Data Science & Society

Lecture 03:
MapReduce (and a bit of Spark)

INFOMDSS 2018 :: Dr. Marco Spruit

ADS

Required Course Literature: GROUP BY exams

http://www.cs.uu.nl/education/vak.php?stijl=2&vak=INFOMDSS&jaar=2018

Mid-term	End-term	REQUIRED Literature
X	-	White, J. (2016). Hadoop: The Definitive Guide. Third edition. O'Reilly. (Chapters 1,2,3)
-	X	Chambers, B., & Zaharia, M. (2018). Apache Spark - The Definitive Guide. O'Reilly.
X	-	Pritzker, P., and May, W. (2015). NIST Big Data interoperability Framework (NBDIF): Volume 1: Definitions. NIST Special Publication 1500-1. Final Version 1. National Institute of Standards and Technology.
X	-	Dean, J., & Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. Communications of the ACM, 51(1), 107-113.
X	-	Ghemawat, S., Gobioff, H., & Leung, S. (2003). The Google file system. SIGOPS Operating Systems Review, 37(5), 29-43.
X	-	Spruit, M., & Lytras, M. (2018). Applied Data Science in Patient-centric Healthcare: Adaptive Analytic Systems for Empowering Physicians and Patients. Telematics and Informatics, 35(4), 643–653.
-	X	Lazer, D., Kennedy, R., King, G., & Vespignani, A. (2014). The parable of Google Flu: traps in big data analysis. Science, 343(6176), 1203-1205.

Background & Recommended Literature

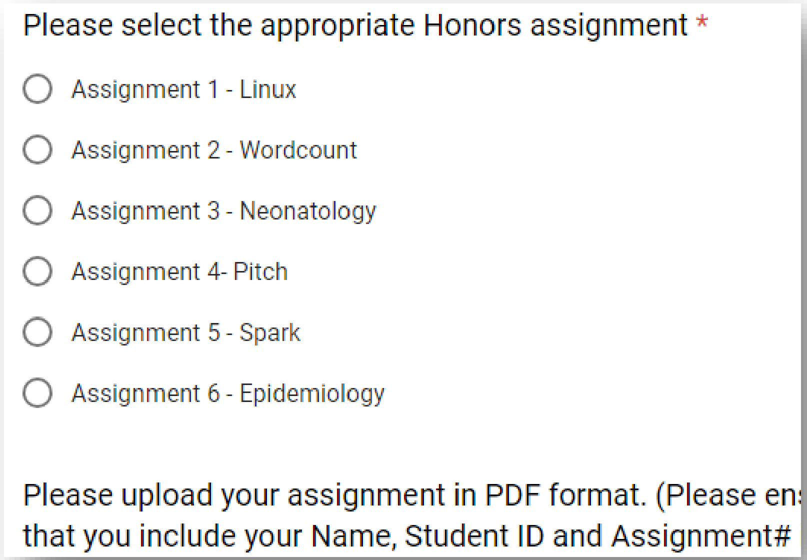
http://www.cs.uu.nl/education/vak.php?stijl=2&vak=INFOMDSS&jaar=2018

Mid-term	End-term	**REQUIRED BACKGROUND Literature**
X	-	Davenport, T. H., & Patil, D. J. (2012). Data scientist: The Sexiest Job of the 21st Century. Harvard business review, 90(5), 70-76.
X	-	Stair, R. & Reynolds, G. (2012). Fundamentals of Information Systems. Sixth Edition. *NOTE: Chapters 1 and 3 ONLY*. Cengage: Boston, MA. ISBN-13: 978-0-8400-6218-5. *(other more recent editions are fine as well).*
X	-	Chapman, P. Clinton, J., Kerber, R., Khabaza, T., Reinartz, T., Shearer, C., and Wirth, R. (2000). CRISP-DM 1.0 Step-by-step Data Mining Guide.

Mid-term	End-term	**RECOMMENDED Literature**
		Manyika, J. (2011). Big data: The next frontier for innovation, competition, and productivity. McKinsey Global Institute, McKinsey & Company.
		Linden, A., Krensky, P., Hare, J., Idoine, C., Sicular, S., & Vashisth, S. (2017). Magic Quadrant for Data Science Platforms. Gartner.
		Spruit,M., & Jagesar,R. (2016). Power to the People! Meta-algorithmic modelling in applied data science. In Fred,A. et al. (Ed.), Proc. 8th Int.Conf. on Knowledge Discovery (pp. 400–406). KDIR 2016, November 11-13, 2016, Porto, Portugal: ScitePress.

About Honours...

- Why?
- Whom of you?
- Please (re-)send me all honors work via
  - http://bit.ly/infomdss-honors



Please select the appropriate Honors assignment *

- ◯ Assignment 1 - Linux
- ◯ Assignment 2 - Wordcount
- ◯ Assignment 3 - Neonatology
- ◯ Assignment 4- Pitch
- ◯ Assignment 5 - Spark
- ◯ Assignment 6 - Epidemiology

Please upload your assignment in PDF format. (Please en
that you include your Name, Student ID and Assignment#

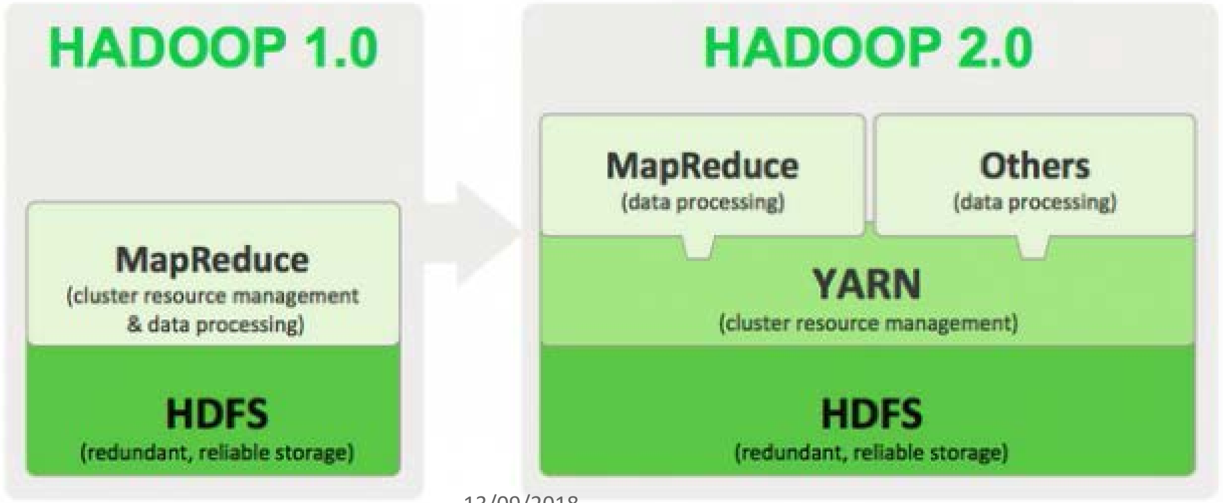
Agenda

- MapReduce
  - A second look
  - Preview of Tutorial 2
  - Rules of Thumb
- Spark
  - NextGen Hadoop
  - Architecture
  - Resilient Distributed Datasets

