Διάλεξη 21: Υπολογιστικές Στοίβες Μεγάλων Δεδομένων

Δημήτρης Ζεϊναλιπούρ

Lecture Outline

- Introduction to "Big-Data" Analytics
  - Example Scenarios and Architectures.
- Map-Reduce Programming Model
  - Other Map Reduce Data Processing Stacks
  - Map-Reduce Counting Problem
- Map-Reduce Architecture
  - Hadoop JobTracker, Tasktrackers and data-nodes
  - Failure Management
- Map-Reduce Optimizations
  - Combiners, Compression, In-Memory Shuffling, Speculative Execution
- Programming Map-Reduce
  - With Languages, PIG and in-the-cloud

Big-data Analytics

- Very large data collections (TB to PB) stored on distributed filesystems:
  - Query logs
  - Search engine indexes
  - Sensor data
- Need efficient ways for analyzing, reformatting, processing them
- In particular, we want:
  - Parallelization of computation (benefiting of the processing power of all nodes in a cluster)
  - Resilience to failure

Big-data Analytics (Example)

- We have a large file of words, one word to a line.
  - e.g., analyze web server logs for popular IPs

154.16.20.4
14.16.20.4
154.16.20.4
11.23.54.11

- Count the number of times each distinct word appears in the file
  - `sort datafile | uniq -c | sort -nk 2`

154.16.20.4	2
14.16.20.4	1
11.23.54.11	1

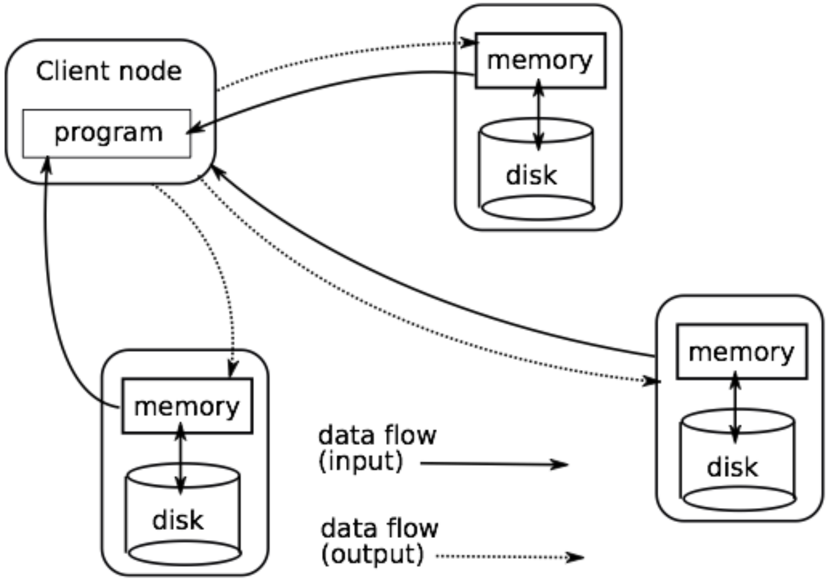
Scenario captures essence of MapReduce

Great thing is it is naturally parallelizable!

Big-data Analytics

Centralized computing with distributed data storage

Run the program at client node, get data from the distributed system.



Downsides: important data flows, no use of the cluster computing resources.

