Distributed Data Management

Summer Semester 2015

TU Kaiserslautern

Prof. Dr.-Ing. Sebastian Michel

Databases and Information Systems Group (AG DBIS)

http://dbis.informatik.uni-kl.de/

Outlook

- Brief outlook on the next 3-4 forthcoming lectures.

- **Today:** Hadoop MapReduce, customizing partitioner/grouping/sorting, n-grams in MR, PageRank in MR.
- **Next week:** PIG, HIVE, and optimizing batches of MR jobs. This is the end of MapReduce in this lecture.
- **Then:** NoSQL databases, data replication, CAP theorem, eventual consistency, …..

HADOOP (A MAPREDUCE IMPLEMENTATION)

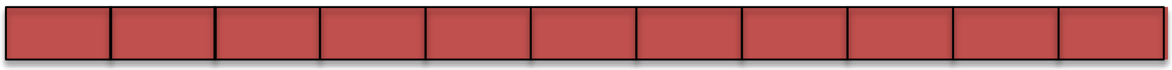
Hadoop MapReduce

- Apache Hadoop. Open Source MR

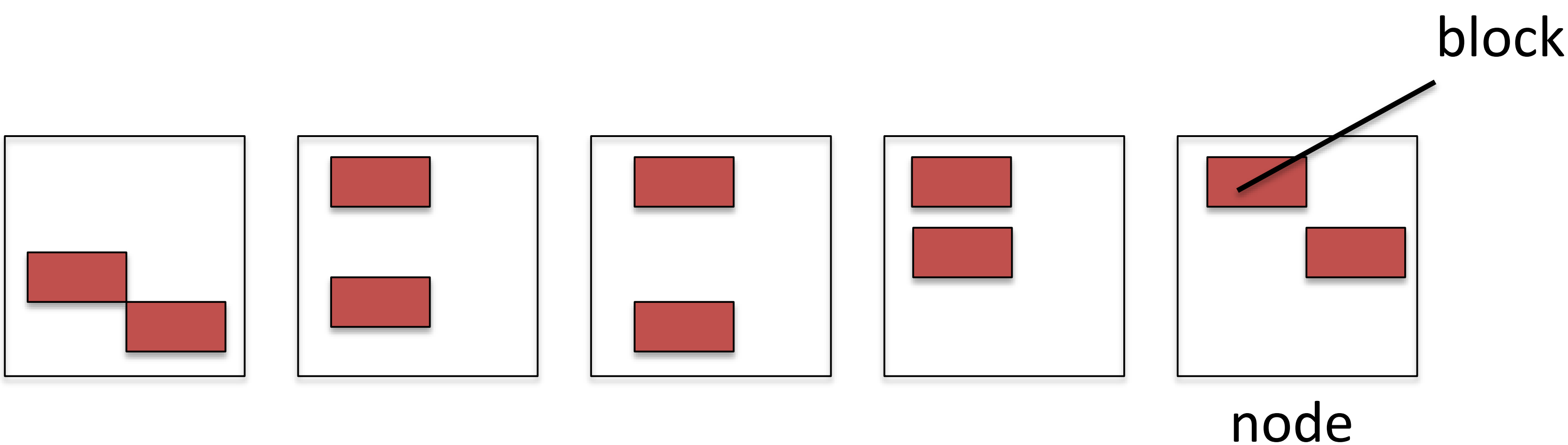
- Wide acceptance:
  – See http://wiki.apache.org/hadoop/PoweredBy
  – Amazon.com, Apple, AOL, eBay, IBM, Google, LinkedIn, Last.fm, Microsoft, SAP, Twitter, …

Hadoop Distributed File System (HDFS): Basics

- Given file is cut in big pieces (blocks) (e.g., 64MB)

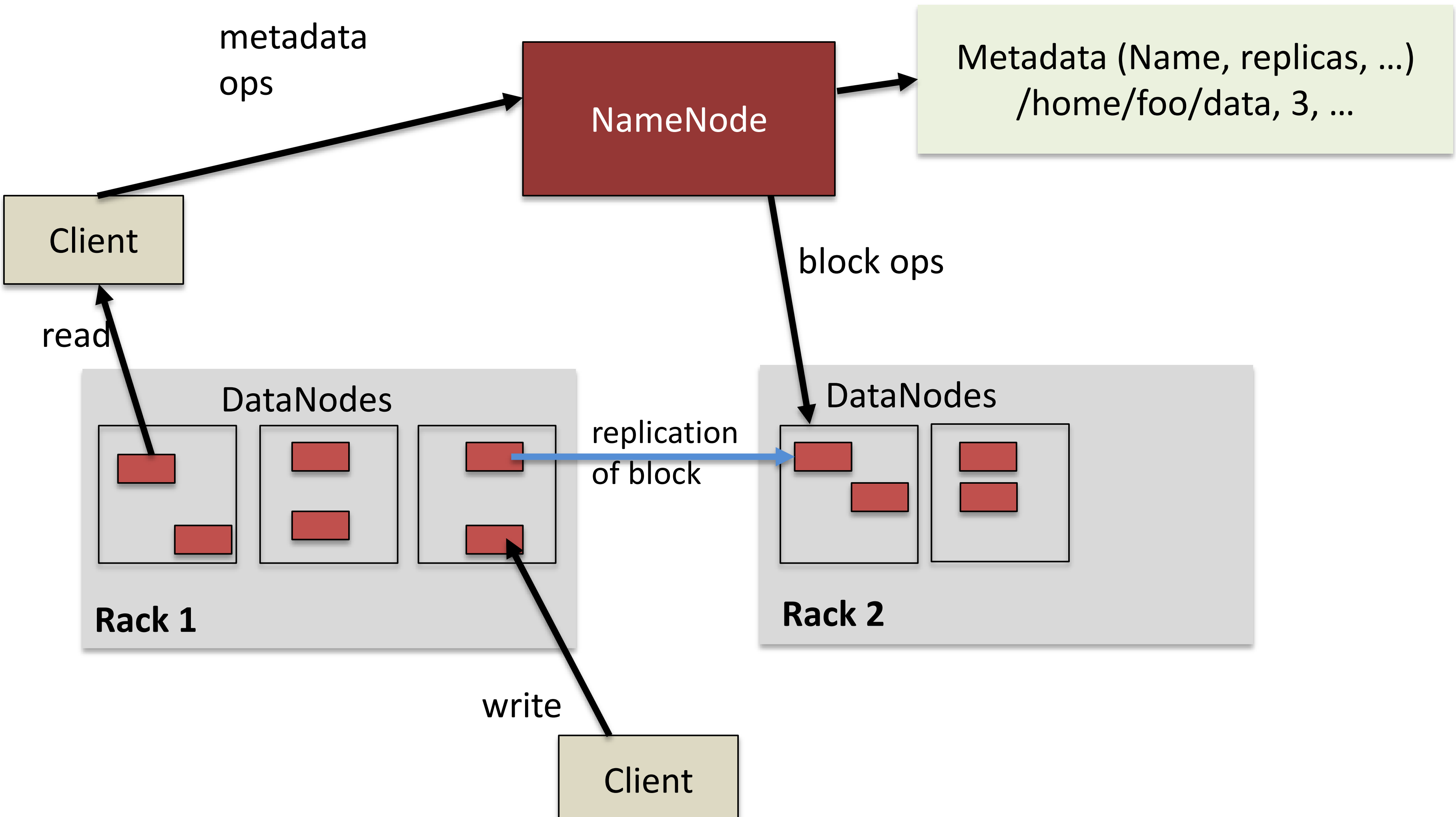


- Which are then assigned to (different) nodes



http://hadoop.apache.org/docs/r2.7.0/hadoop-project-dist/hadoop-hdfs/HdfsUserGuide.html

HDFS Architecture



source: *http://hadoop.apache.org*

UI to Inspect HDFS Properties

127.0.0.1:50070/dfshealth.html#tab-overview

Summary

Security is off.

