

## Principles of Computer Science II

### Large Scale Computation

Ioannis Chatzigiannakis

Sapienza University of Rome

#### Lecture 22



## Problem: Lots of data

- ▶ Example: Homo sapiens high coverage assembly GRCh37
  - ▶ 27478 contigs
  - ▶ contig length total 3.2 Gb.
  - ▶ chromosome length total 3.1 Gb.
- ▶ One computer can read 30-35MB/sec from disc
  - ▶ ~ 10 months to read the data
- ▶ ~ 100 hard drives just to store the data in compressed format
- ▶ Even more to do something with the data.



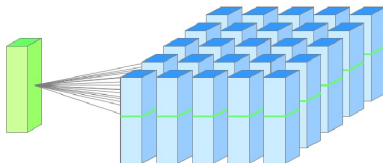
## Spread the work over many machines

- ▶ Good news: same problem with 1000 machines:  $\leq 1$  hour
- ▶ Bad news: concurrency
  - ▶ communication and coordination
  - ▶ recovering from machine failure
  - ▶ status reporting
  - ▶ debugging
  - ▶ optimization
- ▶ Bad news 2: repeat for every problem you want to solve



## Computing Clusters

- ▶ Many racks of computers
- ▶ Thousands of machines per cluster
- ▶ Limited bandwidth between racks



## Computing Environment

- ▶ Each machine has 2-4 CPUs
  - ▶ Typically quad-core
  - ▶ Future machines will have more cores
- ▶ 1-6 locally-attached disks
  - ▶ ~ 10TB of disk
- ▶ Overall performance more important than peak performance of single machines
- ▶ Reliability
  - ▶ In 1 server environment, it may stay up for three years (1000 days)
  - ▶ If you have 10000 servers, expect to lose 10 each day
- ▶ Ultra reliable hardware still fails
  - ▶ We need to keep in mind cost of each machine



## Map Reduce Computing Paradigm

- ▶ A simple programming model
  - ▶ Applies to large-scale computing problems
- ▶ Hides difficulties of concurrency
  - ▶ automatic parallelization
  - ▶ load balancing
  - ▶ network and disk transfer optimization
  - ▶ handling of machine failures
  - ▶ robustness
  - ▶ improvements to core libraries benefit all users of library



## A typical problem

- ▶ Read a lot of data
- ▶ **Map**: extract something important from each record
- ▶ Shuffle and sort
- ▶ **Reduce**: aggregate, summarize, filter or transform
- ▶ Write the results



## In more details

- ▶ Programmer specifies two primary methods:
  - ▶  $\text{map}(k, v) \rightarrow \langle k', v' \rangle *$ 
    - ▶ Takes a key-value pair and outputs a set of key-value pairs
    - ▶ There is one Map call for every  $(k, v)$  pair
  - ▶  $\text{reduce}(k', \langle v' \rangle *) \rightarrow \langle k', v' \rangle *$ 
    - ▶ All values  $v'$  with same key  $k'$  are reduced together and processed in  $v'$  order
    - ▶ There is one Reduce function call per unique key  $k'$
- ▶ All  $v'$  with same  $k'$  are reduced together, in order.



## An example: Frequencies in DNA sequence

A typical exercise for a new engineer in his/her first week:

- ▶ Input files with one document per record
- ▶ Specify a **map** function that takes a key/value pair
  - ▶ key = document URL
  - ▶ value = document contents
- ▶ Output of map function is (potentially many) key/value pairs.
- ▶ In this case, output:  
(word, 1) once per word in the document

"document 1", "CTGGGCTAA"

converted to

(C, 1), (T, 1), (G, 1), ...