Distributed Process Management in UNIX

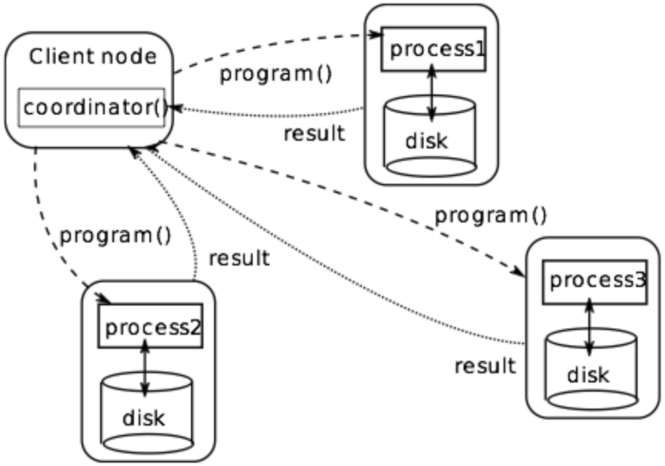
```
#!/bin/bash
COMMAND="ps -ef”
echo "Running $COMMAND"
for i in `cat hostnames.txt`
do
  # echo -n " $i"
  # assuming public/private key has been established
  ssh $i "$COMMAND > /tmp/file " &
  # echo "...Done"
done
echo "Waiting"
sleep 1
echo "Collecting Data"
for i in `cat hostnames.txt`
do
  # echo -n " $i”
  ssh $i "cat /tmp/file " &
  #echo "...Done"
done | awk -F" " '{print $1}' | sort | uniq
```

```
cat  hostnames.txt
b103ws1.in.cs.ucy.ac.cy
b103ws2.in.cs.ucy.ac.cy
b103ws3.in.cs.ucy.ac.cy
b103ws4.in.cs.ucy.ac.cy
b103ws5.in.cs.ucy.ac.cy
b103ws6.in.cs.ucy.ac.cy
b103ws7.in.cs.ucy.ac.cy
b103ws8.in.cs.ucy.ac.cy
b103ws9.in.cs.ucy.ac.cy
b103ws10.in.cs.ucy.ac.cy
b103ws11.in.cs.ucy.ac.cy
b103ws12.in.cs.ucy.ac.cy
b103ws13.in.cs.ucy.ac.cy
b103ws14.in.cs.ucy.ac.cy
b103ws15.in.cs.ucy.ac.cy
b103ws16.in.cs.ucy.ac.cy
```

Drawbacks ☹: No I/O Optimizations,
No Monitoring of Failures => No Fault Tolerance!

Map-Reduce Programming Model

Pushing the program near the data



- MapReduce: A programming model (inspired by standard functional programming operators) to facilitate the development and execution of distributed tasks.
- Published by Google Labs in 2004 at OSDI [DG04]. Widely used since then, open-source implementation in Hadoop.

Map-Reduce Programming Model

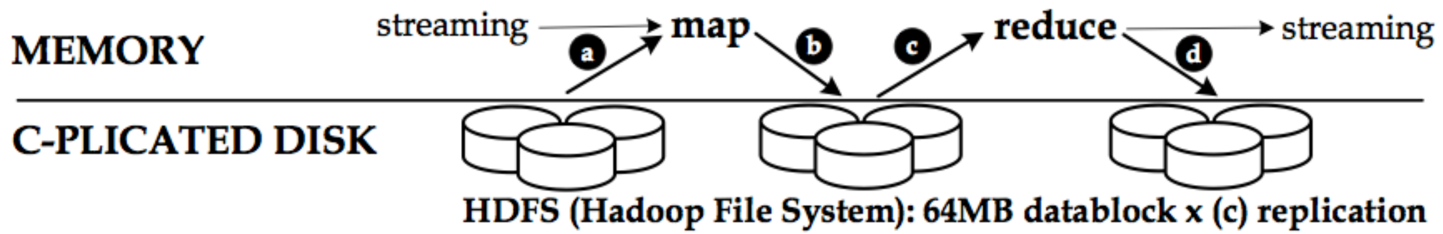
MapReduce in Brief

- The programmer defines the program logic as two functions:

  Map transforms the input into key-value pairs to process
  Reduce aggregates the list of values for each key

- The MapReduce environment takes in charge distribution aspects
- A complex program can be decomposed as a succession of Map and Reduce tasks
- Higher-level languages (Pig, Hive, etc.) help with writing distributed applications

Map-Reduce Programming Model



Map-Reduce Problem

Example: term count in MapReduce (input)

term	count
jaguar	5
mammal	1
family	3
available	1
...	

Count the distinct words in all documents

cat *.txt | sort | uniq -c

URL	Document
u_1	the jaguar is a new world mammal of the felidae family.
u_2	for jaguar, atari was keen to use a 68k family device.
u_3	mac os x jaguar is available at a price of us $199 for apple's new "family pack".
u_4	one such ruling family to incorporate the jaguar into their name is jaguar paw.
u_5	it is a big cat.

1 TB on 1 PC = 2 hours!!!

1TB on 100 PCs = 1min!!!

Map-Reduce Example

Example: term count in MapReduce

Example uses 1 mapper / 1 reduce only!

Map

$\text{list}(K', V')$

term	count
jaguar	1
mammal	1
family	1
jaguar	1
available	1
jaguar	1
family	1
family	1
jaguar	2
...	

