



Big Data and Finance

This Set Of Slides Is Based On The Emc Course "Data Science And Big Data Analytics" And The Book "Data Analytics With Hadoop"

Why MapReduce?

“If one ox could not do the job, they did not try to grow a bigger ox, but used two oxen.

When we need greater computer power, the answer is not to get a bigger computer, but . . . to build systems of computers and operate them in parallel.”

Grace Hopper



Grace Hopper



The MapReduce paradigm helps you **add** more oxen

By definition, big data is too large to handle by conventional means. Sooner or later, you just can't scale up anymore

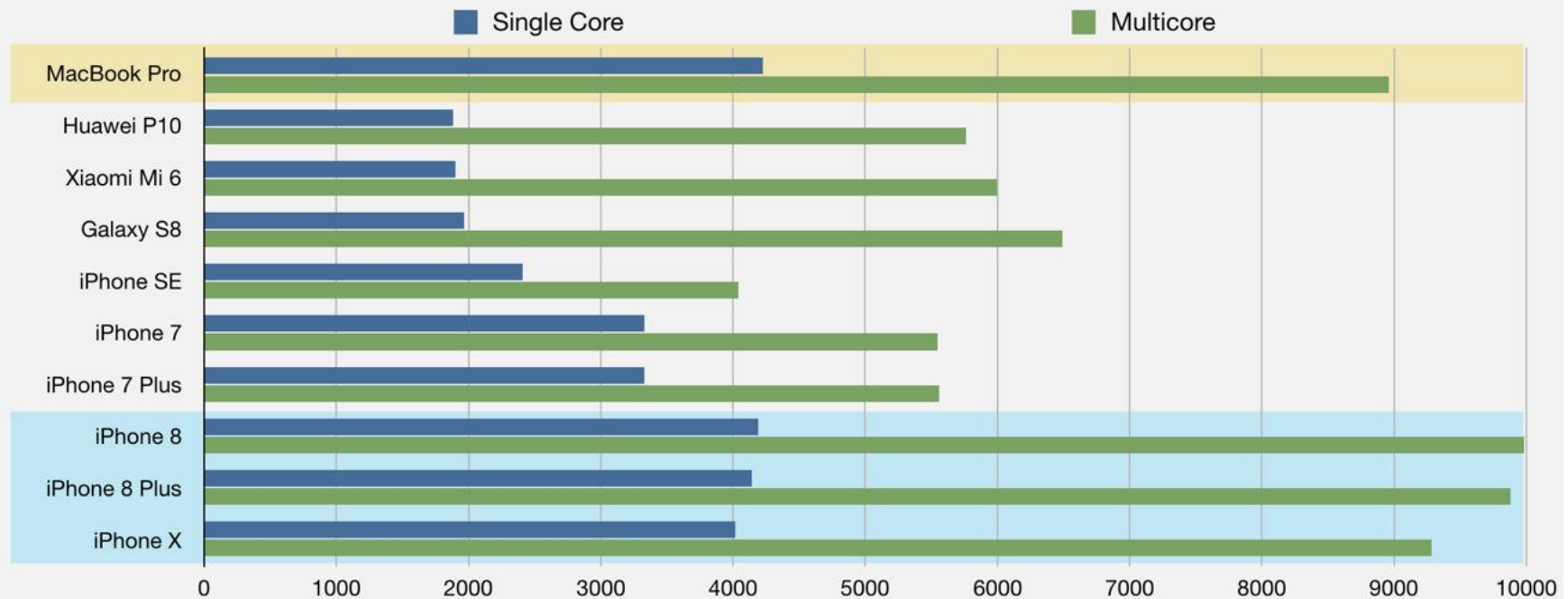


She was an American [computer scientist](#) and [United States Navy rear admiral](#).

She popularized the idea of machine-independent programming languages, which led to the development of [COBOL](#), an early [high-level programming language](#) still in use today.