Safemode is off.

792 files and directories, 420 blocks = 1212 total filesystem object(s).

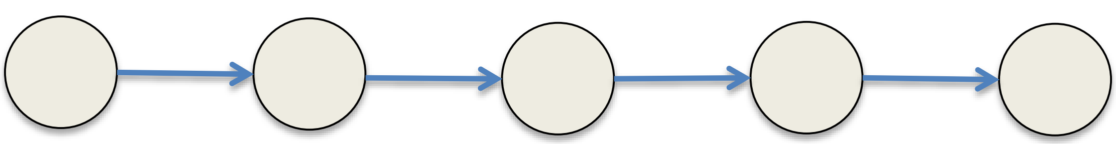
Heap Memory used 102.07 MB of 240 MB Heap Memory. Max Heap Memory is 240 MB.

Non Heap Memory used 59.28 MB of 131.81 MB Commited Non Heap Memory. Max Non Heap Memory is 304 MB.

Configured Capacity:	42.64 GB
DFS Used:	1.11 GB
Non DFS Used:	8.57 GB
DFS Remaining:	32.96 GB
DFS Used%:	2.61%
DFS Remaining%:	77.3%
Block Pool Used:	1.11 GB
Block Pool Used%:	2.61%
DataNodes usages% (Min/Median/Max/stdDev):	2.61% / 2.61% / 2.61% / 0.00%
Live Nodes	1 (Decommissioned: 0)
Dead Nodes	0 (Decommissioned: 0)

Replication

- Can specify default **replication factor** (or per directory/file); default is 3.
- "Rackaware" placement of replicas
- Replication is **pipelined**
  - if block is full, NameNode is asked for other DataNodes (that can hold replica)
  - DataNode is contacted, receives data
  - Forwards to third replica, etc.



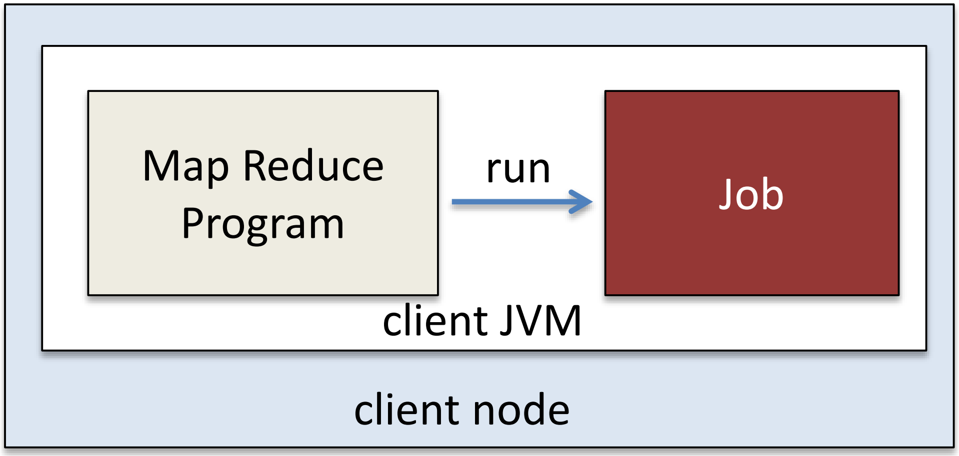
A Note on Input Splits

- An **Input Split** is a chunk of the input data, processed by a single map.
- For instance a set of lines of the original big file.

- **Size of splits usually like size of file system blocks.**

- But does not fit in general precisely with the block boundaries. Then, need to read “a bit” across boundaries.
- Luckily, for applications we consider, we “do not care” and use available input formats.