Shuffle

$(K', \text{list}(V'))$

term	count
jaguar	1,1,1,2
mammal	1
family	1,1,1
available	1
...	

Reduce

$\text{list}(K'', V'')$

term	count
jaguar	5
mammal	1
family	3
available	1
...	

final

Map-Reduce Programming Model

1. User-defined: *map* : $(K, V) \rightarrow \text{list}(K', V')$

(dumping)

```
function map(uri, document)
  foreach distinct term in document
    output (term, count(term, document))
```

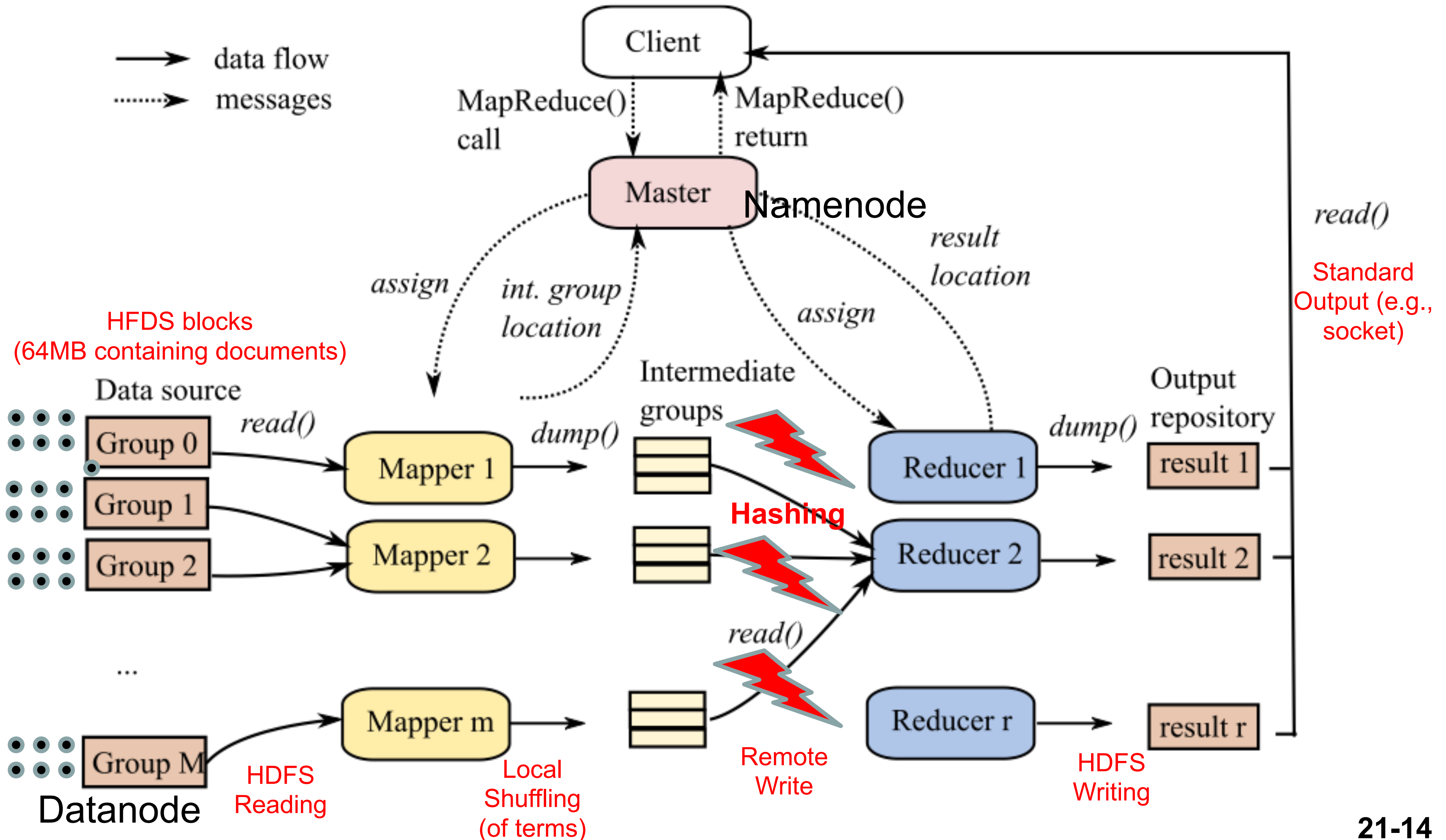
2. Fixed behavior: *shuffle* : $\text{list}(K', V') \rightarrow \text{list}(K', \text{list}(V'))$ regroups all intermediate pairs on the key **(hashing / sorting)**