Multicore is Faster than Single Core

Geekbench 4	MacBook Pro 13" 2017	Huawei P10	Xiaomi Mi 6	Samsung Galaxy S8	iPhone SE	iPhone 7	iPhone 7 Plus	iPhone 8	iPhone 8 Plus	iPhone X
Single Core	4229	1886	1900	1965	2410	3328	3332	4189	4142	4017
Multicore	8959	5763	5996	6495	4043	5545	5558	9983	9883	9286
Processor	Intel Core i5 7267U	Hisilicon Kirin 960	Qualcomm Snapd. 835	Samsung Exynos 8895	A9	A10 Fusion	A10 Fusion	A11 Bionic	A11 Bionic	A11 Bionic
CPU clock	3.5 GHz	2.25 GHz	1.9 GHz	1.6 GHz	1.8 GHz	2.34 GHz	2.34 GHz	?	?	?
RAM	8 GB	4 GB	6 GB	4 GB	1 GB	2 GB	3 GB	2 GB	3 GB	3 GB
Display	2560x1600	1080x1920	1080x1920	1440x2960	640x1136	750x1334	1080x1920	750x1334	1080x1920	1125x2436



Scores are recorded by Primate Lab's Geekbench 4 relative to an Intel Core i7-6600U, which represents 4000.

Higher Display Resolution involved parallel processing (multiple GPUs)



CityU Burgundy – 10 times faster than existing High Power Computers



A new state-of-the-art High-Performance Computing facility – CityU Burgundy – has been launched at CityU.

中央處理器

CPU (TFLOPS): 598

圖像處理器

GPU (FP32 TFLOPS): 1,111

圖像處理器

GPU (FP64 TFLOPS): 513

記憶體最大容量

MAXIMUM MEMORY SIZE PER FAT NODE: 4.5TB

圖像處理器型號

GPU CARD MODEL: NVIDIA V100S, V100

儲存容量

STORAGE SIZE: 750TB

What is MapReduce?

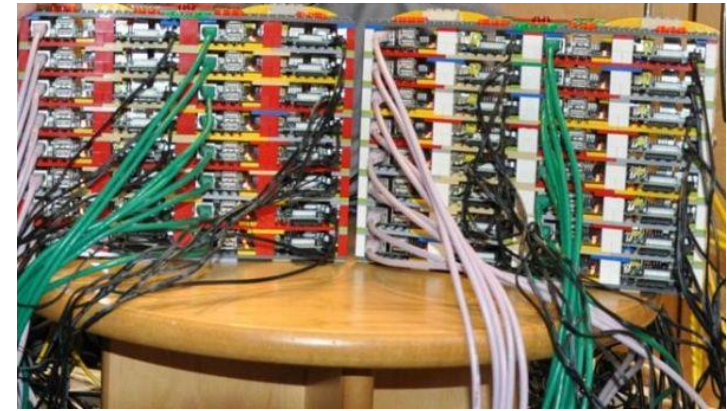
- A parallel programming model suitable for big data processing
 - Split data into distributable chunks (“shards”)
 - Define the steps to process those chunks
 - Run that process in parallel on the chunks
- Scalable by adding more machines to process chunks
 - Leverage commodity hardware to tackle big jobs
- The foundation for Hadoop*
 - **MapReduce** is a **parallel programming model**
 - **Hadoop** is a **concrete platform that implements MapReduce**

[Apache Hadoop](#) is a collection of open-source software utilities that facilitate using a network of many computers to solve problems involving massive amounts of data and computation.

