GENERAL ASSEMBLY

INTRO TO DATA SCIENCE

LECTURE 22: MAP-REDUCE

I. SETTING UP AMAZON WEB SERVICES (AWS)
II. KEEPING AN EYE ON COSTS
III. GENERATING KEY PAIRS
IV. LAUNCHING AN EC2 INSTANCE

0. ETHICS BY MONICA BULGER, DATA & SOCIETY

I. BIG DATA
II. MAP-REDUCE
III. LAUNCHING A SPARK CLUSTER
IV. PYSPARK EXERCISES ON AWS
V. PYTHON EXERCISES: MULTIPROCESSING, MRJOB

- DESCRIBE WHY HADOOP AND MAP-REDUCE EXIST
- WRITE MAP-REDUCE JOBS IN PYTHON
- LAUNCH SPARK CLUSTER AND WRITE MAP-REDUCE JOBS

I. BIG DATA

Big data is characterized by

Big data is characterized by

- *Volume*

Big data is characterized by

- *Volume*

1 exabyte = 1K petabytes = 1M TB = 1B GB

Volume

- *As of 2012, about 2.5 exabytes of data are created each day, and that number is doubling every 40 months or so*

Source: hbr.org/2012/10/big-data-the-management-revolution

Big data is characterized by

1 exabyte = 1K petabytes = 1M TB = 1B GB

- Volume

Volume

- As of 2012, about 2.5 exabytes of data are created each day, and that number is doubling every 40 months or so
- Walmart collects 2.5+ PB of data every hour from its customers

Source: hbr.org/2012/10/big-data-the-management-revolution

Big data is characterized by

- *Volume*
- *Velocity*

Velocity

- *Twitter's firehose sends 6,000 tweets a second on average*

Big data is characterized by

- *Volume*
- *Velocity*

Velocity

- *Twitter's firehose sends 6,000 tweets a second on average*
- *Twitter saw 140,000 tweets in a single second on August 3, 2013*

Big data is characterized by

- *Volume*
- *Velocity*

Velocity

- *Twitter's firehose sends 6,000 tweets a second on average*
- *Twitter saw 140,000 tweets in a single second on August 3, 2013*
- *Assuming a size of 3 kB per tweet, 6K tweets a second is ~1.5 TB a day*

Big data is characterized by

cellphones ubiquitous
early 2000s

- *Volume*
- *Velocity*
- *Variety*

Variety

- *Cell phone data, texts, GPS signals, etc.*

Source: hbr.org/2012/10/big-data-the-management-revolution

Big data is characterized by

- *Volume*
- *Velocity*
- *Variety*

cellphones ubiquitous
early 2000s

Facebook 2004
Twitter 2006

Variety

- *Cell phone data, texts, GPS signals, etc.*
- *Messages, posts, images posted to social networks*

Source: hbr.org/2012/10/big-data-the-management-revolution

Big data is characterized by

- *Volume*
- *Velocity*
- *Variety*

Variety

- *Cell phone data, texts, GPS signals, etc.*
- *Messages, posts, images posted to social networks*
- *Webpages, for desktop, mobile, tablet, etc.*

cellphones ubiquitous early 2000s

Facebook 2004
Twitter 2006

iPhone 2007
iPad 2010

Source: hbr.org/2012/10/big-data-the-management-revolution

Big data is characterized by

- *Volume*
- *Velocity*
- *Variety*

Variety

- *Cell phone data, texts, GPS signals, etc.*
- *Messages, posts, images posted to social networks*
- *Webpages, for desktop, mobile, tablet, etc.*
- *Movies, music, etc.*

cellphones ubiquitous early 2000s

Facebook 2004
Twitter 2006

iPhone 2007
iPad 2010

In 2010, Netflix's streaming surpassed its mailing business

Source: hbr.org/2012/10/big-data-the-management-revolution

Big data is characterized by

- Volume
- Velocity
- Variety

Variety

- Cell phone data, texts, GPS signals, etc.
- Messages, posts, images posted to social networks
- Webpages, for desktop, mobile, tablet, etc.
- Movies, music, etc.
- Internet of Things, readings from sensors, etc.