3. User-defined: *reduce* : $(K', \text{list}(V')) \rightarrow \text{list}(K'', V'')$

```
function reduce(term, counts)
  output (term, sum(counts))
```

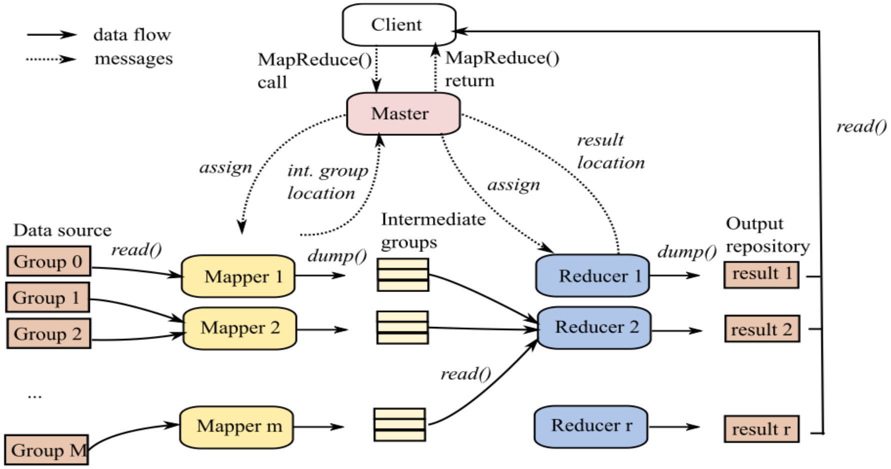
(grouping)

Map-Reduce Architecture (e.g., in Hadoop)



Map-Reduce Architecture (e.g., in Hadoop)

A MapReduce cluster



Nodes inside a MapReduce cluster are decomposed as follows:

- A jobtracker acts as a master node; MapReduce jobs are submitted to it
- Several tasktrackers run the computation itself, i.e., *map* and *reduce* tasks
- A given tasktracker may run several tasks in parallel
- Tasktrackers usually also act as data nodes of a distributed filesystem (e.g., GFS, HDFS)

+ a client node where the application is launched.

Map-Reduce Architecture (Processing Remarks)

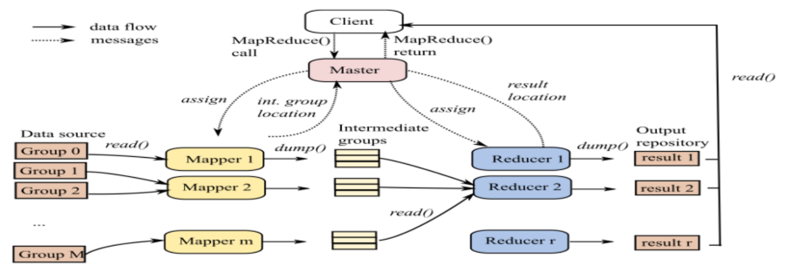
Processing a MapReduce job

A MapReduce job takes care of the distribution, synchronization and failure handling. Specifically:

- the input is split into *M* groups; each group is assigned to a mapper (assignment is based on the data locality principle)
- each mapper processes a group and stores the intermediate pairs locally
- grouped instances are assigned to reducers thanks to a hash function
- (*shuffle*) intermediate pairs are sorted on their key by the reducer
- one obtains grouped instances, submitted to the *reduce* function

Remark: the data locality does no longer hold for the *reduce* phase, since it reads from the mappers.

Map-Reduce Architecture (Failure Management)



Failure management

In case of failure, because the tasks are distributed over hundreds or thousands of machines, the chances that a problems occurs somewhere are much larger; starting the job from the beginning is not a valid option.