## An example: Frequencies in DNA sequence

- ▶ MapReduce library gathers together all pairs with the same key (shuffle/sort)
- ▶ The **reduce** function combines the values for a key
- ▶ In this example:

key = "A"	key = "G"	key = "C"	key = "T"
values = 1, 1	values = 1, 1, 1	values = 1, 1	values = 1, 1
summarize	summarize	summarize	summarize
2	3	2	2

- ▶ Output of reduce paired with key and saved

(A, 3), (G, 3), (C, 2), (T, 2)



## An example: Frequencies in DNA sequence

```
s = 'CTGGGCTAA'  
seq = list(s) # ['C', 'T', 'G', 'G', 'G', 'C', 'T', 'A', 'A']  
sc.parallelize(seq)\  
  .map(lambda symbol: (symbol, 1))\  
  .reduceByKey(add)\  
  .collect()
```

Output:

```
[('A', 2), ('C', 2), ('G', 3), ('T', 2)]
```



## Fault tolerance: handled via re-execution

- ▶ On worker failure:
  - ▶ Detect failure via periodic heartbeats
  - ▶ Re-execute completed and in-progress map tasks
  - ▶ Re-execute in progress reduce tasks
  - ▶ Task completion committed through master
- ▶ On master failure:
  - ▶ Restart execution



## AWS Elastic Map Reduce

- ▶ Managed Hadoop framework on EC2 instances.
- ▶ AWS EMR splits large processing jobs into smaller jobs and distributes them across many compute nodes in a Hadoop cluster.
- ▶ Easily run and scale open-source big data frameworks:
  - ▶ Apache Spark
  - ▶ Apache Flink
  - ▶ Apache Hive
  - ▶ Presto
  - ▶ Apache HBase
  - ▶ ...
- ▶ EMR Notebooks.

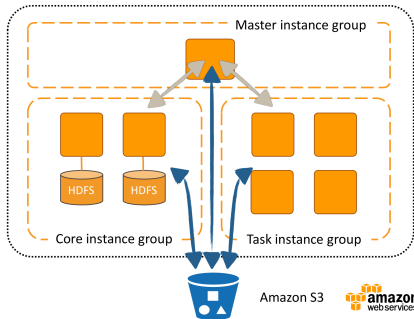


## EMR: Benefits

- ▶ Easy to use – interact using Jupyter via web.
- ▶ Low cost
  - ▶ Pay a per-instance rate for every second used, with a one-minute minimum charge.
- ▶ Elastic
  - ▶ For short-running jobs, you can spin up and spin down clusters and pay per second for the instances used.
  - ▶ For long-running workloads, you can create highly available clusters that automatically scale to meet demand.
- ▶ Reliable
- ▶ Secure
- ▶ Flexible



## Apache EMR Architecture



## AWS EMR and Apache Hadoop

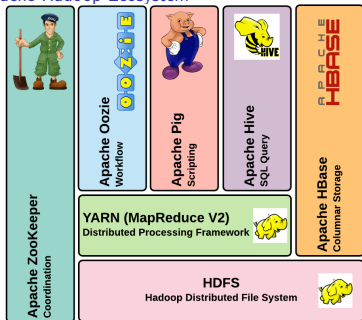
- ▶ The Elastic Map Reduce is built on top Apache Hadoop.
- ▶ An open-source Java software framework that supports massive data processing across a cluster of instances.
- ▶ Distributed processing across the instances that make up the cluster.
- ▶ It can run on a single instance or thousands of instances.
- ▶ Elastic auto-scaling of cluster.
- ▶ Provides a fault-tolerant processing environment.



## Apache Hadoop

- ▶ Apache Hadoop includes the following 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**: A framework for job scheduling and cluster resource management.
  - ▶ **Hadoop MapReduce**: A YARN-based system for parallel processing of large data sets.
  - ▶ **Hadoop Ozone**: An object store for Hadoop.

## Apache Hadoop Ecosystem



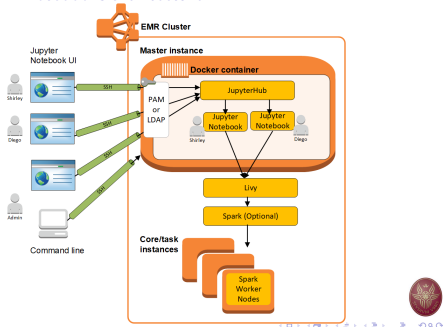
## Apache Spark on AWS EMR

- ▶ Started in 2009 as a research project at UC Berkley's AMPLab.
- ▶ An open-source, distributed processing system used for big data workloads.
- ▶ In contrast to Hadoop, uses in-memory caching to achieve high speed-ups.
  - ▶ Optimized query execution for fast analytic queries against data of any size.
- ▶ Development APIs in Java, Scala, Python and R.
- ▶ Supports code reuse across multiple workloads-batch processing:
  - ▶ interactive queries, real-time analytics, machine learning, and graph processing.