cellphones ubiquitous early 2000s

Facebook 2004
Twitter 2006

iPhone 2007
iPad 2010

In 2010, Netflix's streaming surpassed its mailing business

Source: hbr.org/2012/10/big-data-the-management-revolution

Big data is characterized by

- *Volume*
- *Velocity*
- *Variety*

Variety

- *Cell phone data, texts, GPS signals, etc.*
- *Messages, posts, images posted to social networks*
- *Webpages, for desktop, mobile, tablet, etc.*
- *Movies, music, etc.*
- *Internet of Things, readings from sensors, etc.*
- *What's next?*

cellphones ubiquitous early 2000s

Facebook 2004
Twitter 2006

iPhone 2007
iPad 2010

In 2010, Netflix's streaming surpassed its mailing business

Source: hbr.org/2012/10/big-data-the-management-revolution

Big data is characterized by

- *Volume*
- *Velocity*
- *Variety*
- *Veracity*

Veracity

- *1 out of 3 business leaders don't trust the data they use to make decisions*

Source: www.ibmbigdatahub.com/infographic/four-vs-big-data

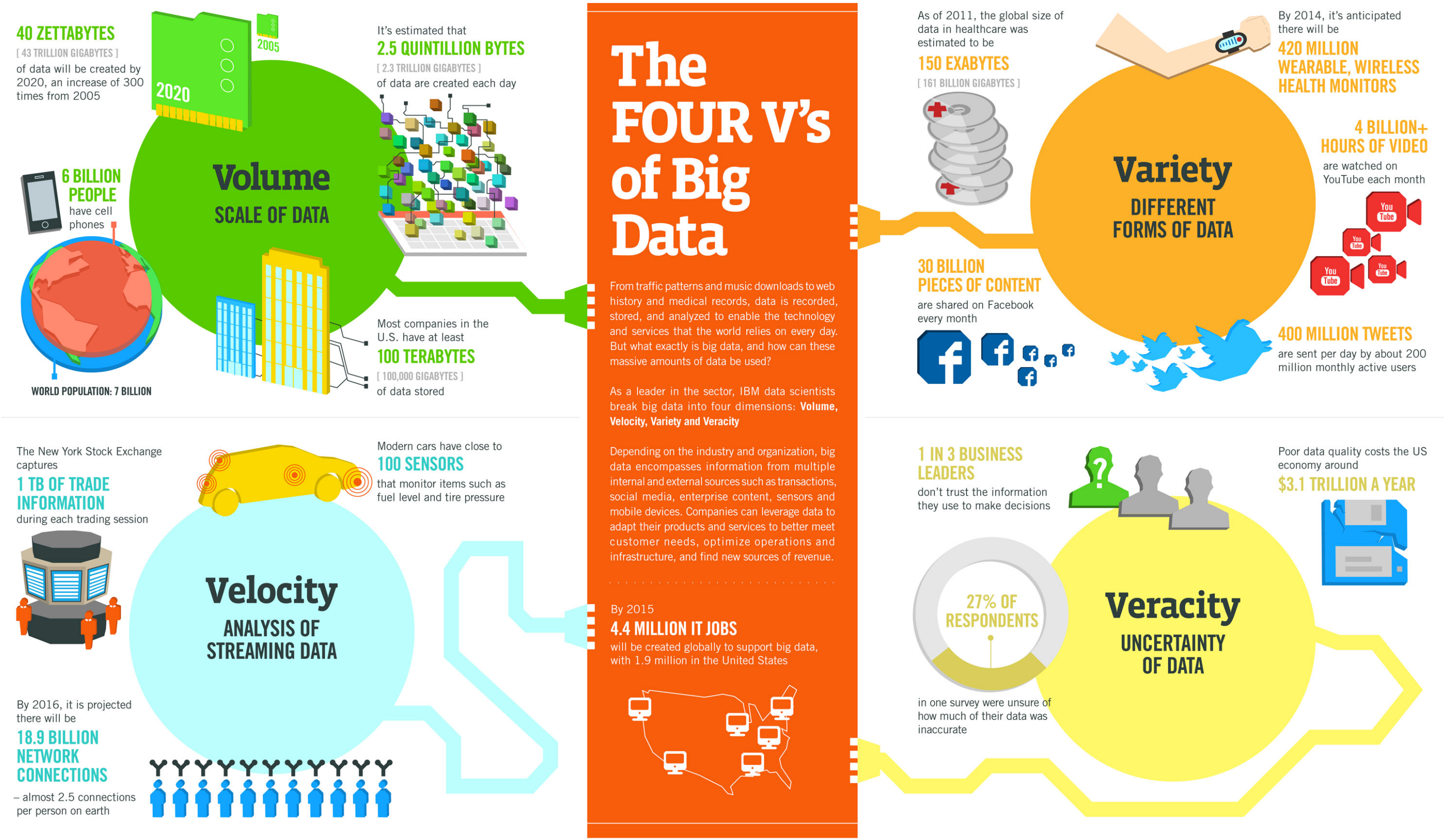
Big data is characterized by

- *Volume*
- *Velocity*
- *Variety*
- *Veracity*

Veracity

- *1 out of 3 business leaders don’t trust the data they use to make decisions*
- *Poor data quality cost the US economy around $3 trillion a year*

Source: www.ibmbigdatahub.com/infographic/four-vs-big-data



Source: www.ibmbigdatahub.com/infographic/four-vs-big-data

II. MAP-REDUCE

How do we process big data?

One approach would be to get a huge supercomputer

- *expensive*
- *difficult to maintain*
- *scalability is bounded*

How do we process big data?

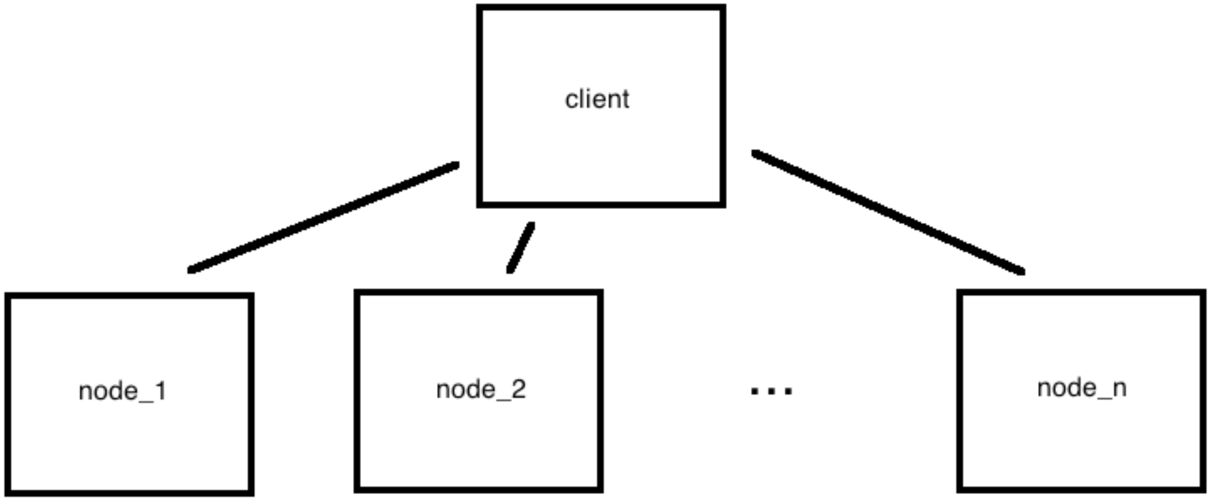
One approach would be to get a huge supercomputer