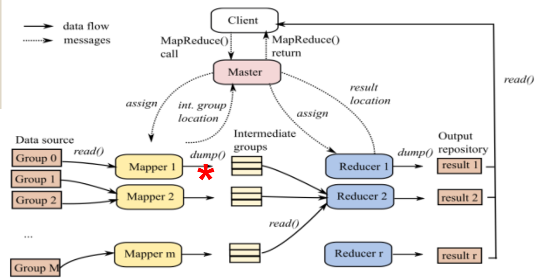
The Master periodically checks the availability and reachability of the tasktrackers (heartbeats) and whether *map* or *reduce* jobs make any progress

1. if a reducer fails, its task is reassigned to another tasktracker; this usually require restarting mapper tasks as well (to produce intermediate groups)
2. if a mapper fails, its task is reassigned to another tasktracker
3. if the jobtracker fails, the whole job should be re-initiated

"ZooKeeper: Wait-free coordination for Internet-scale systems", Hunt et al., USENIX 2010, http://static.usenix.org/event/usenix10/tech/full_papers/Hunt.pdf

YARN brings real failure management to the Hadoop 2 ecosystem

Map-Reduce Optimizations (Combiners)



Combiners

- A mapper task can produce a large number of pairs with the same key
- They need to be sent over the network to the reducer: costly
- It is often possible to combine these pairs into a single key-value pair

Example

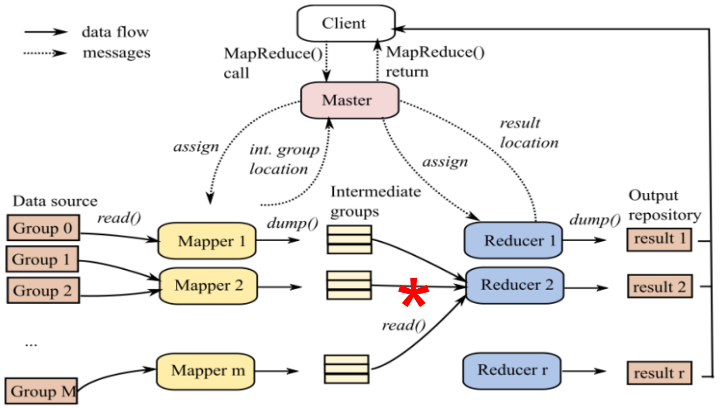
(jaguar,1), (jaguar, 1), (jaguar, 1), (jaguar, 2)$\rightarrow$(jaguar, 5)

- *combiner* : list(V') $\rightarrow$ V' function executed (possibly several times) to combine the values for a given key, on a mapper node
- No guarantee that the *combiner* is called
- Easy case: the combiner is the same as the *reduce* function. Possible when the aggregate function α computed by *reduce* is distributive: $\alpha(k_1, \alpha(k_2, k_3)) = \alpha(k_1, k_2, k_3)$ **Distributive: COUNT, MIN, MAX, SUM**

Won't work with Holistic functions: MEDIAN, RANK (all are necessary)

Map-Reduce Optimizations (Compression)

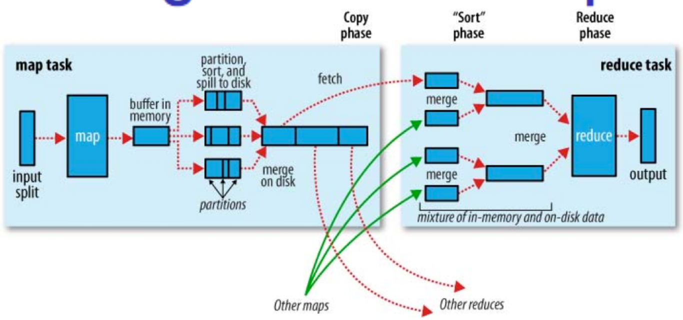
Compression

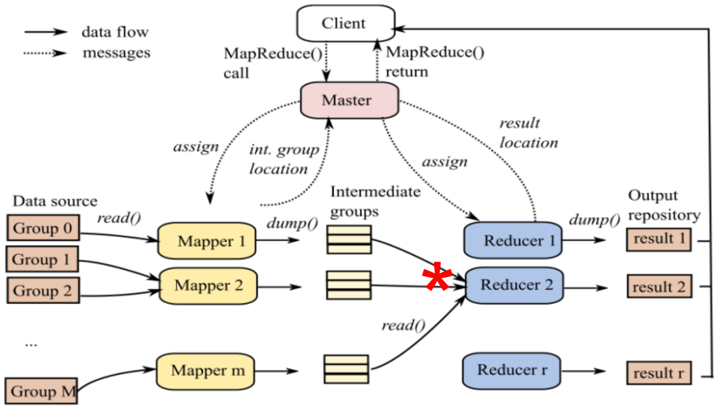


- Data transfers over the network:
  - From datanodes to mapper nodes (usually reduced using data locality)
  - From mappers to reducers
  - From reducers to datanodes to store the final output
- Each of these can benefit from data compression
- Tradeoff between volume of data transfer and (de)compression time
- Usually, compressing *map* outputs using a fast compressor increases efficiency