## EMR Notebooks

- ▶ EMR Notebooks is a Jupyter Notebook environment built in to the Amazon EMR console.
- ▶ Quickly create Jupyter notebooks, attach them to Spark clusters
- ▶ Use Jupyter Notebook editor to remotely run queries and code.
- ▶ Open, attach multiple notebooks to a single cluster, and re-use a notebook on different clusters.
- ▶ You can start a cluster, attach an EMR notebook for analysis, and then terminate the cluster.
- ▶ You can also close a notebook attached to one running cluster and switch to another.
- ▶ Multiple users can attach notebooks to the same cluster simultaneously.

## EMR Notebooks architecture



## Considerations when using EMR Notebooks

- ▶ User notebooks and files are saved to the file system on the master node.
- ▶ This is ephemeral storage that does not persist through cluster termination.
- ▶ **When a cluster terminates, this data is lost if not backed up.**
- ▶ EMR Notebooks support persistence to S3.
- ▶ EMR Notebooks supports connection with GitHub repositories.

## Create Cluster

The screenshot shows the Amazon EMR console 'Create Cluster' page. The 'Welcome to Amazon Elastic MapReduce' section is visible. The 'Create Cluster' button is highlighted with a red circle. The 'How Elastic MapReduce Works' section is also visible, showing the flow from Upload to Create to Monitor.

## Quick Options

The screenshot shows the Amazon EMR console 'Quick Options' page. The 'Create Cluster - Quick Options' button is highlighted with a red circle. The 'General configuration' section is visible, showing the cluster name, logging, and launch mode. The 'Software configuration' section is also visible, showing the selected applications and software stack.

# Software Configuration

Create Cluster - Advanced Options [Go to quick options](#)

## Step 3: Software and Steps

Step 2: Hardware

Step 3: General Cluster Settings

Step 4: Security

### Software Configuration

AMI ID:

OS:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

AMI:

### Multi-Region (Optional)

Use multiple master nodes to improve cluster availability. [Learn more](#)

AWS Glue Data Catalog settings (optional)