- *expensive*
- *difficult to maintain*
- *scalability is bounded*

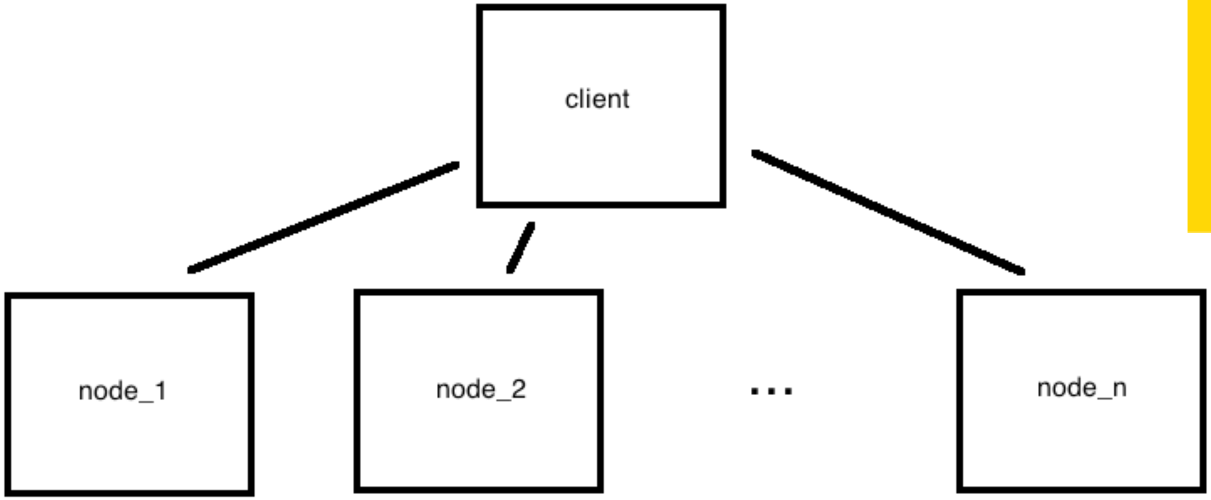
Another approach is to get a bunch of regular (commodity) machines

- *cheaper*
- *easier to maintain*
- *scalability is unbounded (just add another node)*

We can visualize this horizontal cluster architecture as a single client-multiple server relationship



We can visualize this horizontal cluster architecture as a single client-multiple server relationship



NOTE

A horizontally distributed system also has better *fault tolerance* than a single machine.

There are two ways to process data in a distributed architecture:

1) *move data to code (& processing power)*

2) *move code to data*

There are two ways to process data in a distributed architecture:

1) *move data to code (& processing power)*
   - *SETI*

2) *move code to data*
   - *map-reduce → less overhead (network traffic, disk I/O)*

"Computing nodes are the same as storage nodes."

MapReduce *is method for parallel computing, introduced by Google.*

Frequently when people say "map-reduce" they're referring to Hadoop, but it is really a **paradigm** *found in many other places as well*

numpy, NoSQL, Cloudera, MapR, GFS, ...

The *MapReduce framework consists of two phases*

- *a* **mapper***, which performs some local operation*
- *a* **reducer***, which aggregates the results that are sent back*

The *MapReduce framework consists of two phases*

- *a* **mapper***, which performs some local operation*
- *a* **reducer***, which aggregates the results that are sent back*

Example

(1, 2, 3, ..., 300) ⟶ *averaging* ⟶ 150

The *MapReduce framework consists of two phases*

- *a* **mapper**, *which performs some local operation*
- *a* **reducer**, *which aggregates the results that are sent back*

Example

parallelize

(1, 2, 3, ..., 300) → (1, 2, 3, ..., 100)

(1, 2, 3, ..., 300) → (101, 102, ..., 200) → 150

(1, 2, 3, ..., 300) → (201, 202, ..., 300)