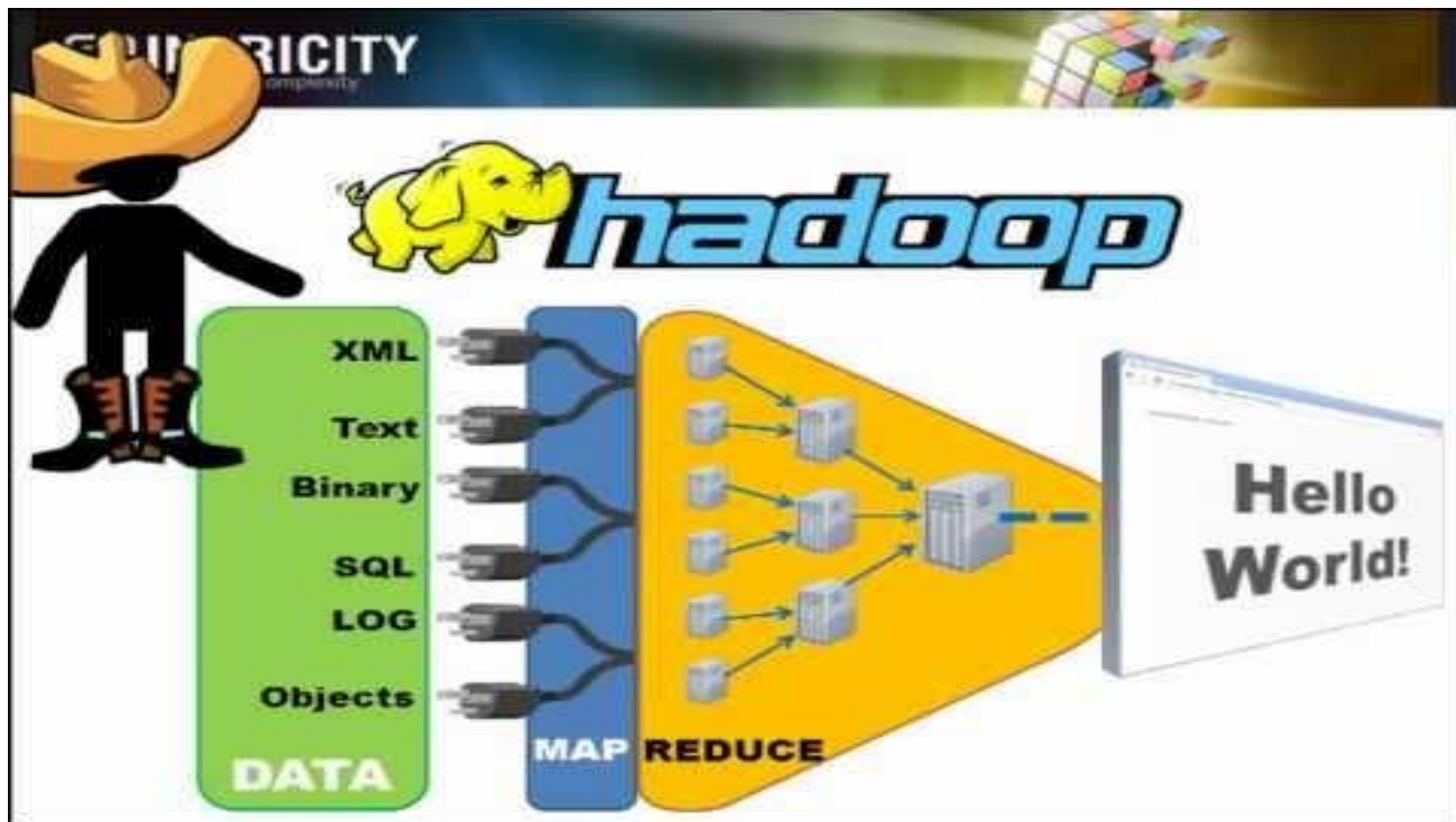
750 Raspberry Pis Turned Into Supercomputer for Los Alamos National Laboratory



It's often a challenge for programmers and scientists to get time on high-performance supercomputers. These machines are expensive to build and maintain, but there's no substitute for the massively parallel computing environment of a supercomputer. A new project at the Los Alamos National Laboratory's High Performance Computing Division seeks to make supercomputers more accessible with a little help from some Raspberry Pi clusters.