Use for Hive table metadata

Use for Spark table metadata

Use for Tez table metadata

Use for Hadoop table metadata

Use for Pig table metadata

Use for Hive table metadata

Use for Tez table metadata

Use for Hadoop table metadata

Use for Pig table metadata

Use for Hive table metadata

Use for Tez table metadata

Use for Hadoop table metadata

Use for Pig table metadata

Use for Hive table metadata

Use for Tez table metadata

Use for Hadoop table metadata

Use for Pig table metadata

Use for Hive table metadata

Use for Tez table metadata

Use for Hadoop table metadata

Use for Pig table metadata

Use for Hive table metadata

Use for Tez table metadata

Use for Hadoop table metadata

Use for Pig table metadata

Use for Hive table metadata

Use for Tez table metadata

Use for Hadoop table metadata

Use for Pig table metadata

Use for Hive table metadata

Use for Tez table metadata

Use for Hadoop table metadata

Use for Pig table metadata

Use for Hive table metadata

Use for Tez table metadata

Use for Hadoop table metadata

Use for Pig table metadata

Use for Hive table metadata

Use for Tez table metadata

Use for Hadoop table metadata

Use for Pig table metadata

Use for Hive table metadata

Use for Tez table metadata

Use for Hadoop table metadata

Use for Pig table metadata

Use for Hive table metadata

Use for Tez table metadata

Use for Hadoop table metadata

Use for Pig table metadata

Use for Hive table metadata

Use for Tez table metadata

Use for Hadoop table metadata

Use for Pig table metadata

Use for Hive table metadata

Use for Tez table metadata

Use for Hadoop table metadata

Use for Pig table metadata

Use for Hive table metadata

Use for Tez table metadata

Use for Hadoop table metadata

Use for Pig table metadata

Use for Hive table metadata

Use for Tez table metadata

Use for Hadoop table metadata

Use for Pig table metadata

Use for Hive table metadata

Use for Tez table metadata

Use for Hadoop table metadata

Use for Pig table metadata

Use for Hive table metadata

Use for Tez table metadata

Use for Hadoop table metadata

### Steps (optional)

A step is a unit of work you submit to the cluster. For instance, a step might contain one or more Hadoop or Spark jobs. You can also submit additional steps to a cluster after it is running. [Learn more](#)

Concurrency:  Run multiple steps at the same time to improve cluster utilization

After last step completes:  Clusters enters waiting state

Cluster auto-terminates

Step type:  [Add step](#)

# Hardware Configuration

Create Cluster - Advanced Options [Go to quick options](#)

## Step 1: Software and Steps

Step 2: Hardware

Step 3: General Cluster Settings

Step 4: Security

### Hardware Configuration

Specify the networking and hardware configuration for your cluster. Request Spot instances (optional ECR) to save money.

#### Cluster Composition

Specify the configuration of the master, core and task nodes as an instance group or instance fleet. This choice applies to all nodes for the lifetime of the cluster. Instance fleets and instance groups cannot coexist in a cluster. [Get ECR Spot](#)

##### Instance group configuration

Define instance groups

Specify a single instance type and purchasing option for each node type.

Instance fleets

Specify target capacity and low instance cost nodes for each node type. The instance type and purchasing options. [Learn more](#)

#### Networking

Use a Virtual Private Cloud (VPC) to process sensitive data or connect to a private network. Launch the cluster into a VPC with a public, private or shared subnet. Subnets may be associated with and AWS Outposts on AWS Local Zones.

Launch the cluster into a VPC with a public, private, or shared subnet. Subnets may be associated with an AWS Outposts or AWS Local Zone.

Network:

EC2 Subnet:

#### Cluster Nodes and Instances

Choose the instance type, number of instances, and a purchasing option. [Learn more about instance auto-scaling options](#)

Instance options for an Amazon Machine Image changed. [Learn more](#)

Node type	Instance type	Instance count	Purchasing option
Master	m3.xlarge	1	On-demand

# Choose Instance Types

Choose the instance type, number of instances, and a purchasing option. [Learn more about instance auto-scaling options](#)

Instance options for an Amazon Machine Image changed. [Learn more](#)

Node type	Instance type	Instance count	Purchasing option
Master	m3.xlarge	1	On-demand
Core	m3.xlarge	2	On-demand
Task	m3.xlarge	2	On-demand

#### Cluster scaling

Adjust the number of Amazon EC2 instances available to the cluster via IAM-managed scaling or a custom automatic scaling policy. [Learn more](#)

Cluster scaling:  Enable Cluster Scaling

#### EBS Root Volume

Specify the root device volume size up to 1 TB. This setting applies to all instances in the cluster. [Learn more](#)

Root device EBS volume size:  GB

# General Cluster Settings

Create Cluster - Advanced Options [Go to quick options](#)

## Step 1: Software and Steps

Step 2: Hardware

Step 3: General Cluster Settings

Step 4: Security

### General Options

Cluster name:

Logins

ECR Spot:

Log interruption

Reliability

Termination protection

#### Tags

Key

Value (optional)

#### Additional Options

EMRFS consistent view

Custom AMI ID:

#### Routing Actions





## Link Git repository to notebook

The screenshot shows the 'Create notebook' page in the Amazon EMR console. The 'Name and configure your notebook' section is active. A modal dialog titled 'Link Git repository to notebook' is open, displaying a search for Git repositories. The search results show a repository named 'git' with the URL 'https://github.com/git/git' and the branch 'master'. The 'Repository name' is set to 'git' and the 'URL' is 'https://github.com/git/git'. The 'Branch' is set to 'master'. The 'Link Git repository to notebook' button is highlighted.