From lecture 02

The Basic Hadoop Components

- Hadoop **Common** - libraries and utilities
- Hadoop Distributed File System (**HDFS**) – a distributed file-system
- Hadoop **YARN** – a resource-management platform, scheduling
- Hadoop **MapReduce** – a programming model for large scale data processing


HADOOP 1.0
MapReduce
(cluster resource management & data processing)
HDFS
(redundant, reliable storage)
HADOOP 2.0
MapReduce
(data processing)
Others
(data processing)
YARN
(cluster resource management)
HDFS
(redundant, reliable storage)

MapReduce

A second look onto MapReduce

The Motivation for MapReduce

- … is the problem of ever-growing data …
- Big(ger) data means …
  - More storage, on lots of hard disk drives

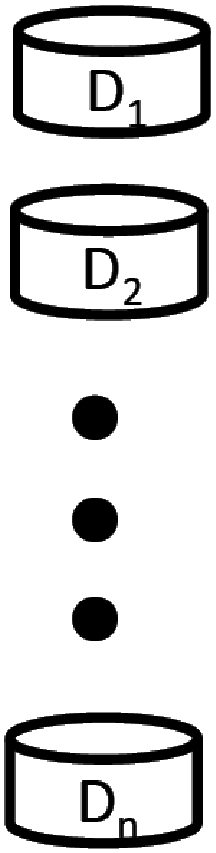
- Processing requires more processing power
  - More CPUs, more RAM -> better hardware
- Accessing & Transporting lots of data …
  - Can your laptop move a Petabyte-file?

- QED? Bring computation to the data!

Case 1: data needs updating

- Processing transactional data
  - e.g. log customer actions, with orders, addresses etc
- What to do?

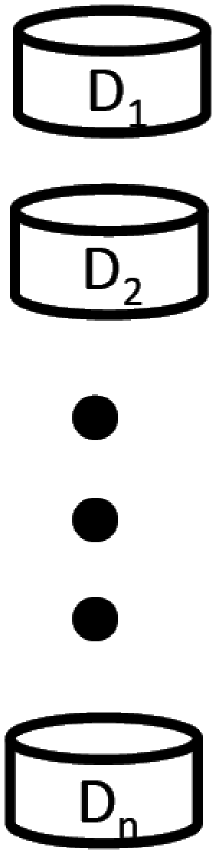
Use an RDBMS!



Case 2: need to sweep through data

- Simple reporting, to organise data output
  - e.g. list customers, with their orders, addresses etc
- What to do?

Take Computation to the Data!



The MapReduce Framework

› User defines:

1. <key, value>
2. mapper & reducer functions

› Hadoop handles the logistics
  – e.g. the distribution and execution

› Logistics have specific requirements!
  – all key-value pairs with the same key need to be processed by the same reducer

MapReduce flow

- User defines a <u>map</u> function
- Hadoop distributes/replicates map() to data
  - map() reads data and outputs <key,value>

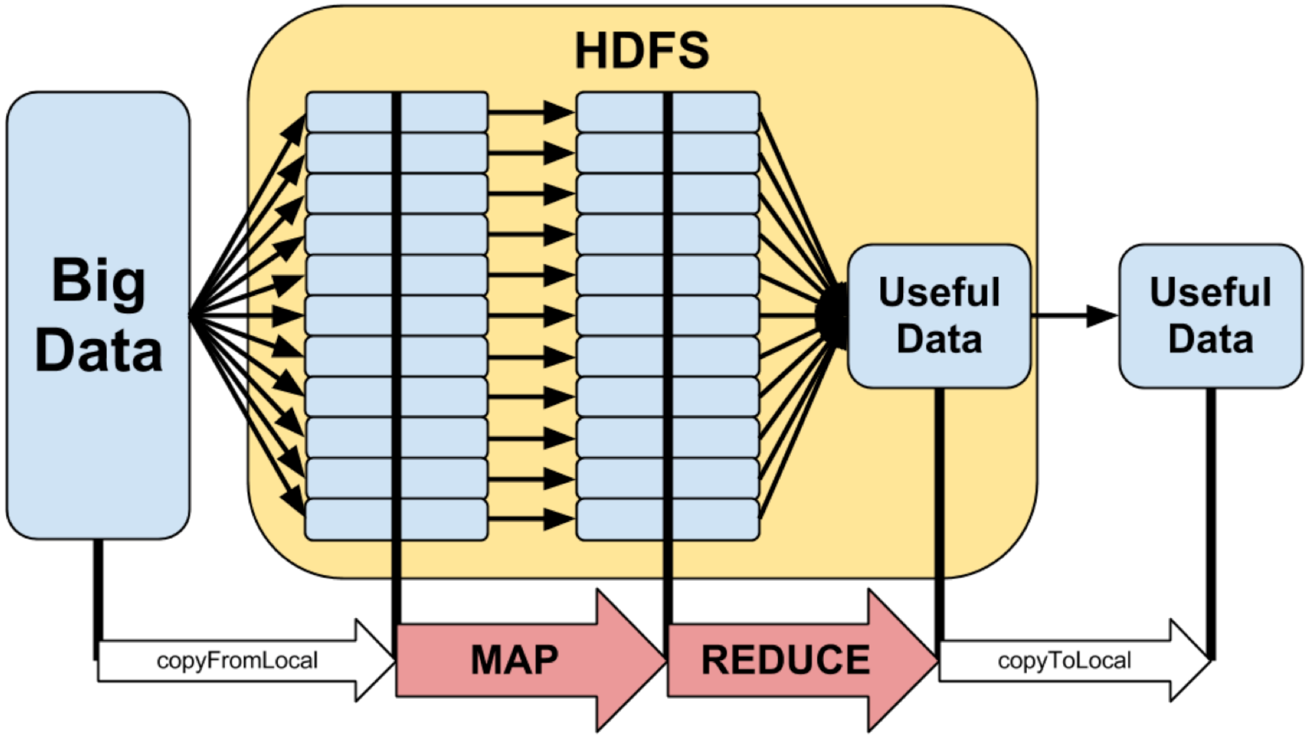
- Hadoop shuffles and groups <key,value> data

- User defines a <u>reduce</u> function
- Hadoop distributes groups to reducers()
  - reduce() reads <key,value> and outputs your result

From lecture 02

MapReduce jobs

› Hadoop jobs go through a map stage and a reduce stage where
  - the mapper transforms the raw input data into key-value pairs where multiple values for the same key may occur
  - the reducer transforms all of the key-value pairs sharing a common key into a single key with a single value



"The" MapReduce Example

Counting word frequencies

Why this example?