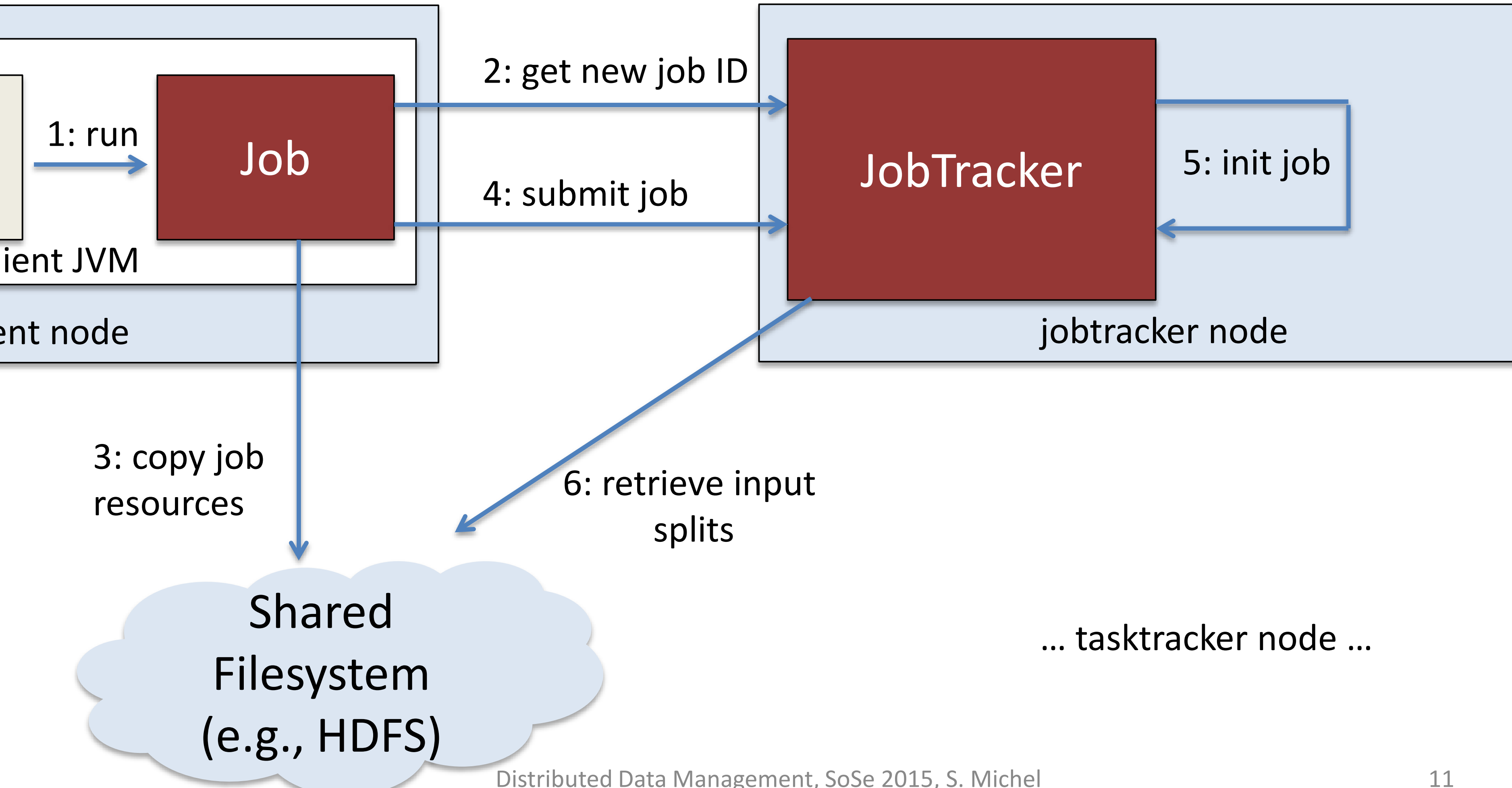
MR job execution in Hadoop



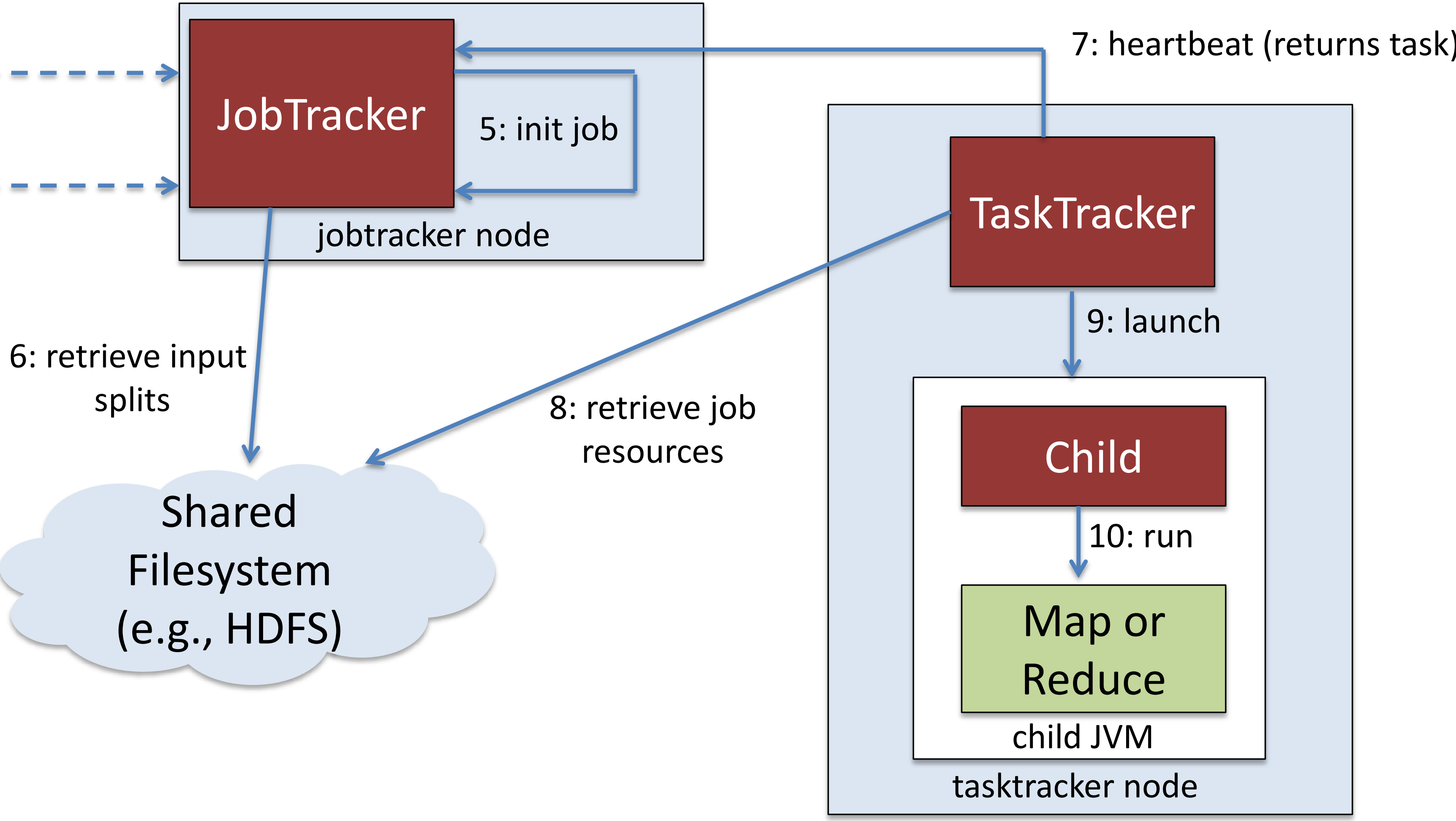
This job exec. in Hadoop considers the very early implementation/architecture for illustrative purposes. For details on Hadoop "MapReduce 2" using YARN see here: http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html

source: T. White, Hadoop, The Definitive Guide, 3rd edition

MR job execution in Hadoop (Cont'd)



MR job execution in Hadoop (Cont'd)



Job Submission, Initialization, Assignment, Execution

- asks for new job id
- checks if input/output directories exist
- computes input splits
- writes everything to HDFS
- submits job to **JobTracker**
- Retrieves splits (chunks) from HDFS
- Creates for each split a Map task
- **TaskTracker** is responsible for executing a certain assigned task (multiple on one physical machine)

Example: First Browse HDFS

Hortonworks Sandbox with HDP 2.2.4 [wird ausgeführt] - Oracle VM VirtualBox

```
[root@sandbox ~]# hdfs dfs -ls /
Found 11 items
drwxrwxrwx   - yarn   hadoop          0 2015-04-30 09:46 /app-logs
drwxr-xr-x   - hdfs   hdfs            0 2015-04-14 05:20 /apps
drwxr-xr-x   - root   hdfs            0 2015-04-30 10:05 /ddm15
drwxr-xr-x   - hdfs   hdfs            0 2015-04-14 05:55 /demo
drwxr-xr-x   - hdfs   hdfs            0 2015-04-14 05:03 /hdp
drwxr-xr-x   - mapred hdfs            0 2015-04-14 05:02 /mapred
drwxr-xr-x   - hdfs   hdfs            0 2015-04-14 05:02 /mr-history
drwxr-xr-x   - hdfs   hdfs            0 2015-04-14 05:45 /ranger
drwxr-xr-x   - hdfs   hdfs            0 2015-04-14 05:07 /system
drwxrwxrwx   - hdfs   hdfs            0 2015-04-30 09:46 /tmp
drwxr-xr-x   - hdfs   hdfs            0 2015-04-30 07:08 /user
[root@sandbox ~]# hdfs dfs -ls /ddm15
Found 4 items
drwxr-xr-x   - root hdfs          0 2015-04-30 07:07 /ddm15/text
drwxr-xr-x   - root hdfs          0 2015-04-30 09:36 /ddm15/weather-out
-rw-r--r--   1 root hdfs  448818933 2015-04-30 09:32 /ddm15/weather_shuf.csv
drwxr-xr-x   - root hdfs          0 2015-04-30 10:05 /ddm15/wordcount-out
[root@sandbox ~]# hdfs dfs -ls /ddm15/text
Found 1 items
-rw-r--r--   1 root hdfs     167776 2015-04-30 07:07 /ddm15/text/hamlet.txt
[root@sandbox ~]# _
```

Starting the WordCount Job

- Have a ddm15.jar with the WordCount class in a package called mapred.
- Input file and output folder specified

Hortonworks Sandbox with HDP 2.2.4 [wird ausgeführt] - Oracle VM VirtualBox

```
[root@sandbox src]# hadoop jar ddm15.jar mapred/WordCount /ddm15/text/hamlet.txt
 /ddm15/wordcount-out_
```

- This starts the MapReduce job; you will see plenty of output info and updates on completion (in Percent).