Map-Reduce Optimizations (Shuffling in Memory)

Optimizing the *shuffle* operation

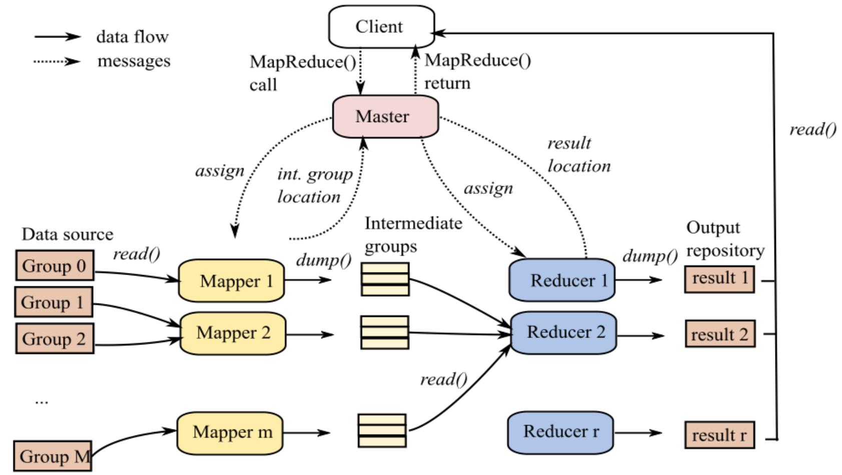




- Sorting of pairs on each reducer, to compute the groups: costly operation
- Sorting much more efficient in memory than on disk
- Increasing the amount of memory available for *shuffle* operations can greatly increase the performance
- . . . at the downside of less memory available for *map* and *reduce* tasks (but usually not much needed)

Map-Reduce Optimizations (Speculative Execution)

Speculative execution



- The MapReduce jobtracker tries detecting tasks that take longer than usual (e.g., because of hardware problems)
- When detected, such a task is speculatively executed on another tasktracker, without killing the existing task
- Eventually, when one of the attempts succeeds, the other one is killed

MapReduce in Hadoop (MR => HADOOP => HBASE)

- ***Map-Reduce:*** *a **programming model for processing** large data sets.*
  - *Invented by Google! "MapReduce: Simplified Data Processing on Large Clusters, Jeffrey Dean and Sanjay Ghemawat, OSDI'04: Sixth Symposium on Operating System Design and Implementation,San Francisco, CA, December, 2004."*
  - *Can be implemented in any language (recall javascript Map-Reduce we used in the context of CouchDB).*
- ***Hadoop:*** *Apache's open-source **software framework** that supports **data-intensive distributed applications***
  - *Derived from Google's MapReduce + Google File System (GFS) papers. (Input by Yahoo!, Facebook, etc.)*
  - *Enables applications to work with thousands of computation-independent computers and petabytes of data.*
  - Download: http://hadoop.apache.org/

MapReduce in Hadoop (Who is driving Hadoop?)

Hadoop PMC

The Hadoop Project Management Committee contains (in alphabetical order):