A long time ago, …
This was sort-of Yahoo/Google's problem…?

[processing all words in all webpages for search]

How would you count all the words in Star Wars?

- Wordcount pseudo-algorithm:

- In a nutshell:

1. Get word
2. Look up word in table
3. Add 1 to count

- Result?

Word	Count
a	1000
far	2000
Jedi	5000
Luke	9000
…	

ADS

… How about **all** the words in **all** Star Wars texts?

› … And all books, blogs, fan-fiction?

Episode IV

A long time ago, in a galaxy far, far, away …

Use Map-Reduce

Wordcount in MapReduce

› Keep it simple!

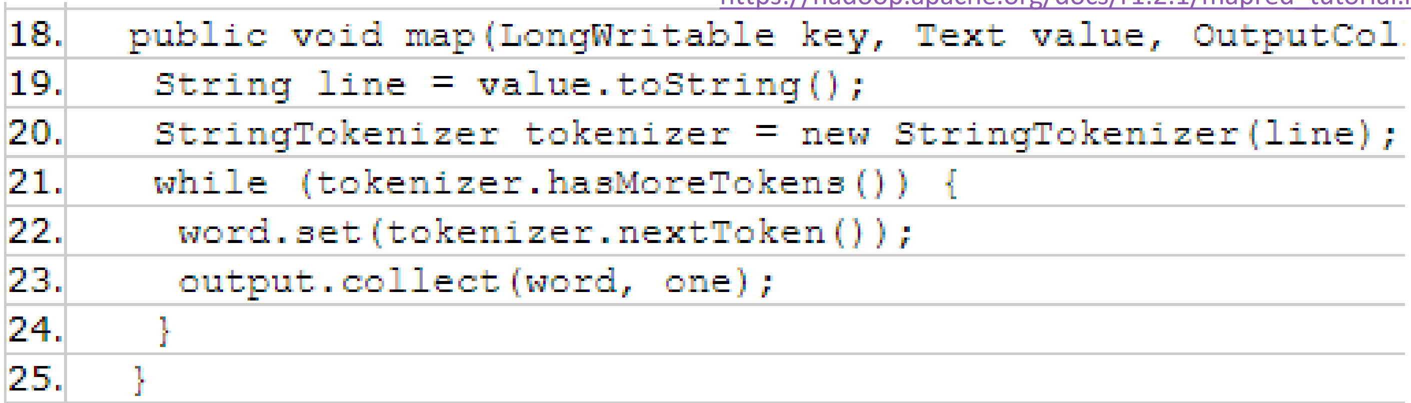
› Let <word, 1> be the <key,value>

› Let Hadoop do the hard work

› The Mapper in pseudo-code:

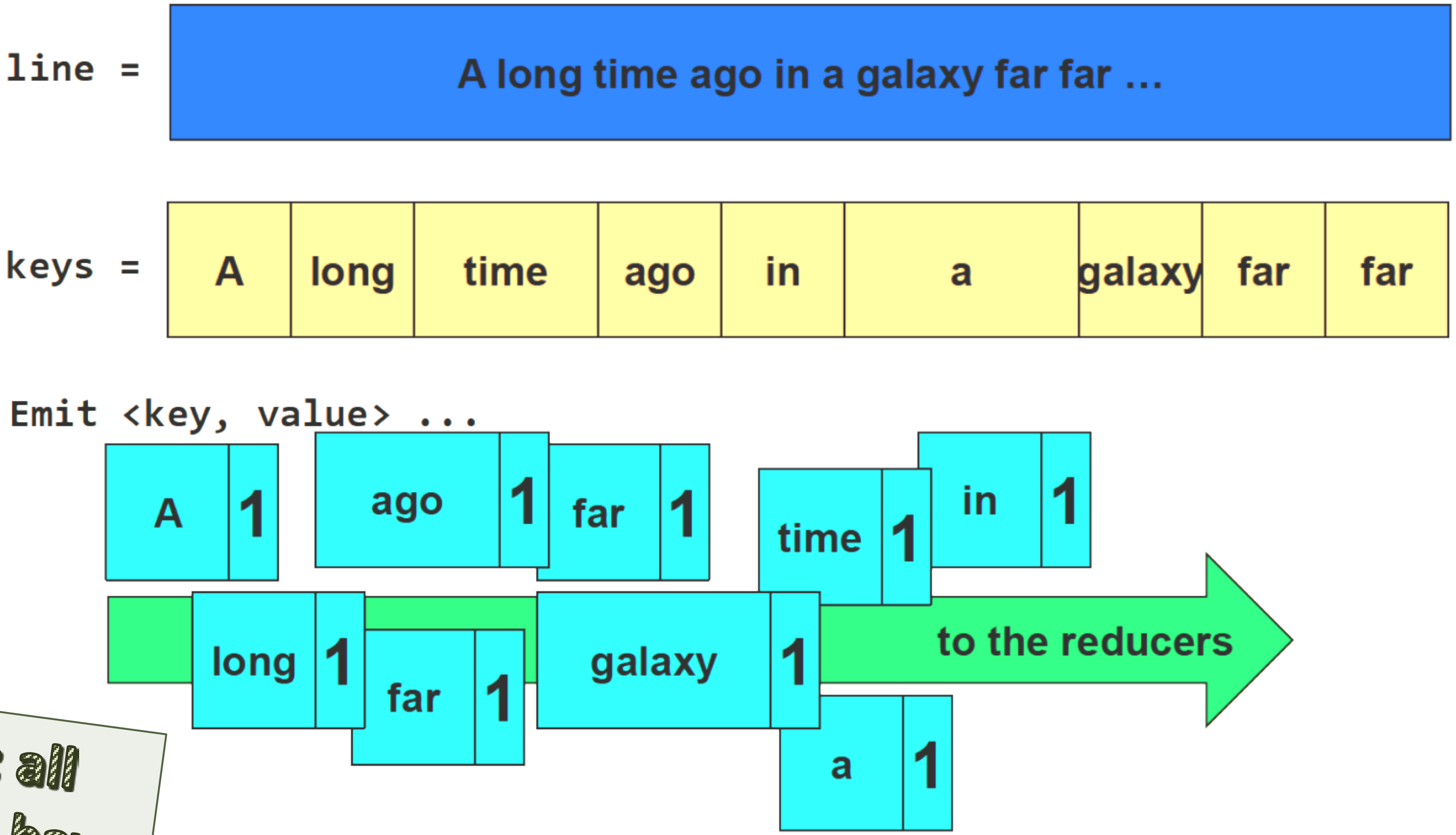
```
Loop      Get word
Until
Done      Emit  <word> < 1>
```

https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html



```
  public void map(LongWritable key, Text value, OutputCol
   String line = value.toString();
   StringTokenizer tokenizer = new StringTokenizer(line);
   while (tokenizer.hasMoreTokens()) {
    word.set(tokenizer.nextToken());
    output.collect(word, one);
   }
  }
```