Inspect the Results

- We see one file per reducer and a file with name _SUCCESS

Hortonworks Sandbox with HDP 2.2.4 [wird ausgeführt] - Oracle VM VirtualBox

```
[root@sandbox ~]# hdfs dfs -ls /ddm15/wordcount-out/
Found 5 items
-rw-r--r--   1 root hdfs          0 2015-05-07 06:49 /ddm15/wordcount-out/_SUCCE
SS
-rw-r--r--   1 root hdfs      18068 2015-05-07 06:49 /ddm15/wordcount-out/part-r
-00000
-rw-r--r--   1 root hdfs      17608 2015-05-07 06:49 /ddm15/wordcount-out/part-r
-00001
-rw-r--r--   1 root hdfs      17804 2015-05-07 06:49 /ddm15/wordcount-out/part-r
-00002
-rw-r--r--   1 root hdfs      17631 2015-05-07 06:49 /ddm15/wordcount-out/part-r
-00003
[root@sandbox ~]# _
```

- Let’s have a look at one of the files

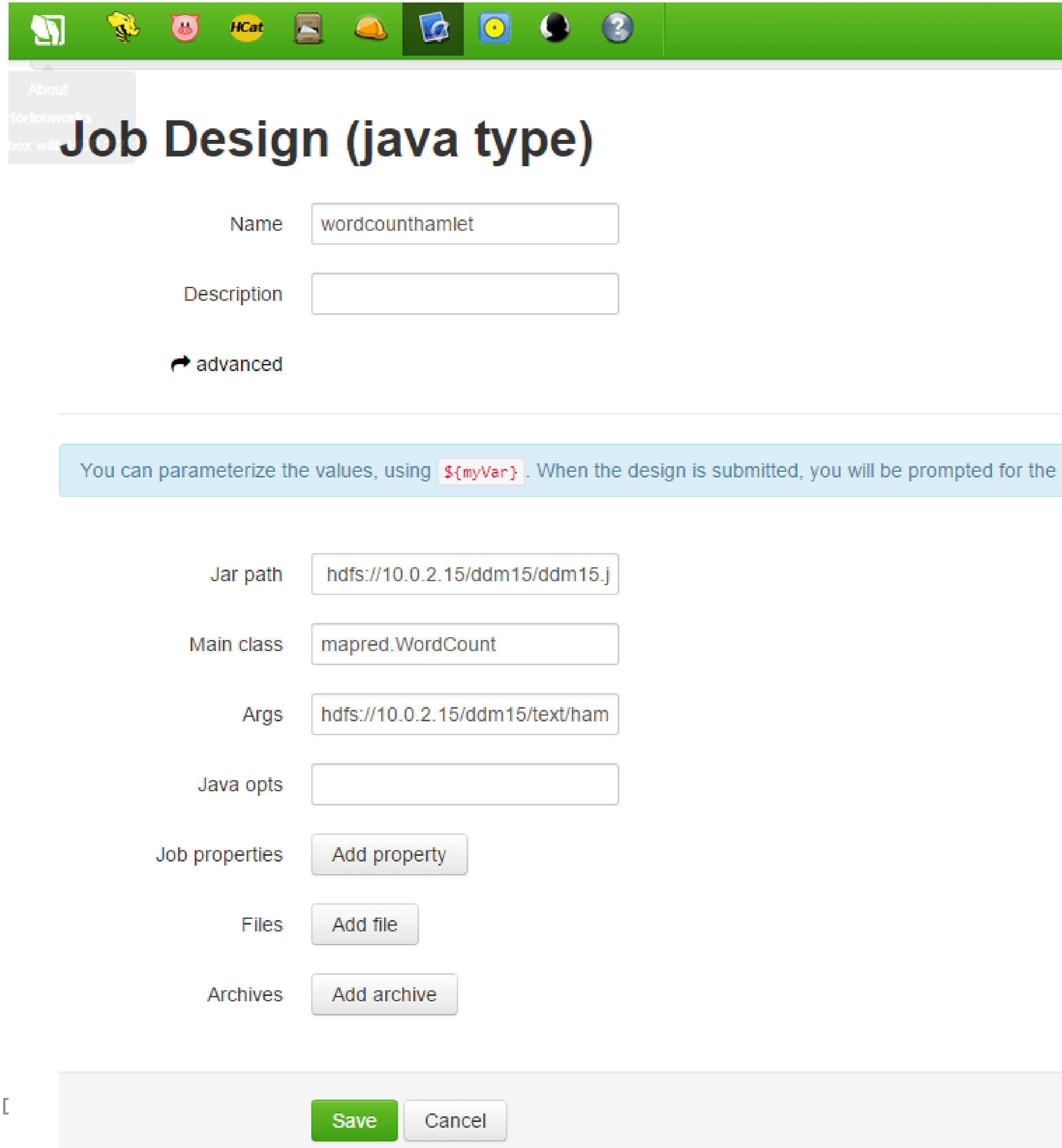
Hortonworks Sandbox with HDP 2.2.4 [wird ausgeführt] - Oracle VM VirtualBox

```
[root@sandbox ~]# hdfs dfs -cat /ddm15/wordcount-out/part-r-00000 | head
'Tis    26
'Twill  1
'tis    36
'tis,   2
'tweene 1
'twill  3
```

Alternatively, there are GUIs ….

If you use the hortonworks virtual machine, you can use it right away

There are also simple Uis for monitoring progress/status in Hadoop and HDFS directly.



File Browser

Search for file name | Rename | Move | Copy | Change Permissions | New | Upload
Download | Delete

Home / ddm15 / text-out Trash

	Type	Name	Size	User	Group	Permissions	Date
		.		hue	hdfs	drwxr-xr-x	May 07, 2015 01:02 AM
		..		root	hdfs	drwxr-xr-x	May 07, 2015 01:02 AM
		_SUCCESS	0 bytes	hue	hdfs	-rw-r--r--	May 07, 2015 01:02 AM
		part-r-00000	17.6 KB	hue	hdfs	-rw-r--r--	May 07, 2015 01:02 AM
		part-r-00001	17.2 KB	hue	hdfs	-rw-r--r--	May 07, 2015 01:02 AM
		part-r-00002	17.4 KB	hue	hdfs	-rw-r--r--	May 07, 2015 01:02 AM
		part-r-00003	17.2 KB	hue	hdfs	-rw-r--r--	May 07, 2015 01:02 AM

Stragglers and Speculative Execution

- **JobTracker continuously controls progress (see Web user interface)**

- **Stragglers** are slow nodes
  - have to wait for the slowest one (think: only one out of 1000 is slow and delays overall response time)

- **Speculative execution**
  - run same task on more nodes if the first instance is observed to underperform (after some time)
  - wasted resources vs. improved performance