username	name	organization	roles
acmurthy	Arun C Murthy	Hortonworks	
amareshwari	Amareshwari Sriramadasu	InMobi	
atm	Aaron T. Myers	Cloudera	
bobby	Robert(Bobby) Evans	Yahoo!	
cdouglas	Chris Douglas	Microsoft	
cutting	Doug Cutting	Cloudera	
ddas	Devaraj Das	Hortonworks	
dhruba	Dhruba Borthakur	Facebook	
eli	Eli Collins	Cloudera	
enis	Enis Soztutar	Hortonworks	
gkesavan	Giridharan Kesavan	Hortonworks	
hairong	Hairong Kuang	Facebook	
jghoman	Jakob Homan	LinkedIn	
jitendra	Jitendra Nath Pandey	Hortonworks	
mahadev	Mahadev Konar	Hortonworks	
mattf	Matt Foley	Hortonworks	
nigel	Nigel Daley	Jive	
omalley	Owen O'Malley	Hortonworks	
phunt	Patrick Hunt	Cloudera	ZooKeeper
rangadi	Raghu Angadi	Twitter	
sharad	Sharad Agarwal	InMobi	
shv	Konstantin Shvachko		HDFS
sradia	Sanjay Radia	Hortonworks	
sseth	Siddharth Seth	Hortonworks	
stack	Michael Stack	StumbleUpon	HBase
suresh	Suresh Srinivas	Hortonworks	
szetszwo	Tsz Wo (Nicholas) Sze	Hortonworks	
tgraves	Thomas Graves	Yahoo!	
todd	Todd Lipcon	Cloudera	
tomwhite	Tom White	Cloudera	
tucu	Alejandro Abdelnur	Cloudera	
vinodkv	Vinod Kumar Vavilapalli	Hortonworks	
yhemanth	Hemanth Yamijala		
zshao	Zheng Shao	Facebook	

https://hadoop.apache.org/who.html

MapReduce in Hadoop (MR => HADOOP => HBASE)

- ***Hadoop Project Modules:***
  - ***Hadoop Common:*** *The common utilities that support the other Hadoop modules.*
  - ***Hadoop Distributed File System (HDFS™):*** *A distributed file system that provides high-throughput access to application data.*
  - ***Hadoop YARN (Yet Another Resource Negotiator):*** *A framework for job scheduling and cluster resource management.*
  - ***Hadoop MapReduce (MapReduce v2.0):*** *A YARN-based system for parallel processing of large data sets.*
- ***Other Hadoop-related projects at Apache include:***
  - **Ambari™**: A web-based tool for provisioning, managing, and monitoring Apache Hadoop clusters **Avro™**: A data serialization system.
  - **Cassandra™**: A scalable multi-master database with no single points of failure.
  - **Chukwa™**: A data collection system for managing large distributed systems.
  - **HBase™**: A scalable, distributed database that supports structured data storage for large tables.
  - **Hive™**: A data warehouse infrastructure that provides data summarization and ad hoc querying.
  - **Mahout™**: A Scalable machine learning and data mining library.
  - **Pig™**: A high-level data-flow language and execution framework for parallel computation.
  - **Spark™**: A fast and general compute engine for Hadoop data.
  - **Tez™**: A generalized data-flow programming framework, built on Hadoop YARN,
  - **ZooKeeper™**: A high-performance coordination service for distributed applications.

Programming with Hadoop (with Languages)

Hadoop programming interfaces

- Different APIs to write Hadoop programs:
  - A rich Java API (main way to write Hadoop programs)
  - A Streaming API that can be used to write *map* and *reduce* functions in any programming language (using standard inputs and outputs)
  - A C++ API (Hadoop Pipes)
  - With a higher-language level (e.g., Pig, Hive)
- Advanced features only available in the Java API
- Two different Java APIs depending on the Hadoop version; presenting the "old" one

Programming with Hadoop (in the Cloud!)

Hadoop in the cloud

- Possibly to set up one's own Hadoop cluster
- But often easier to use clusters in the cloud that support MapReduce:
  - Amazon EC2
  - Cloudera
  - etc.
- Not always easy to know the cluster's configuration (in terms of racks, etc.) when on the cloud, which hurts data locality in MapReduce

Amazon Elastic MapReduce (Amazon EMR)