What one mapper does…



NB: all words have count = 1

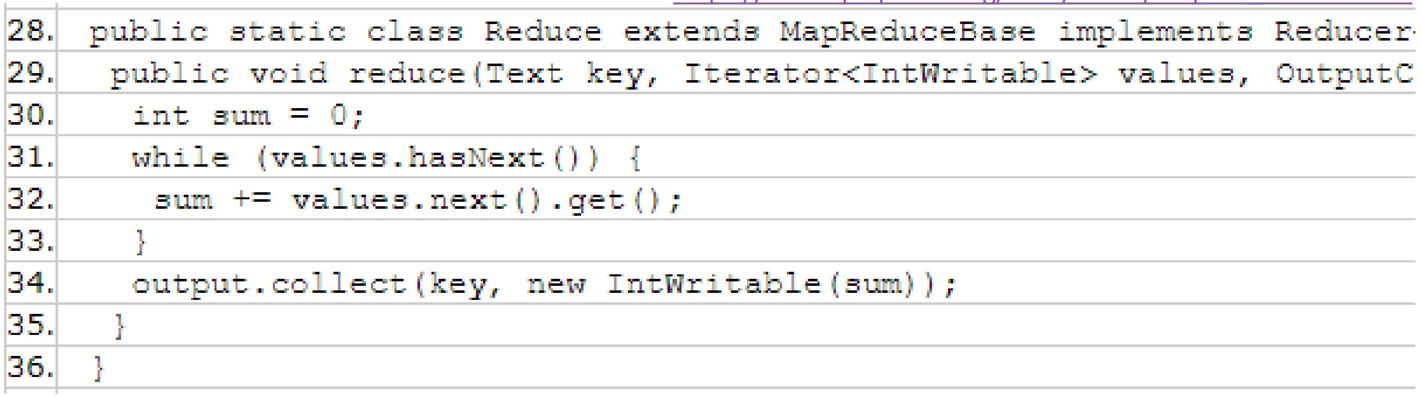
Wordcount MapReduce

› The Reducer in pseudo-code:

Loop Over key-values

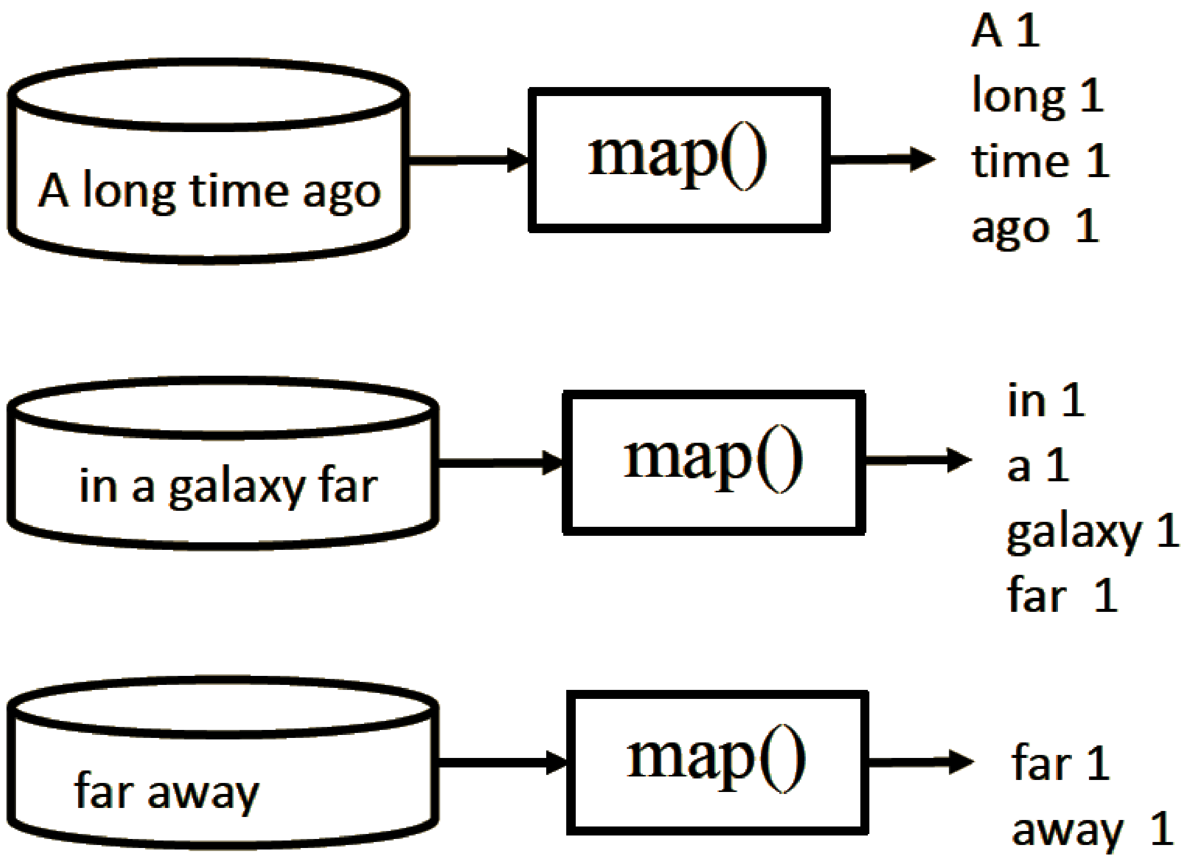
```
Get next <word><value>
If <word> is same as previous word
        add <value> to count
else
        emit <word> < count>
```