Failure/Recovery in MR

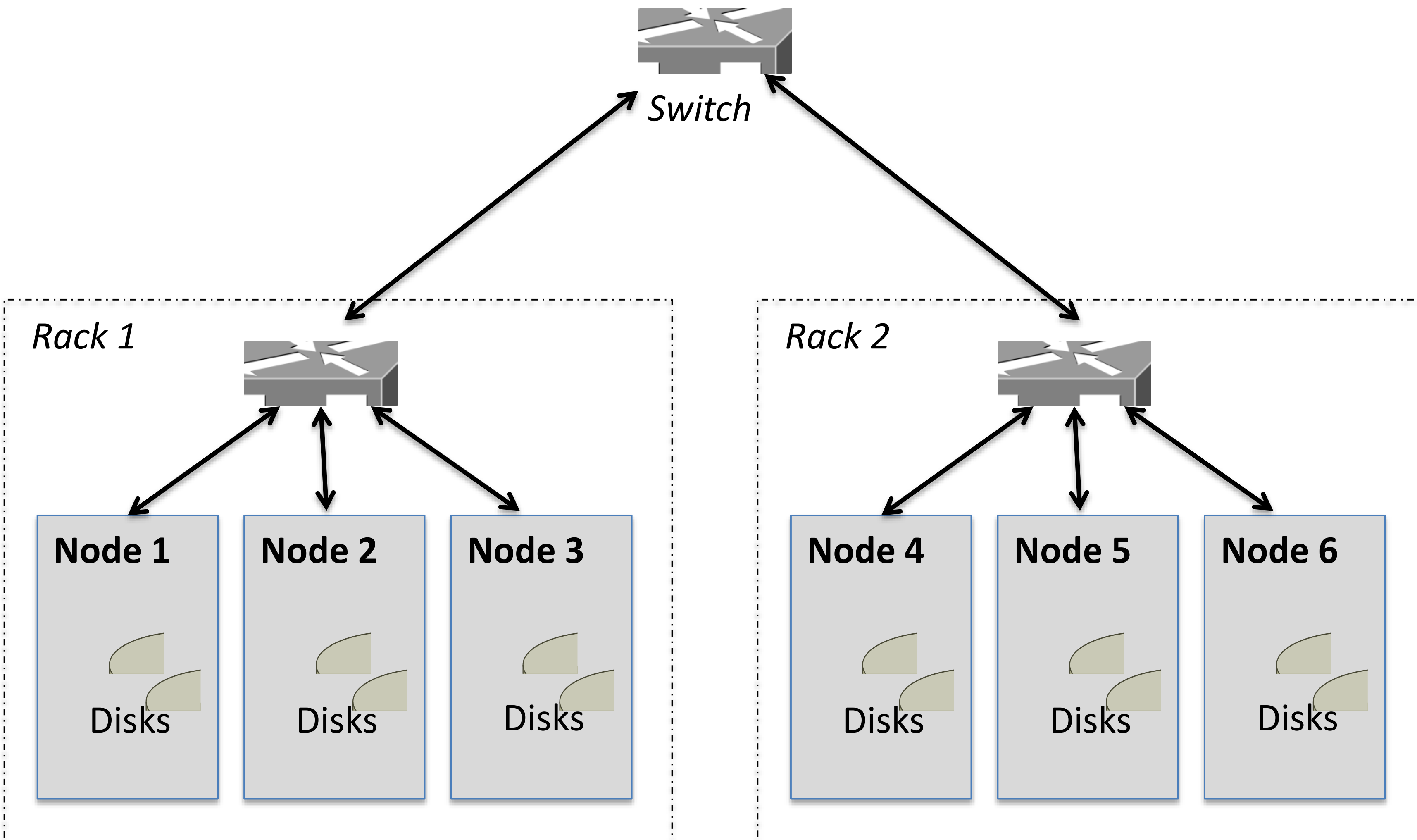
- **Task or Tasktracker failure:**
  - detected by master through periodic heartbeats
  - can also be black listed if too many failures occur
  - just restart if dead.
  - Jobtracker re-schedules failed task (but not again on the same Tasktracker)

- **Jobtracker failure**:
  - unlikely to happen (only one machine) but if: all running jobs failed
  - improved in Hadoop “2” (YARN)

… and Specifically in HDFS

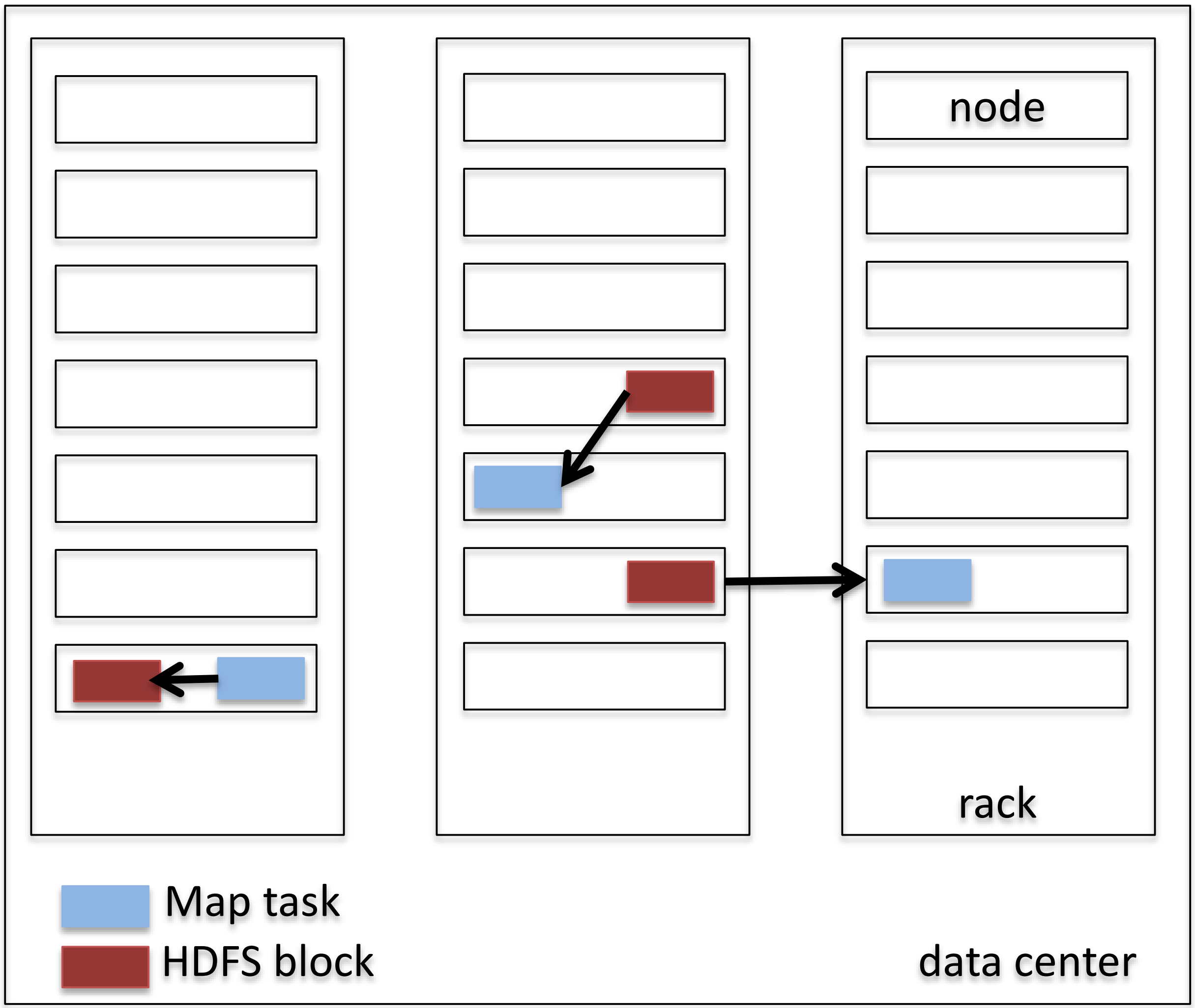
- NameNode marks DataNodes without recent Heartbeats as dead
- Replication factor of some blocks can fall below their specified value
- The NameNode constantly tracks which blocks need to be replicated and initiates replication whenever necessary.
- If NameNode crashed: Manual restart/recovery.