What is Hadoop?



When to Use MapReduce?

Problems that are “embarrassingly parallel”

Examples

- Word count
- Reverse indexing
- tf-idf*
- Distributed *grep*** and distributed object recognition
- Distributed "associative" aggregation (sum; mean; min or max; count)
- Hadoop calls them "combiners"

*It stand for “term frequency-inverse document frequency” which is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus.

** *grep* is a command-line utility for searching plain-text data sets for lines that match a regular expression.

The Map part of MapReduce

- Transform
 - (Map) input values to output values: $\langle k1, v1 \rangle \rightarrow \langle k2, v2 \rangle$
- Input – Key/Value Pairs
 - For instance, Key = line number, Value = text string
- Map Function
 - Steps to transform input pairs to outputpairs
 - For example, count the different words in the input
- Output – Key/Value Pairs
 - For example, Key = $\langle \text{word} \rangle$, Value = $\langle \text{count} \rangle$
- Map output is the input to Reduce

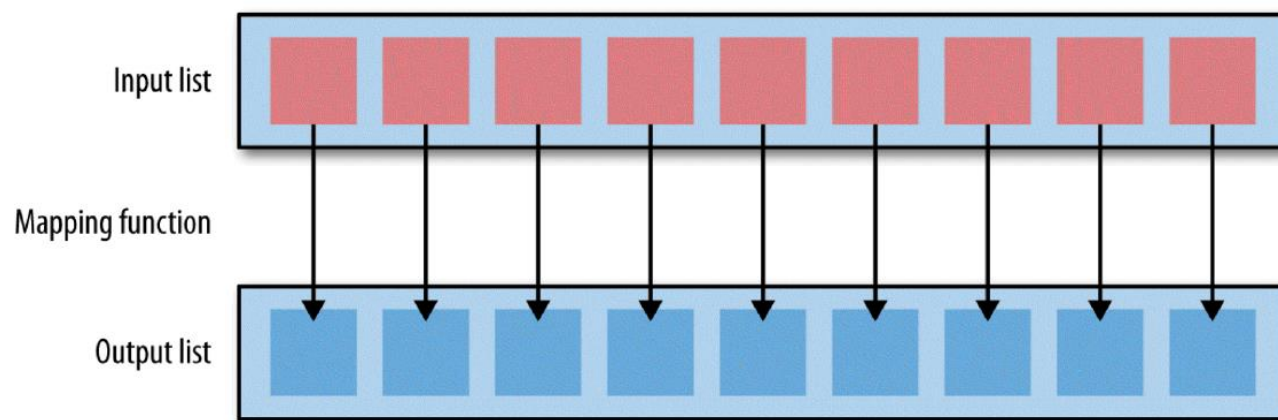


Figure 2-3 A map function as input a list of key/value pairs and operates upon each individual element in the list, outputting zero or more key/value pairs