https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html

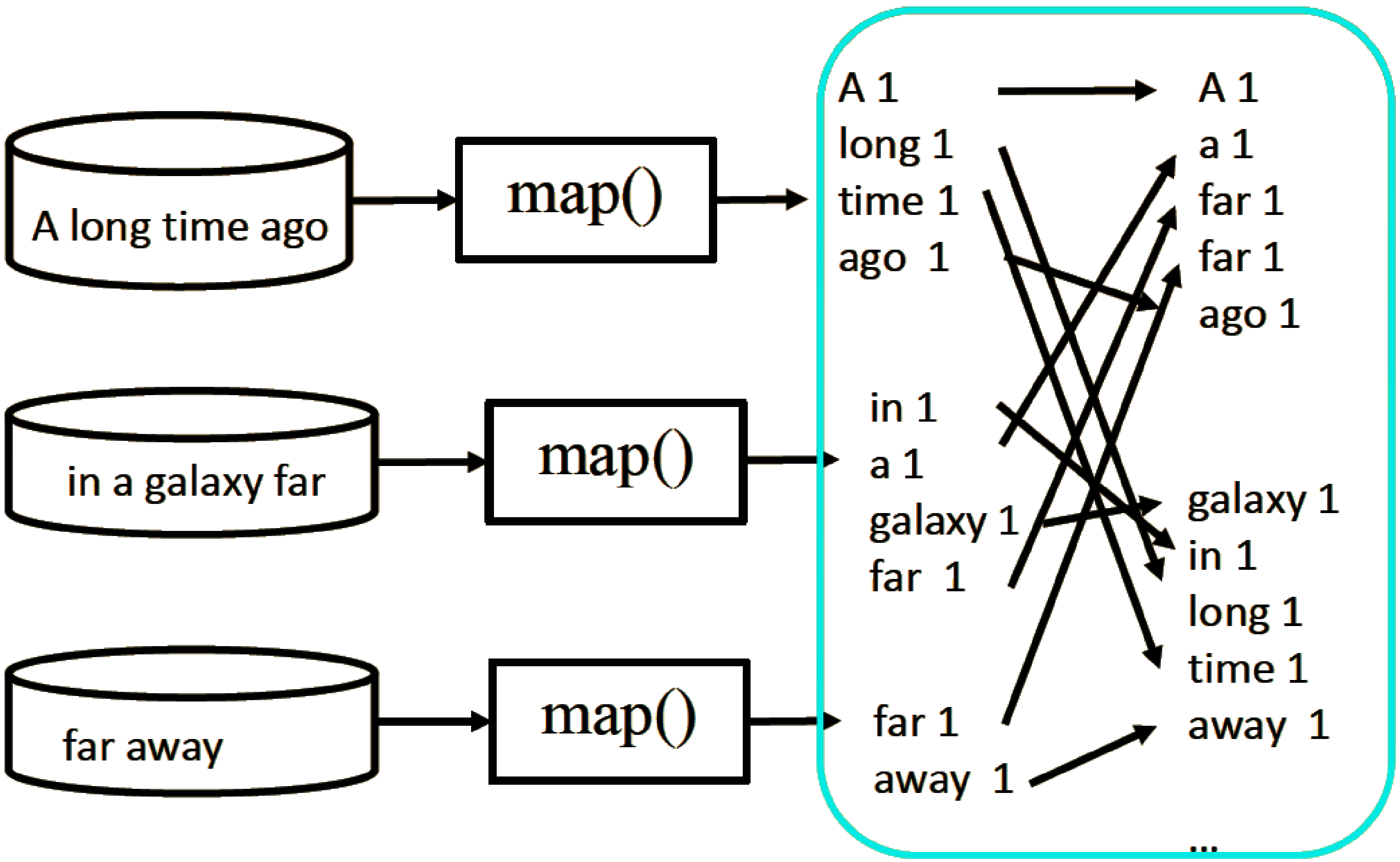


```
public static class Reduce extends MapReduceBase implements Reducer
 public void reduce(Text key, Iterator<IntWritable> values, OutputC
  int sum = 0;
  while (values.hasNext()) {
   sum += values.next().get();
  }
  output.collect(key, new IntWritable(sum));
 }
}
```