Typical Setup



source: T. White, Hadoop, The Definitive Guide, 3rd edition

Locality



- data-local
- rack-local
- off-rack

map tasks

source: T. White, Hadoop, The Definitive Guide, 3rd edition

Cost Model + Configuration for Rack Awareness

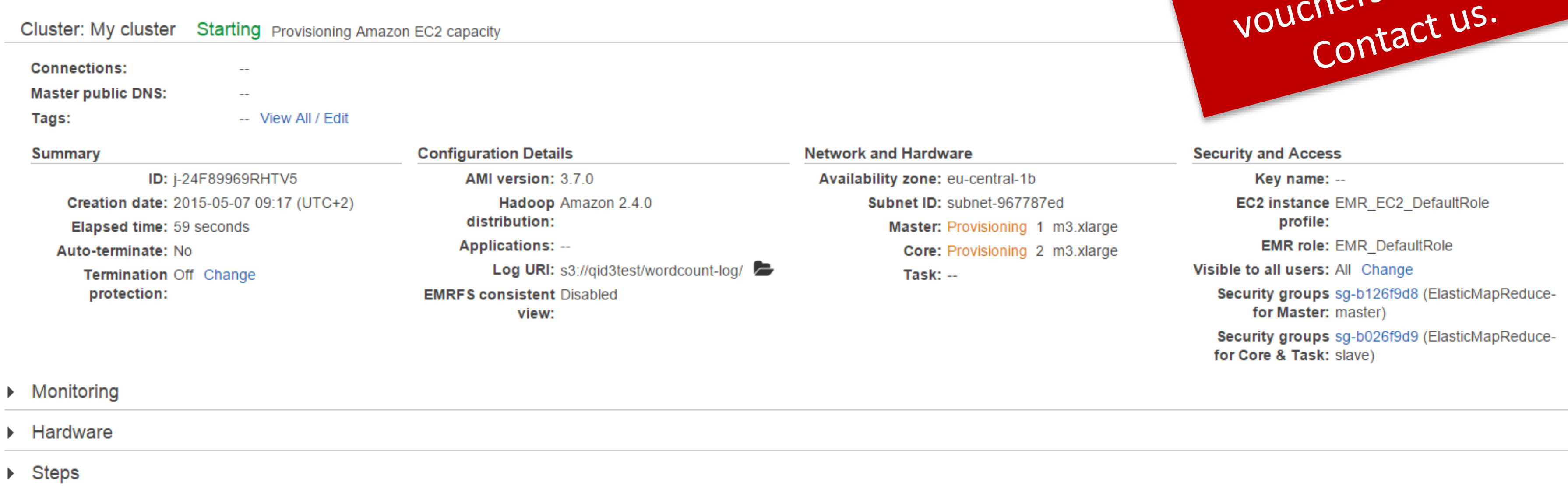
- Simple cost model applied in Hadoop:
  - Same node: 0
  - Same rack: 2
  - Same data center: 4
  - Different data center: 6
- Hadoop needs help: You have to specify config. (**topology**)
- Sample configuration:

```
'13.2.3.4' : '/datacenter1/rack0',
'13.2.3.5' : '/datacenter1/rack0',
'13.2.3.6' : '/datacenter1/rack0',
'10.2.3.4' : '/datacenter2/rack0',
'10.2.3.4' : '/datacenter2/rack0'
....
```

MapReduce in Amazon AWS

- Amazon offers running MapReduce in the Cloud.
- Called Elastic MapReduce (EMR)
- You can put data in the S3 storage
- And start a MR job by uploading your custom .jar file

There are still AWS vouchers available. Contact us.



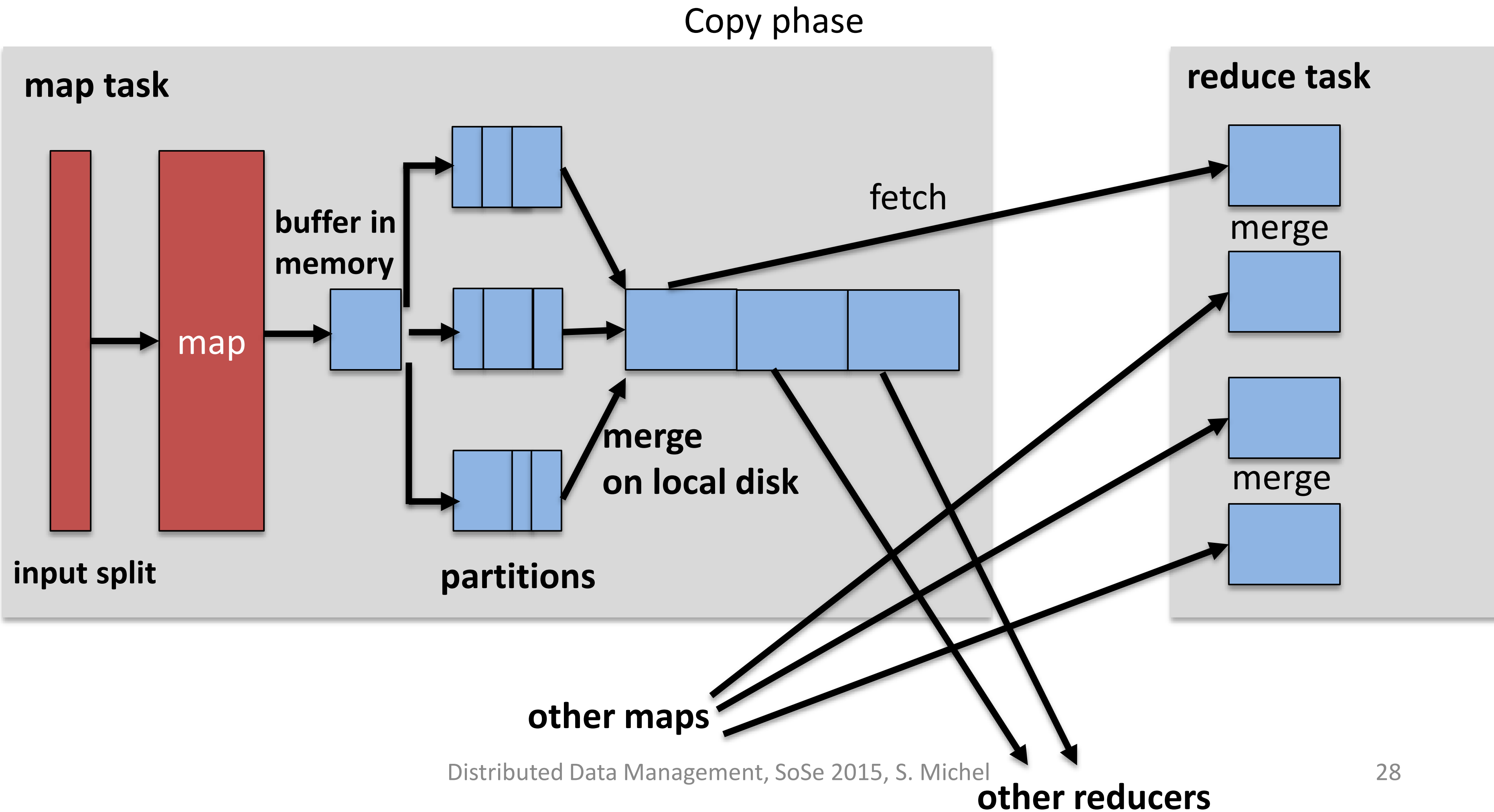
CUSTOMIZING PARTITIONING/SORTING/GROUPING IN HADOOP

Shuffle and Sort: Overview

- **Output** of map **is partitioned by key** as standard
- Reducer is guaranteed to get **entire partition**
- **Sorted by key (but not by value within each group)**
- Output of each reducer is sorted also by this key
- Selecting which key to use, hence, affects partitions and sort order (see few slides later how to customize)