## Add repository

The screenshot shows the 'Add repository' page in the Amazon EMR console. The 'Repository name' is set to 'git'. The 'Git repository URL' is 'https://github.com/git/git'. The 'Branch' is set to 'master'. The 'Git credentials' section is expanded, showing the 'Use an existing AWS account' option selected. The 'Create a new account' option is also visible. The 'Use name and password' option is selected, with the 'Username' set to 'john@github.com' and the 'Password' field filled. The 'Personal access token (PAT)' option is also visible. The 'Add repository' button is highlighted.

## Link Git repository to notebook

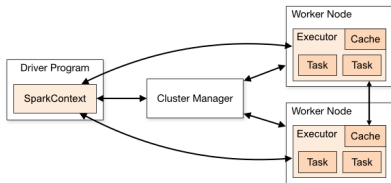
The screenshot shows the 'Create notebook' page in the Amazon EMR console. The 'Name and configure your notebook' section is active. A modal dialog titled 'Link Git repository to notebook' is open, displaying a search for Git repositories. The search results show a repository named 'git' with the URL 'https://github.com/git/git' and the branch 'master'. The 'Repository name' is set to 'git' and the 'URL' is 'https://github.com/git/git'. The 'Branch' is set to 'master'. The 'Link Git repository to notebook' button is highlighted.

## Create notebook

The screenshot shows the 'Create notebook' page in the Amazon EMR console. The 'Name and configure your notebook' section is active. The 'Notebook name' is 'my-notebook'. The 'Character' is set to 'Create an existing cluster'. The 'Security group' is set to 'sg-0123456789'. The 'AWS service role' is 'EMR\_Notebook\_DefaultRole'. The 'Notebook location' is set to 'Use a location that EMR creates'. The 'Link to a Git repository' section is expanded, showing the 'git' repository selected. The 'Link Git repository to notebook' button is highlighted.



## Spark Driver Program



- ▶ Map/Reduce operations are issued to the cluster manager.
- ▶ Map/Reduce operations work on a given dataset.
- ▶ The dataset is encoded using the RDD structure.



## Spark Context

- ▶ SparkContext is the entry point to any spark functionality.
- ▶ A SparkContext represents the connection to a Spark cluster.
- ▶ Used to create RDD and broadcast variables on that cluster.
- ▶ Only one SparkContext should be active per session.

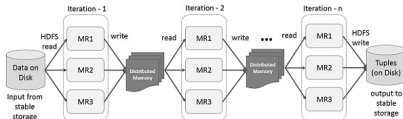


## Resilient Distributed Datasets

- ▶ A fundamental data structure of Spark.
- ▶ Spark makes use RDD to achieve faster and efficient MapReduce operations.
- ▶ An **immutable** distributed **read-only** collection of objects.
  - ▶ immutable = state cannot change after it is constructed.
  - ▶ Can contain any type of Python, Java, or Scala objects, including user-defined classes.
- ▶ Two ways to construct an RDD:
  1. Referencing a dataset in an external storage system: S3, HDFS, HBase, ...
  2. Through Map/Reduce operations.
- ▶ RDD is divided into logical partitions.
  - ▶ Each logical partition may be computed on different nodes of the cluster.



## Iterative Operations on MapReduce



- ▶ Reuse intermediate results across multiple computations in multi-stage applications.
- ▶ Each Map/Reduce operation works on a given/input RDD.
- ▶ Each Map/Reduce operation constructs/outputs a new RDD.
- ▶ If the Distributed memory (RAM) is not sufficient to store intermediate RDD, then it will store those results on the disk.