The *MapReduce framework consists of two phases*

- *a* **mapper**, *which performs some local operation*
- *a* **reducer**, *which aggregates the results that are sent back*

Example

parallelize · *map* · *sum, count*

(1, 2, 3, ..., 300) → (1, 2, 3, ..., 100) → (50, 100)

(1, 2, 3, ..., 300) → (101, 102, ..., 200) → (150, 100)

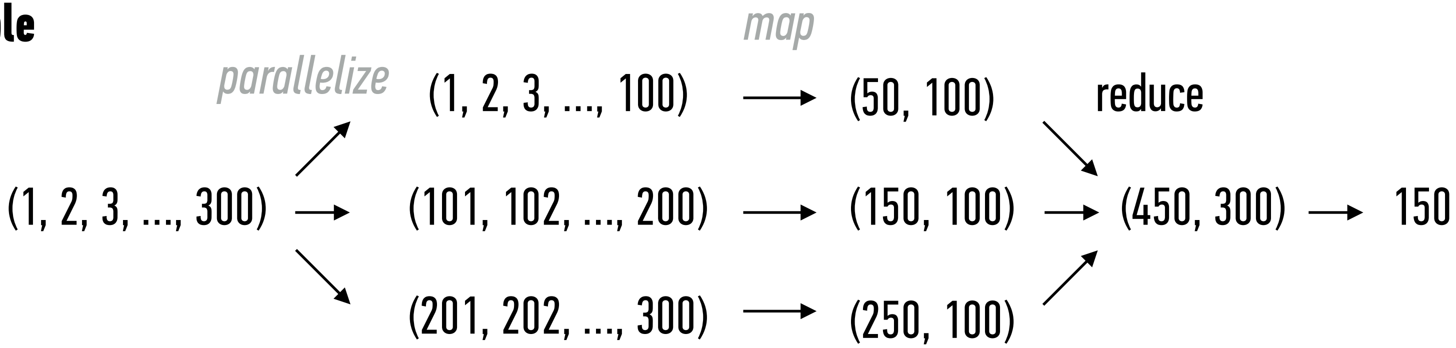
(1, 2, 3, ..., 300) → (201, 202, ..., 300) → (250, 100)

(50, 100), (150, 100), (250, 100) → 150

The *MapReduce framework consists of two phases*

- *a* **mapper**, *which performs some local operation*
- *a* **reducer**, *which aggregates the results that are sent back*

Example



Example: Word Count

```
where
where in
where in the
where in the world
where in the world is
where in the world is carmen
where in the world is carmen sandiego
```

Example: Word Count

```
(where, 1)
(where, 1), (in, 1)
(where, 1), (in, 1), (the, 1)
…
```

map

keyBy

```
where
where in
where in the
where in the world
where in the world is
where in the world is carmen
where in the world is carmen sandiego
```

Example: Word Count

```
where
where in
where in the
where in the world
where in the world is
where in the world is carmen
where in the world is carmen sandiego
```

map

keyBy

```
(where, 1)
(where, 1), (in, 1)
(where, 1), (in, 1), (the, 1)
…
```

partitioner

shuffle & sort

```
(where, [1, 1, 1, 1, 1, 1, 1])
(in, [1, 1, 1, 1, 1, 1])
(the, [1, 1, 1, 1, 1])
…
```

Example: Word Count

```
where
where in
where in the
where in the world
where in the world is
where in the world is carmen
where in the world is carmen sandiego
…
```

map

keyBy

```
(where, 1)
(where, 1), (in, 1)
(where, 1), (in, 1), (the, 1)
…
```

partitioner

shuffle & sort

```
(where, [1, 1, 1, 1, 1, 1, 1])
(in, [1, 1, 1, 1, 1, 1])
(the, [1, 1, 1, 1, 1])
…
```

reduceByKey

reduce

```
(where, 7)
(in, 6)
(the, 5)
…
```

Example: Word Count

```
where
where in
where in the
where in the world
where in the world is
where in the world is carmen
where in the world is carmen sandiego
```