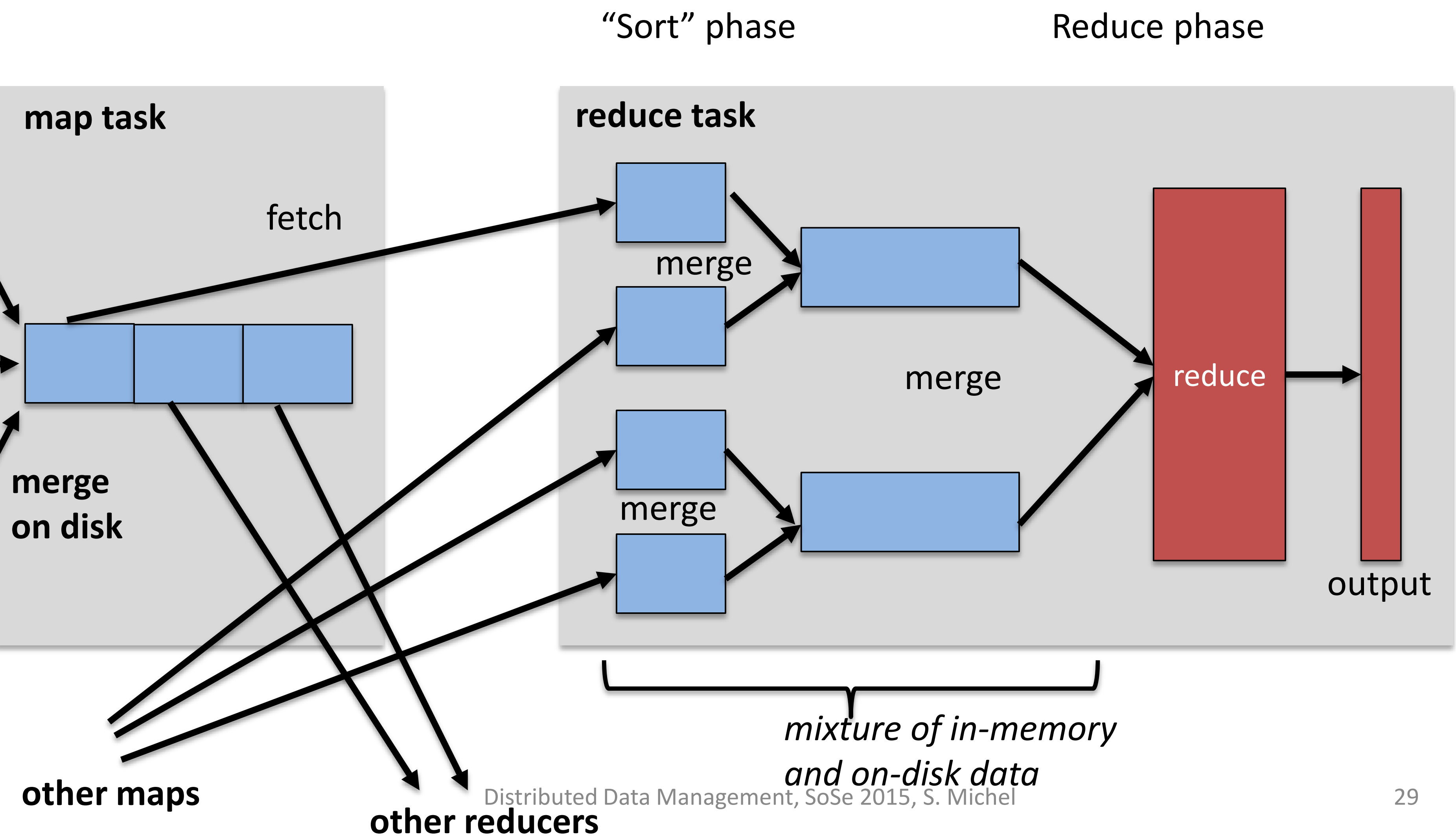
Shuffle and Sort: Illustration

- Buffer of Map output. Full? Partitioned and sorted -> disk (local); thus, multiple “spill files” for each partition.
- Are eventually merged (for each partition)



Shuffle and Sort: Illustration (Cont'd)

- Partitions (with same key) are gathered (from Map tasks) and merged.



Secondary Sort

- In MapReduce (Hadoop) tuples/records are sorted by key before reaching the reducers.
- For a single key, however, tuples are not sorted in any specific order (and this can also vary from one execution of the job to another).
- How can we impose a specific order?

Partitioning, Grouping, Sorting

- Consider weather data, temperature (temp) for each day. **Want: maximum temp per year**
- So, want data per year sorted by temp:

1900	35°C	max for year 1900
1900	34°C	
1900	34°C	
...		
1901	36°C	max for year 1901
1901	35°C	

- Idea: composite key: (year, temp)

example source: T. White, Hadoop, The Definitive Guide, 3rd edition

Partitioning, Grouping, Sorting (Cont'd)

- Obviously, doesn't work: (1900, 35°C) and (1900, 34°C) end up at different partitions
- Solution(?): Write **a custom partitioner** that considers year as partition and sort **comparator** for sorting by temperature

Need for Custom Grouping

- With that custom **partitioner by year** and still year and temp as key we get

		Partition	Group
1900	35°C		
1900	34°C		
1900	34°C		
…			
1901	36°C		
1901	35°C		

- Problem: reducer still consumes groups by key (within correct partitions)

Custom Grouping

- Solution: Define custom grouping method (class) that considers **year for grouping**

		Partition	Group
1900	35°C		
1900	34°C		
1900	34°C		
...			
1901	36°C		
1901	35°C		

Custom Sorting

- Finally, we provide a **custom sorting** that sorts the keys by temperature in descending order (= large values first)
- What happens then? Hadoop uses year for grouping (as said on previous slide), but which temp is used as the key (remember, we still have composite keys).
- The first one observed is used as key, i.e., the largest (max) temperature is used for the temp.

Note that this example specifically aims at computing the max using secondary sort. How would you implement a job such that the output is sorted by (year,temp) ?

Secondary Sort: Summary

- Recipe to get sorting by value
  - **Use composite key** of natural key and natural value
  - **Sort comparator has to order by the composite key** (i.e., both natural key and natural value)
  - **Partitioner and grouping comparator for the composite key should use only the natural key** for partitioning and grouping.

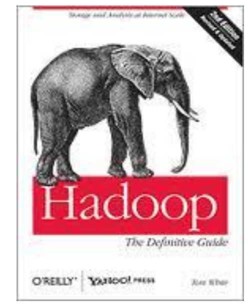
Hint (for Hadoop):

```
job.setMapperClass(…);
job.setPartitionerClass(…);
job.setSortComparatorClass(…);
job.setGroupingComparatorClass(…);
job.setReducerClass(…);
```

MR/Hadoop Literature

- Read on: hadoop.apache.org, there is also a tutorial

- Hadoop Book: Tom White. Hadoop: The definitive Guide. O'Reilly.



- Hadoop Illuminated: http://hadoopilluminated.com/hadoop_book/

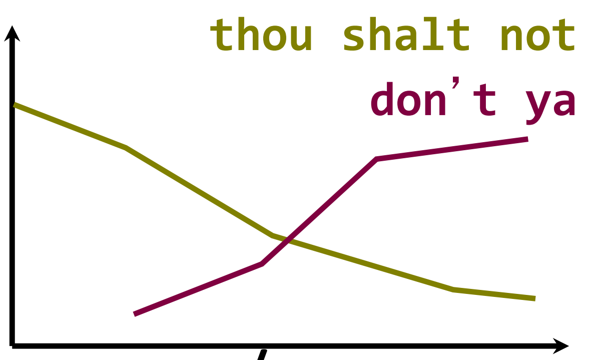
- Websites, e.g., http://bradhedlund.com/2011/09/10/understanding-hadoop-clusters-and-the-network/