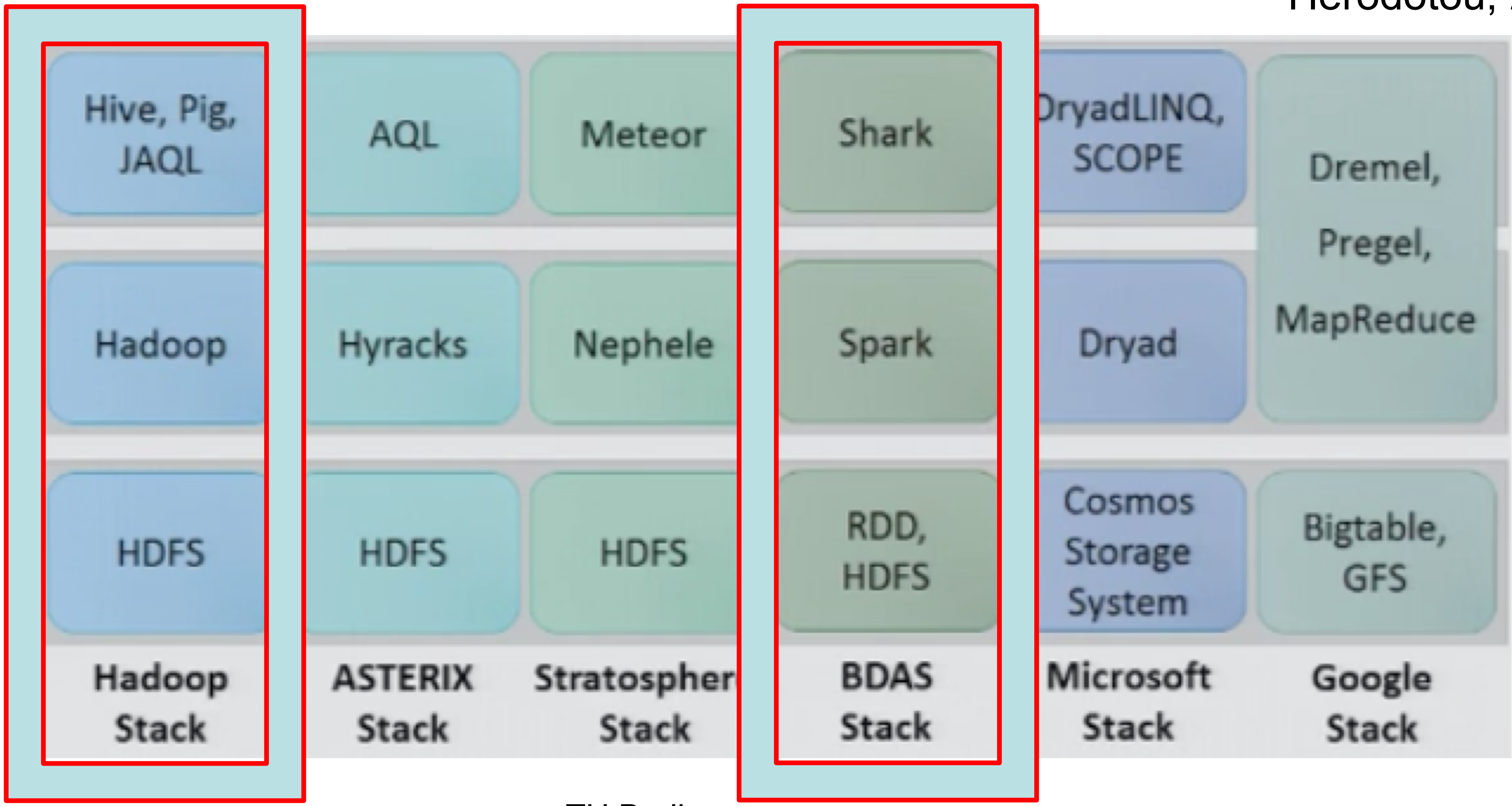
Amazon Elastic MapReduce (Amazon EMR) is a web service that enables businesses, researchers, data analysts, and developers to easily and cost-effectively process vast amounts of data. It utilizes a hosted Hadoop framework running on the web-scale infrastructure of Amazon Elastic Compute Cloud (Amazon EC2) and Amazon Simple Storage Service (Amazon S3).

Hadoop Cloud Issues & Costs

- The public cloud storage & deep storage might not be an option if legislative barriers exists (e.g., privacy & security concerns)
- A Hadoop Cluster using between **125-250 nodes** is projected to cost **~1M USD** per year to be operational!
- The amount of **storage doubles every year** but the storage media only **decline** at a **rate of less** than **1/5 per year.**
  - We need ways to decay large volumes of data

Modern Data Processing Stacks

Herodotou, 2013



Apache

UCI & UCR

TU Berlin =>Apache

UC Berkeley => Apache

What is Spark?

- Fast, expressive cluster computing system compatible with Apache Hadoop
  - Works with any Hadoop-supported storage system (HDFS, S3, Avro, …)
- Improves **efficiency** through:
  - In-memory computing primitives
  - General computation graphs
- Improves **usability** through:
  - Rich APIs in Java, Scala, Python
  - Interactive shell

→ Up to 100 × faster

→ Often 2-10 × less code