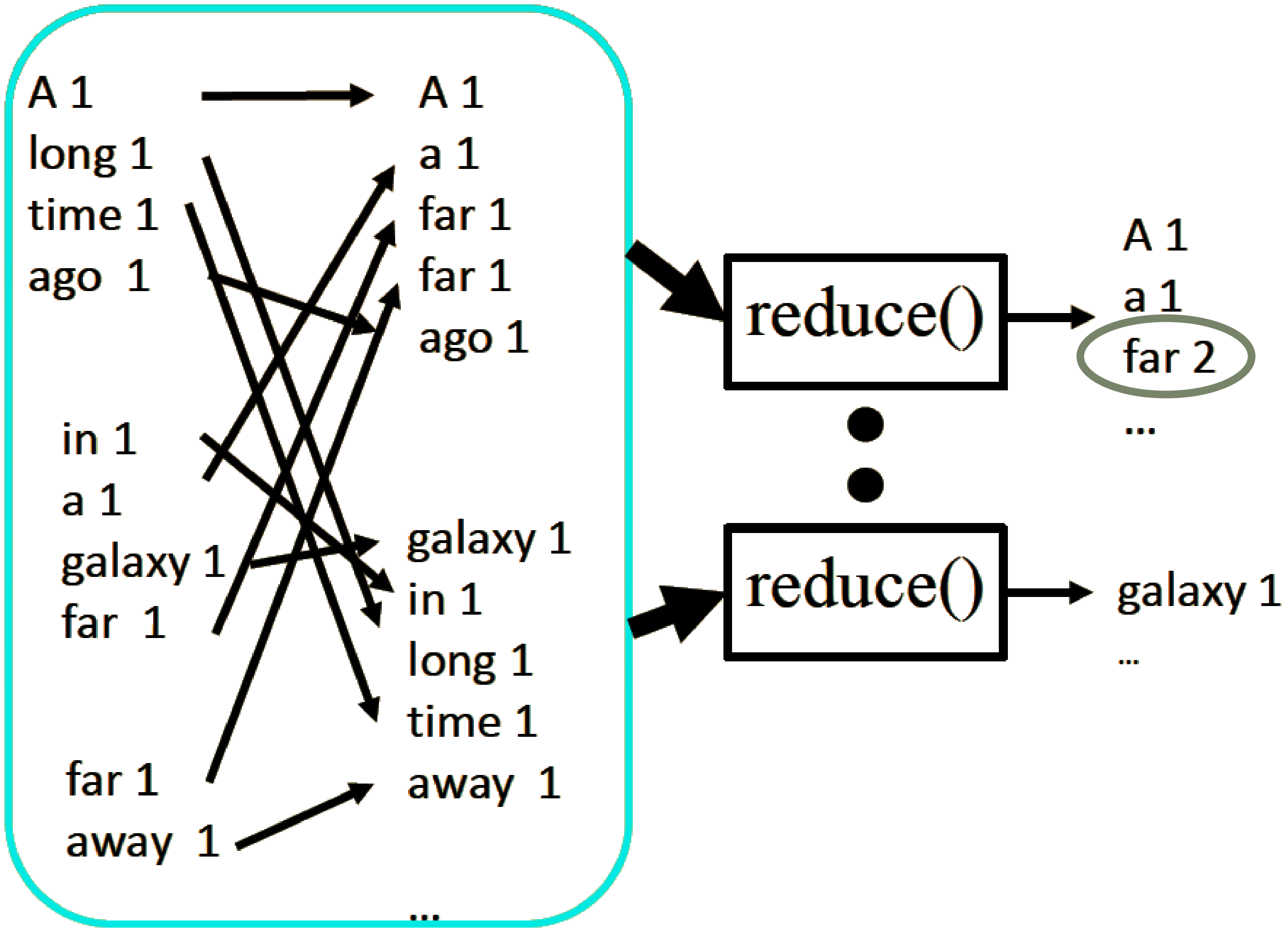
map() output


A long time ago
map()
A 1
long 1
time 1
ago 1
in a galaxy far
map()
in 1
a 1
galaxy 1
far 1
far away
map()
far 1
away 1

Hadoop shuffles, groups, and distributes



reduce() aggregates



Data Science & Society

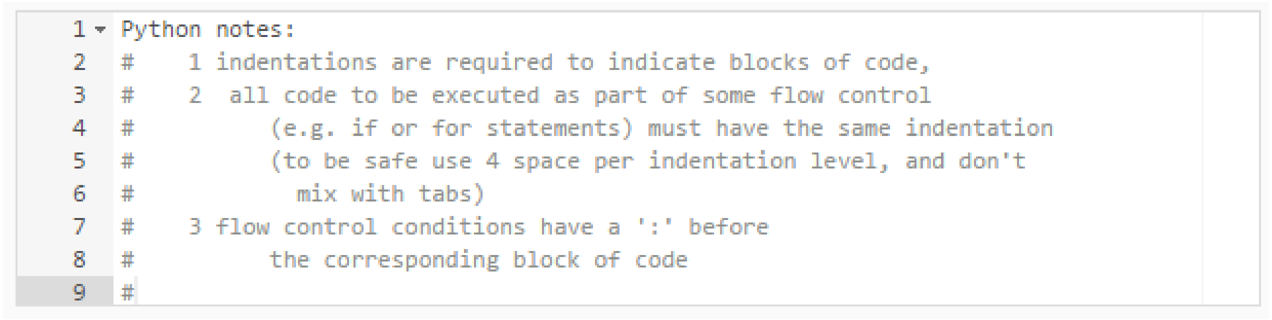
A Preview of Tutorial 02

MapReduce: its ubiquitous example

ADS

Running Wordcount in MapReduce

- 2 ways:
  - Streaming: easier to understand
  - API: lots of function calling, but flexible and fast
- we'll use Streaming with Python



```
Python notes:
#    1 indentations are required to indicate blocks of code,
#    2  all code to be executed as part of some flow control
#          (e.g. if or for statements) must have the same indentation
#          (to be safe use 4 space per indentation level, and don't
#            mix with tabs)
#    3 flow control conditions have a ':' before
#          the corresponding block of code
#
```

- You'll experience it yourself in... Tutorial 2

Hands-on: Map-Reduce Debug tips

- **cat** testfile1 | ./ your-mapper-program.py | sort
- **cat** testfile* | ./ your-mapper-program.py | sort | ./ your-reducer-program.py

- Add logging code

```
myid="_123"
mylog=open("/tmp/mymaplog"+myid,"w")
        mylog.write("reducer = <" + this_key + ", " + value + ">\n")
mylog.close()

more /tmp/mymaplog_123
```