- http://lintool.github.io/MapReduceAlgorithms/MapReduce-book-final.pdf

(MORE) DATA MANAGEMENT WITH MAPREDUCE

n- Grams

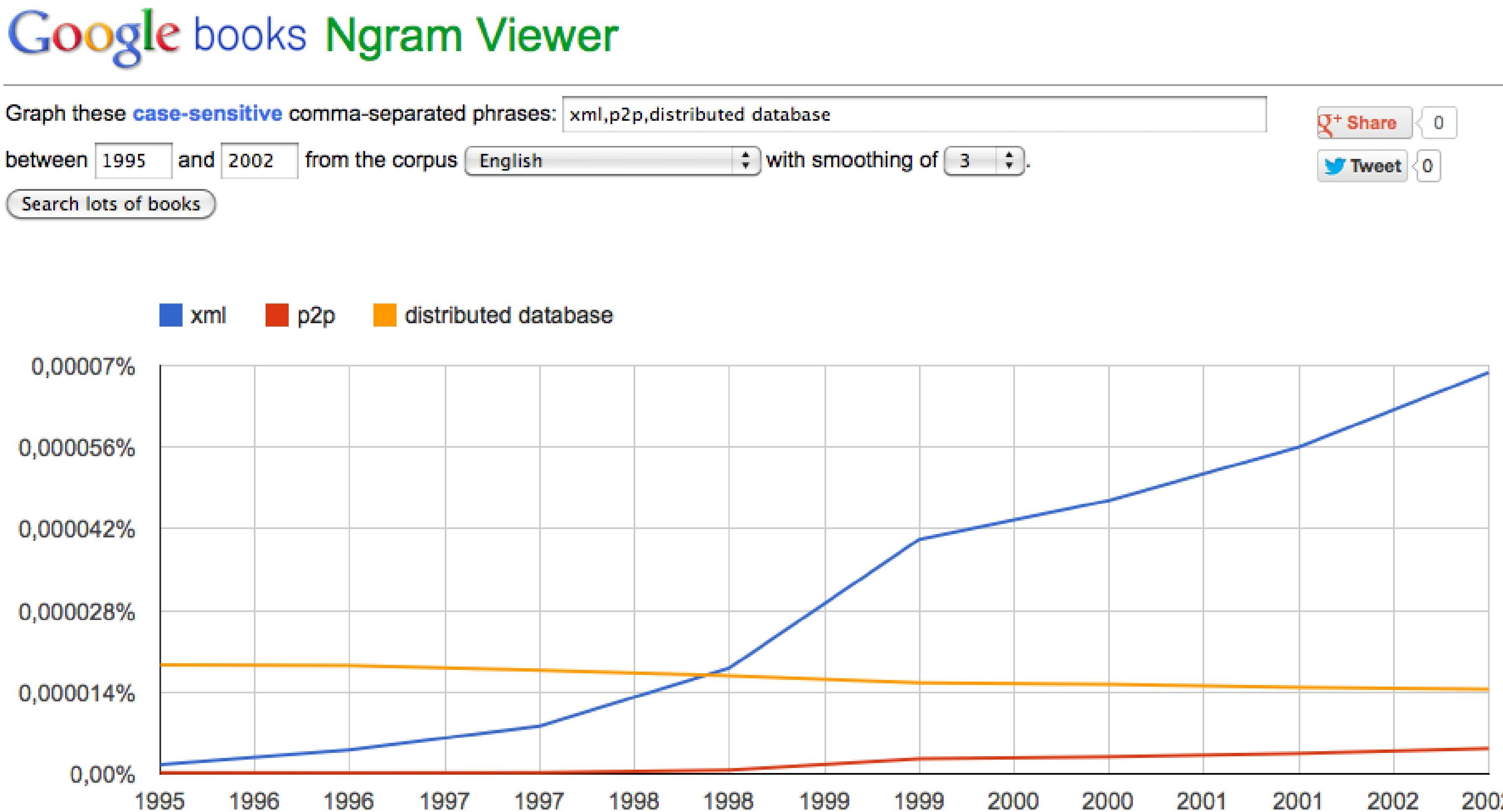
- Statistics about **variable-length word sequences** (contiguous)
  (e.g., `lord of the rings`, `at the end of`, …)
  have **many applications** in fields including
  - Information Retrieval
  - Natural Language Processing
  - Digital Humanities



- E.g., http://books.google.com/ngrams/
- A n-gram dataset is also available from there

n-gram slides based on a talk by Klaus Berberich

Example: Google Books Ngrams



n-grams Example

- **Document:** a x b b a y

  words

- **Possible n-grams:**
  - (a), (x), (b), (y)
  - (ax), (xb), (bb), …
  - (axb), (xbb), …
  - (axbb), (xbba), (bbay)
  - (axbba), (xbbay)
  - (axbbay)

Task: Computing n-grams in MR

- Given a set of documents.

- How can we efficiently compute n-grams, that
  - **occur at least τ times**
  - and **consist of at most σ words**

  using MapReduce?

Klaus Berberich, Srikanta J. Bedathur: Computing n-gram statistics in MapReduce. EDBT 2013:101-112

Naïve Solution: Simple Counting

Note: if a k-gram appears multiple times in the document, it is also emitted multiple times.

```
map(did, content):
  for k in <1 ... σ >:
      for all k-grams in content:
          emit(k-gram, did)


reduce(n-gram, list<did>):
  if length(list<did>) >=  τ:
     emit(n-gram, length(list<did>))
```

A Priori Based

- **(Famous) A priori Principle*:** *k*-gram can occur more than τ times only if its constituent (*k-1*)-grams occur at least τ times

(a,b,c) qualified only

if (b,c), (a,b) and (a), (b), (c)

How to implement?

**) Rakesh Agrawal, Tomasz Imielinski, Arun N. Swami: Mining Association Rules between Sets of Items in Large Databases. SIGMOD Conference 1993: 207-216*

A Priori Based (Cont'd)

- **Iterative Implementation:**
  - First 1-grams that occur τ times
  - Then 2-grams that occur τ times
  - …
- Needs **multiple MapReduce rounds** (of full data scans)
- **Already determined k-grams are kept**

Suffix Based

- **Emit only suffixes** in map phase
- Each of them **represents multiple *n*-grams** corresponding to its prefixes
  - **For instance, axbbay represents**
    - a, ax, axb, axbb, axbba, and axbbay

```
map(did, content):
    for all suffixes in
    content:
        emit(suffix, did)
```

Suffix Based: Partitioning

- Partition the suffixes **by first word**
  - to ensure all n-grams end up property for counting, that is:
    - all occurrences of *ax* have to end up at same reducer
    - suffix property: *ax* is only generated from suffixes that start with *ax*..

```
partition(suffix, did):
    return suffix[0] % m
```

Analogously with custom grouper.

Suffix Based: Sorting

- Reducer has to generate n-grams based on suffixes
  - read prefixes
  - count for each observed prefix its frequency
  - optimization: sort suffixes in reverse lexicographic order
  - then: simple counting using stack

aacd
aaca
aabx
aaba
aab
ax
.....

```
compare(suffix0, suffix1):
    return -strcmp(suffix0, suffix1)
```

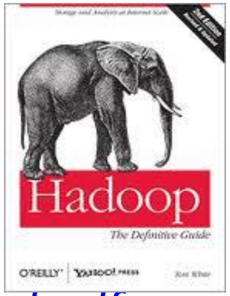
Discussion

- Let’s assess aforementioned algorithms with respect to properties like:
  - multiple MapReduce jobs vs. single job
  - amount of network traffic
  - ease of implementation

Literature

- Jeffrey Dean und Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters“. Google Labs.
- http://craig-henderson.blogspot.de/2009/11/dewitt-and-stonebrakers-mapreduce-major.html
- Klaus Berberich, Srikanta J. Bedathur: Computing n-gram statistics in MapReduce. EDBT 2013: 101-112
- Rakesh Agrawal, Tomasz Imielinski, Arun N. Swami: Mining Association Rules between Sets of Items in Large Databases. SIGMOD Conference 1993: 207-216
- S. Brin & L. Page. The anatomy of a large-scale hypertextual web search engine. In WWW Conf. 1998.
- Hadoop Book: Tom White. Hadoop: The definitive Guide.

  O’Reilly, 3rd edition.



- *Publicly available “book”:* [*http://lintool.github.io/MapReduceAlgorithms/MapReduce-book-final.pdf*](http://lintool.github.io/MapReduceAlgorithms/MapReduce-book-final.pdf)