## Profile most-frequent k-mer

```
CGGGGCTGGGTCGTACATTCCCCTTTGATA
TTTGAGGGTGCCCAATAACCAAAGCGGACAAA
GGGATGCCGTTTGACGACCTAAATCAACGGCC
AAGGCCAGGAGCGCCTTTGCTGGTTCTACCTG
AATTTTCTAAAAAGATTATAATGTCGGTCTC
CTGCTGTACAACCTGAGATCATGCTGCTTCAAC
TACATGATCTTTTGATGGATGAGGGAATGATG
```

- ▶ Seven (7) 32-nucleotide DNA sequences
- ▶ A "secret" pattern  $P=ATGCAACT$  of length  $l = 8$  implanted.
- ▶ Can you reconstruct  $P$  by analyzing the DNA sequences?



## An example

```
CGGGGCTATGCAACTGGGTCGTACATTCCCCTTTGATA
TTTGAGGGTGCCCAATAAATGCAACTCCAAAGCGGACAAA
GGATGCAACTGATGCCGTTTGACGACCTAAATCAACGGCC
AAGGATGCAACTCCAGGAGCGCCTTTGCTGGTTCTACCTG
AATTTTCTAAAAAGATTATAATGTCGGTCCATGCAACTTC
CTGCTGTACAACCTGAGATCATGCTGCATGCAACTTCAAC
TACATGATCTTTTGATGCAACTGGATGAGGGAATGATG
```

- ▶ The same DNA sequences with the implanted pattern  $ATGCAACT$
- ▶ Can you spot the locations of the implanted pattern?



## An example

```
CGGGGCTATGCAACTGGGTCGTACATTCCCCTTTGATA
TTTGAGGGTGCCCAATAAATGCAACTCCAAAGCGGACAAA
GGATGCAACTGATGCCGTTTGACGACCTAAATCAACGGCC
AAGGATGCAACTCCAGGAGCGCCTTTGCTGGTTCTACCTG
AATTTTCTAAAAAGATTATAATGTCGGTCCATGCAACTTC
CTGCTGTACAACCTGAGATCATGCTGCATGCAACTTCAAC
TACATGATCTTTGATGCAACTGGATGAGGGAATGATG
```

- ▶ Same as before but showing the implant locations.
- ▶ Devise an MapReduce algorithm to automatically identify the implanted pattern
- ▶ Length  $l$  is known.
- ▶ Sequence: <https://goo.gl/xN7WvE>



```
from operator import add

conf = SparkConf().setAppName("Profile").setMaster("local")
sc = SparkContext(conf=conf)

val raw_data = sc.wholeTextFiles("pattern.txt")

def splitLine(line):
    pairs = []
    if len(line) > 1:
        for symbol in range(0, len(line)-8):
            pairs.append((line[symbol:symbol+8], 1))

    return pairs

pairs = raw_data.flatMap(splitLine) \
    .reduceByKey(add) \
    .sortBy(lambda a: -a[1])

print(pairs.take(5))
```



## Profile most-Frequent first appearing k-mer

```
CGGGGCTGGGTCGTACATTCCCCTTTGATA
TTTGAGGGTGCCCAATAACCAAAGCGGACAAA
GGGATGCCGTTTGACGACCTAAATCAACGGCC
AAGGCCAGGAGCGCCTTTGCTGGTTCTACCTG
AATTTTCTAAAAAGATTATAATGTCGGTCCTC
CTGCTGTACAACCTGAGATCATGCTGCTTCAAC
TACATGATCTTTTGTGGATGAGGAATGATGC
```

- ▶ Identify most-frequent first-appearing k-mer
- ▶ In first line, 3-mer CGG appears before GGC, TGG, GTC, ...



```
from operator import add
```

```
conf = SparkConf().setAppName("Profile").setMaster("local")
sc = SparkContext(conf=conf)
```

```
raw_data = sc.textFile("yeast_chr1.txt")
```

```
def splitLine(line):
    pairs = []
    for symbol in range(0, len(line)-6, 3):
        for second in range(symbol+3, len(line)-6, 3):
            pairs.append(((line[symbol:symbol+3],
                           line[second:second+3]), 1))

    return pairs
```

```
return pairs
```

```
pairs = raw_data.flatMap(splitLine) \
                .reduceByKey(add) \
                .sortBy(lambda a: -a[1])
print(pairs.take(10))
```