MapReduce Execution

Examples and Principles

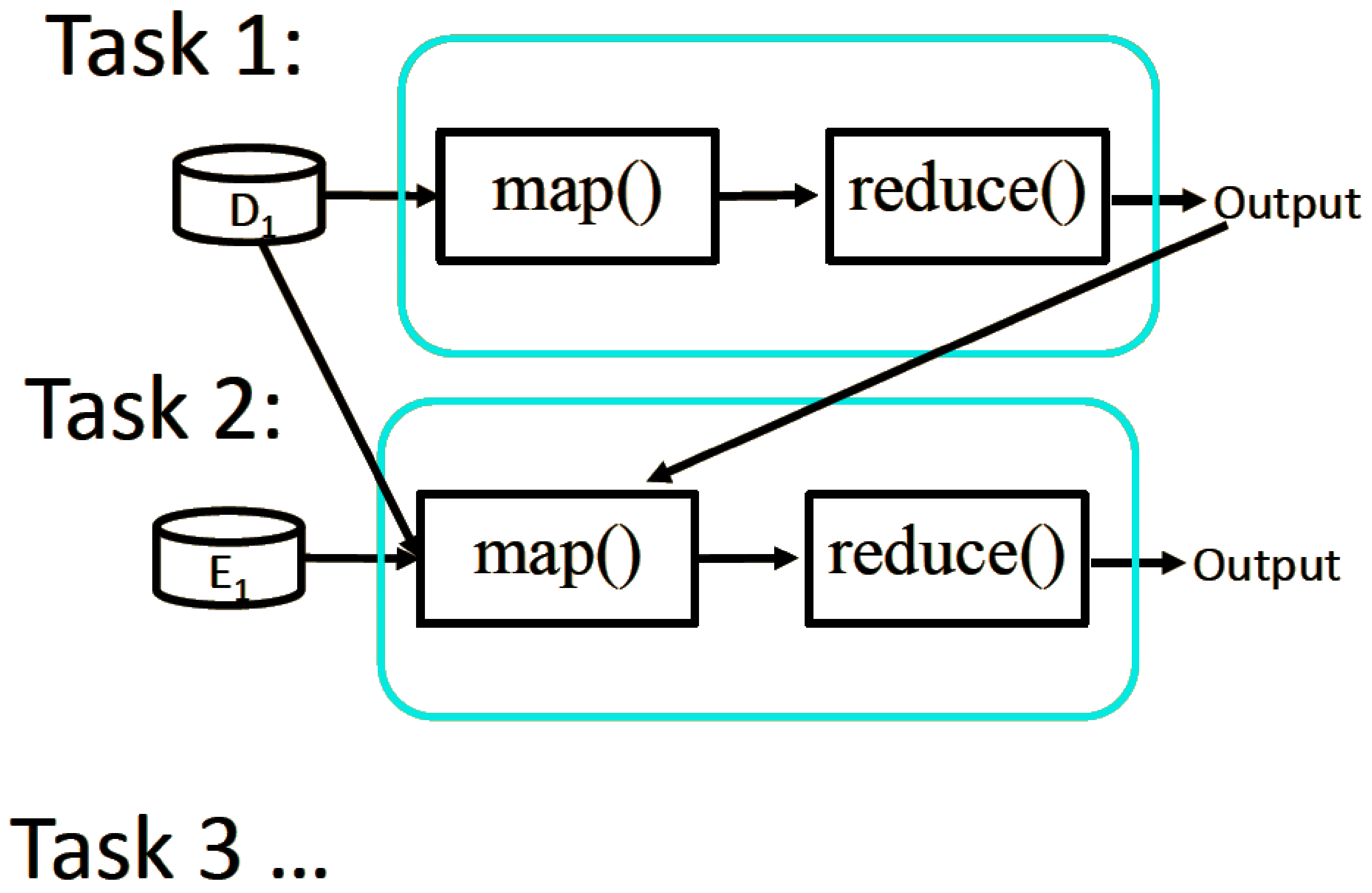
Some Rules of Thumb

- 1 mapper per data split (typically) [more on... StackOverflow]
  - *e.g.* With a 64MB HDFS data block size, for a 500 MB file: 8 MR mappers
- 1 reducer per computer core (best parallelism)

- But… trade-off
  - Number of Output Files
  - Processing Time
- Good key-value properties
  - Key-Value simplicity
  - Enabling reducers to get correct output, through Shuffling & Grouping

- *In short:* Good Task Decomposition
  - *Mappers:* simple and separable
  - *Reducers:* easy consolidation

Cascading MapReduce



Example 2: Trending Wordcount

› *e.g.* **Twitter Data:** date, message, location, … [other metadata]

1. Task 1 = Get word count by day
2. Task 2 = Get total word count

1. Design: Use composite key
   - MapReduce: <date word,count>
2. the easy way: re-use previous wordcount

   the better way: use Task 1 output
   - (it’s partially aggregated)

MapReduce Design Considerations

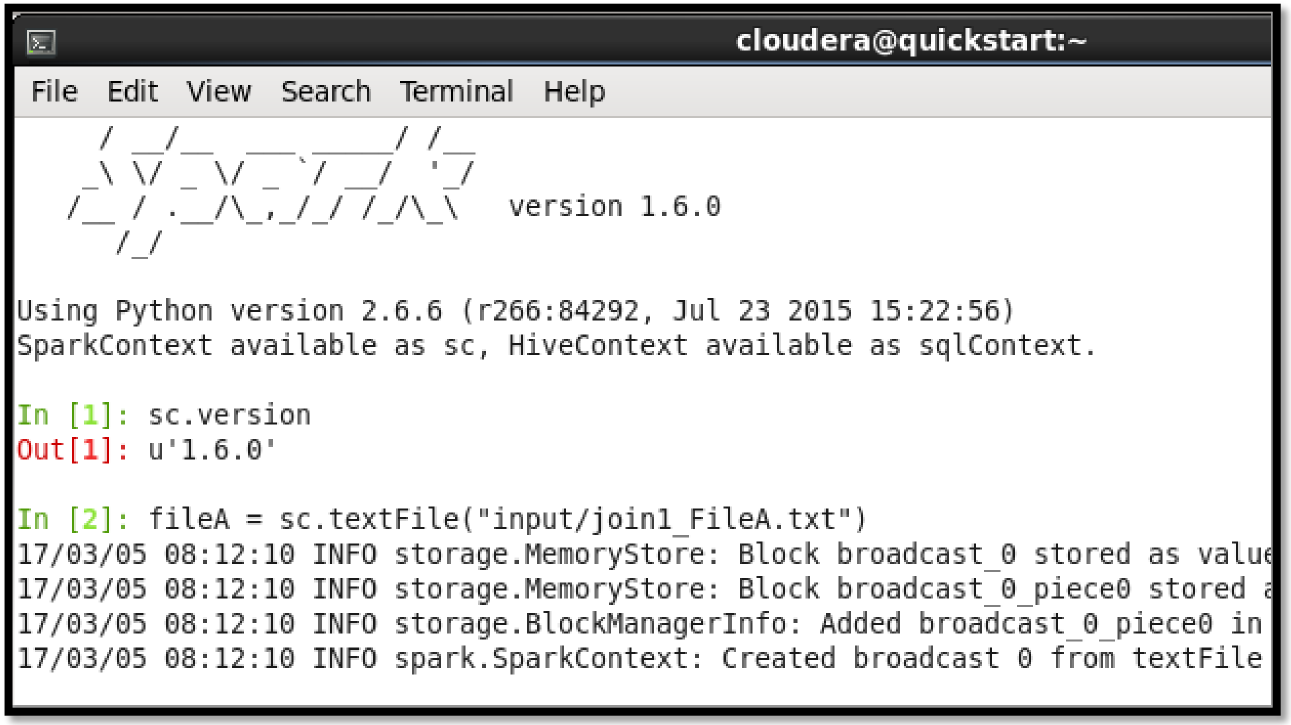
- Composite <keys>
- Extra info in <values>
- Cascade MapReduce jobs
- Bin keys into ranges
- Aggregate map output when possible
  - (combiner option)

Potential Limitations of MapReduce

- Must fit <key, value> paradigm
- MapReduce data not persistent
- Requires programming/debugging
- Not interactive

Beyond MapReduce

› Data access tools (Pig, HIVE)
  – SQL like syntax
› Interactivity & Persistency (Spark)



```
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.6.0
      /_/

Using Python version 2.6.6 (r266:84292, Jul 23 2015 15:22:56)
SparkContext available as sc, HiveContext available as sqlContext.

In [1]: sc.version
Out[1]: u'1.6.0'

In [2]: fileA = sc.textFile("input/join1_FileA.txt")
17/03/05 08:12:10 INFO storage.MemoryStore: Block broadcast_0 stored as value
17/03/05 08:12:10 INFO storage.MemoryStore: Block broadcast_0_piece0 stored a
17/03/05 08:12:10 INFO storage.BlockManagerInfo: Added broadcast_0_piece0 in
17/03/05 08:12:10 INFO spark.SparkContext: Created broadcast 0 from textFile
```

