the order of the operations on the RDD.

It's important to remind that partial operations (e.g., sums) are executed and then merged.

138

AGGREGATE

Complex aggregation of the elements in the RDD. Useful when the kind of information to be aggregated is different from the kind of information in the RDD.
It introduces the concept of *accumulators*, i.e., the information to be aggregated. It takes 3 parameters:
the initial value of the accumulator a function that mergers an accumulator with a value in the RDD a function that merges two accumulators.

Let's compute the average of the values in a given RDD. We use a vector of two positions as accumulator, where the first position stores the sum of the elements and the second stores the number of elements. Eventually, the two values in the accumulator are used to compute the average.

139

AGGREGATE (CONT.)

```
# empty accumulator ( partial_sum, partial_count )
>>> empty_acc = (0.0, 0.0)

# merge by summing partial_sum and adding 1 to partial_count
>>> def mergeValue(acc, value): return (acc[0] + value, acc[1] + 1)

# merge by summing partial_sums and partial_counts
>>> def mergeAccum(acc1, acc2): return (acc1[0] + acc2[0], acc1[1] + acc2[1])

# spark aggregate
>>> sum_and_count = myrdd.aggregate( empty_acc, mergeValue, mergeAccum )
>>> print "Average is", sum_and_count[0]/sum_and_count[1]

Average is 9.5
```

140

FOREACH

Applies a given function to each element of the RDD. This is different from map as it does not create a new RDD but it actually generates actions.

```
>>> def f(x): print "This is x:", x
>>> myrdd.foreach( f )
```

141

ACTIONS ON KEY/VALUE PAIRS RDD

Three additional functions are made available for Key/Value pairs RDD.

Count by Key:

```
>>> left.countByKey()
defaultdict(<type 'int'>, {1: 1, 3: 2})
```

(The RDD to refear)

```
>>> left = sc.parallelize( [(1, "red"), (3, "blue"), (3, "green")] )
>>> right = sc.parallelize( [(3, "apples")] )
```

Look up:

```
>>> left.lookup(3)
['blue', 'green']
```

Collect as Map: It is possible to materialize the RDD at the driver as a dictionary.

```
>>> mymap = left.groupByKey().collectAsMap()
>>> for key in mymap:
>>> print key, list(mymap[key]) # this is to convert value from special spark type

1 ['red']
3 ['blue', 'green']
```

142

LOAD A TEXT FILE

Preparation:

- download the files from the course website (dataset/exercises)
- put into hdfs the file (hadoop fs -put <filename>)
- check the file exists (hadoop fs -ls)

Loads a text file into an RDD. Every line of text is stored in an entry of the RDD.

```
>>> poems = sc.textFile("hdfs://…")
>>> poems.take(5)
```

```
[u'', u'', u'', u'\tSONNETS', u'']
```

143

WORD COUT (REDUCE BY KEY)

For each word create a kay-value pair (word, 1) and then group by key and sum up occurrences. Finally, extract the most frequent words, but only if they have at least 600 occurrences.

```
>>> words = ( # parentheses are used just for indentation
        # get words from lines
        poems.flatMap( lambda line: line.split() )
        # crate (word,1) pairs
        .map( lambda word: (word,1) )
        # sum up occurrences
        .reduceByKey( lambda count1,count2: count1+count2 )
        # remove infrequent
        .filter( lambda (word,count): count>=600 )
        # reverse pairs
        .map( lambda (word,count): (count, word) )
        ) w
>>> words.top(10)
```

```
[(1246, u'the'), (920, u'to'), (826, u'and'), (812, u'of'), (742, u'in')]
```

Note: This can be improved with a better tokenization.

144

WRITING TO FILES

The method saveAsTextFile takes a directory path and Spark will output the content of an RDD into multiple files underneath that directory.

```
>>> words.saveAsTextFile("hdfs://…")
```

145