map

keyBy

```
(where, 1)
(where, 1), (in, 1)
(where, 1), (in, 1), (the, 1)
…
```

partitioner

shuffle & sort

```
(where, [1, 1, 1, 1, 1, 1, 1])
(in, [1, 1, 1, 1, 1, 1])
(the, [1, 1, 1, 1, 1])
…
```

reduceByKey

reduce

```
(where, 7)
(in, 6)
(the, 5)
…
```

sortByKey

```
(carmen, 2)
(is, 3)
(in, 6)
(the, 5)
(sandiego, 1)
(where, 7)
(world, 4)
```

III. IMPLEMENTATION

The map-reduce framework handles a lot of messy details for you:

- parallelization & distribution (eg, input splitting)

- partitioning (shuffle/sort/redirect)

- fault-tolerance (fact: tasks/nodes will fail!)

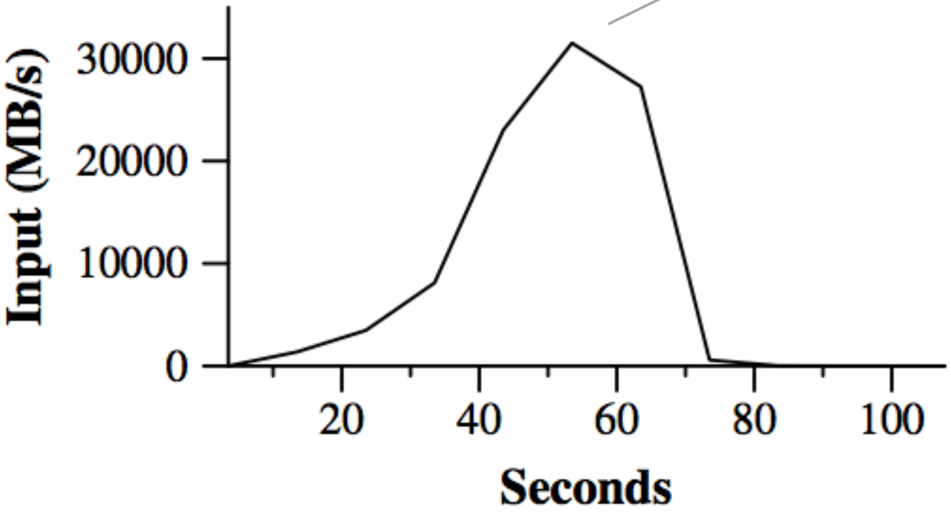
- I/O scheduling

- status and monitoring

This (along with the functional semantics) allows you to focus on solving the problem instead of accounting & housekeeping details.

Data transfer rate over time

Peaks at 30+ GB/s when
1764 workers have been assigned



grep program

- 10^{10} 100-byte records
- searches for 3-char pattern (occurs ~90K times)
- input split into ~64MB pieces (M = 15,000)

Source: Jeffrey Dean and Sanjay Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters" (2004)

The Google File System (GFS) was developed alongside map-reduce to serve as the native file system for this type of processing.

The Hadoop platform is bundled with an open-source implementation of this file system called HDFS.

If you use Amazon EMR, you can use their file system (Amazon S3) as well.

Hadoop *is a popular open-source Java-based implementation of the map-reduce framework (including file storage for input/output).*

You can download Hadoop and configure a set of machines to operate as a map-reduce cluster, or you can run it as a service via Amazon's Elastic Map-Reduce.

Hadoop is written in Java, but the Hadoop Streaming utility allows client code to be supplied as executables (eg, written in any language).

Doug Cutting, Hadoop's creator, named the framework after his child's stuffed toy elephant

IV. EXERCISES

- *Launch Spark cluster in AWS*
- *Write jobs in pyspark using AWS cluster*
- *multiprocessing example in ipython notebook*
- *mrjob implementation in python*

DISCUSSION