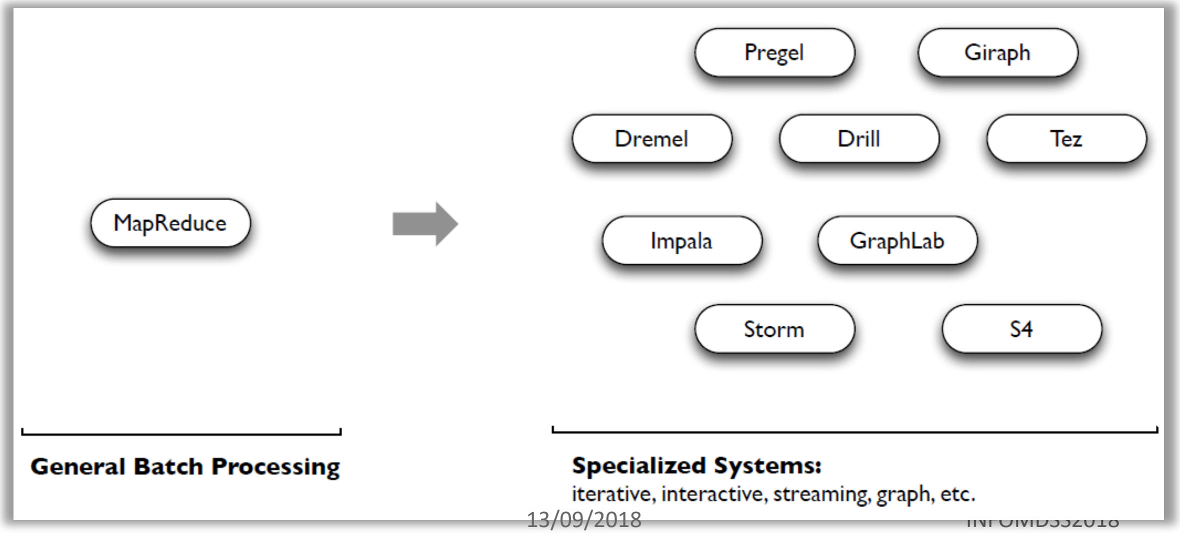
Apache Spark

Solving Some Shortcomings of MapReduce

Recap: Shortcomings of MapReduce

1. Force your pipeline into Map and Reduce steps
   – Other workflows? i.e. join, filter, map-reduce-map
2. Read from disk for each MapReduce job
   – Iterative algorithms? i.e. machine learning
3. Only native JAVA programming interface
   – Other languages? Interactivity?

How Java's Floating-Point Hurts Everyone Everywhere



Solution?

- New framework: same features of MapReduce and more
- Capable of reusing Hadoop ecosystem, e.g. HDFS, YARN…
- Born at UC Berkeley

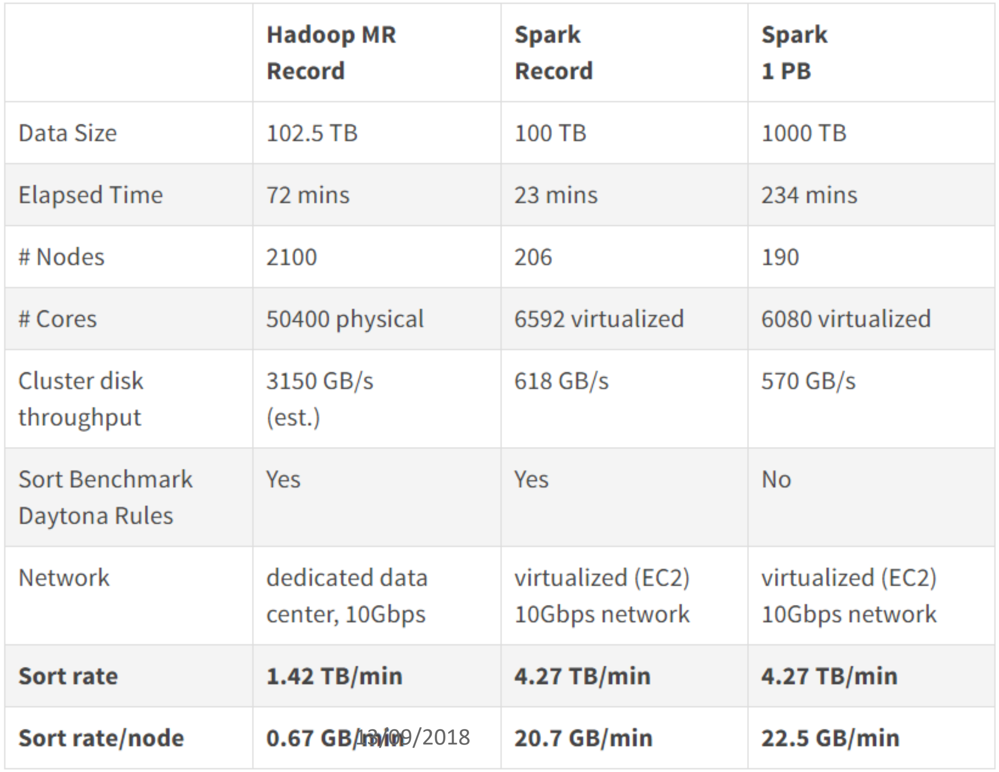
- Unlike the various specialized systems, Spark's goal was to ***generalize*** **MapReduce** to support new apps within same engine

Solutions by Spark

1. Force your pipeline into Map and Reduce steps
   - Other workflows? i.e. join, filter, map-reduce-map
   - 20 highly efficient distributed operations, any combination of them
2. Read from disk for each MapReduce job
   - Iterative algorithms? i.e. machine learning
   - in-memory caching of data, specified by the user
3. Only native JAVA programming interface
   - Other languages? Interactivity?
   - Native Python, Scala (, R) interface. Interactive shells

100TB Sorting competition (1 trillion records!)

- "[...] using Spark on 206 EC2 machines, we sorted 100 TB of data on disk in 23 minutes. In comparison, the previous world record set by Hadoop MapReduce used 2100 machines and took 72 minutes. This means that **Apache Spark sorted the same data 3X faster using 10X fewer machines**. All the sorting took place on disk (HDFS), without using Spark's in-memory cache."



	Hadoop MR Record	**Spark Record**	**Spark 1 PB**
Data Size	102.5 TB	100 TB	1000 TB
Elapsed Time	72 mins	23 mins	234 mins
# Nodes	2100	206	190
# Cores	50400 physical	6592 virtualized	6080 virtualized
Cluster disk throughput	3150 GB/s (est.)	618 GB/s	570 GB/s
Sort Benchmark Daytona Rules	Yes	Yes	No
Network	dedicated data center, 10Gbps	virtualized (EC2) 10Gbps network	virtualized (EC2) 10Gbps network
Sort rate	**1.42 TB/min**	**4.27 TB/min**	**4.27 TB/min**
Sort rate/node	**0.67 GB/min**	**20.7 GB/min**	**22.5 GB/min**

https://databricks.com/blog/2014/11/05/spark-officially-sets-a-new-record-in-large-scale-sorting.html