The Reduce Part of MapReduce

Merge (Reduce) Values from the Map phase

- Reduce is optional. Sometimes all the work is done in the Mapper

Input

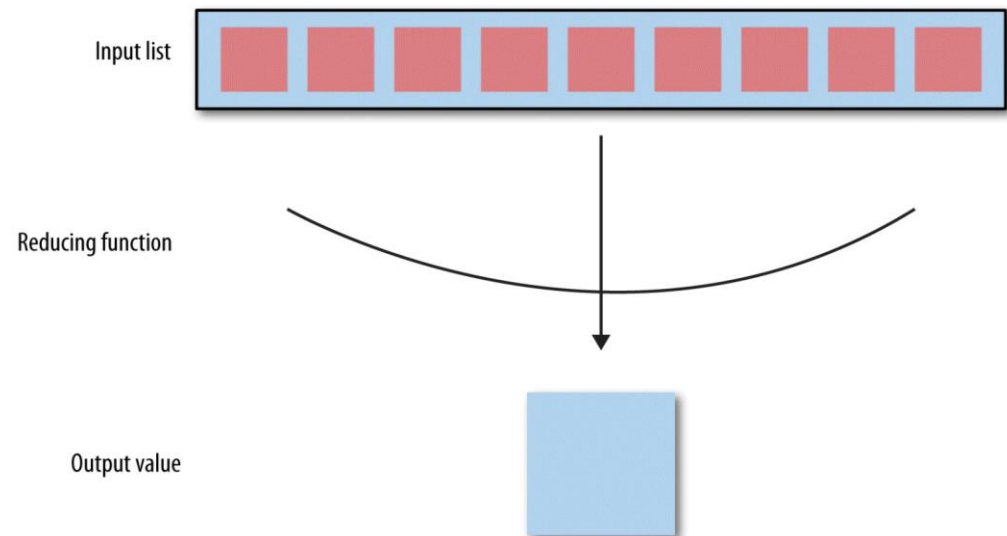
- Values for a given Key from all the Mappers

Reduce Function

- Steps to combine (Sum?, Count?, Print?,...) the values

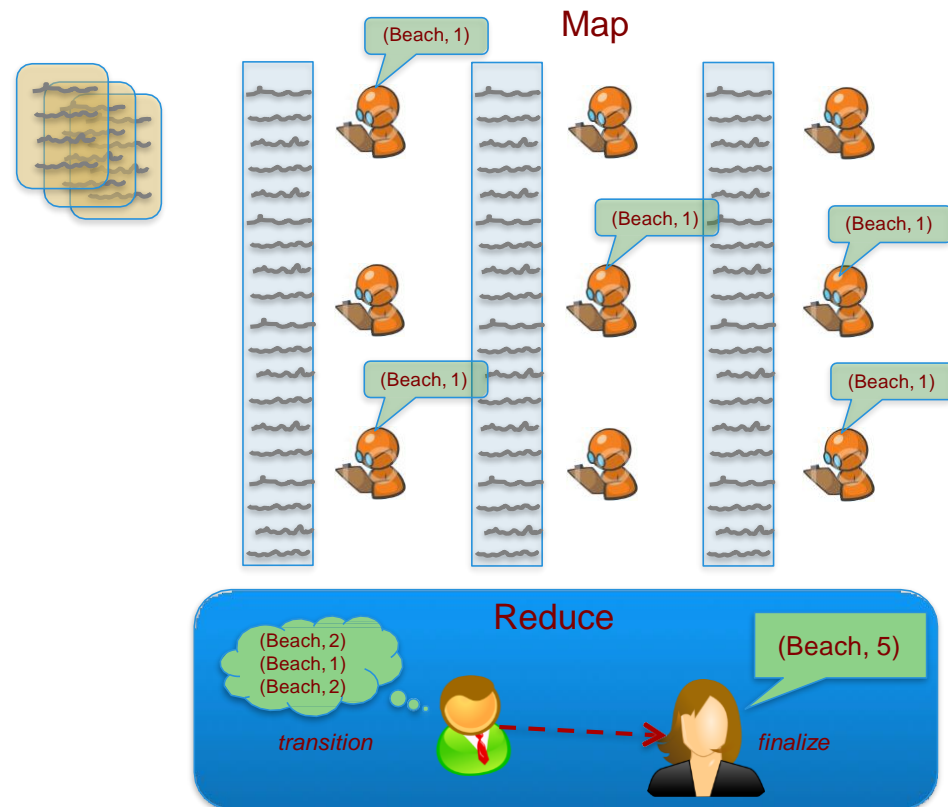
Output

- Print values?, load into a DB? send to the next MapReduce job?



Motivating Example: Word Count

- This is the “Hello World” of MapReduce
- Distribute the text of millions of documents over hundreds of machines
- **MAPPERS** can be word-specific. They run through the stacks and shout “One!” every time they see the word “beach”
- **REDUCERS** listen to all the Mappers and total the counts for each word.



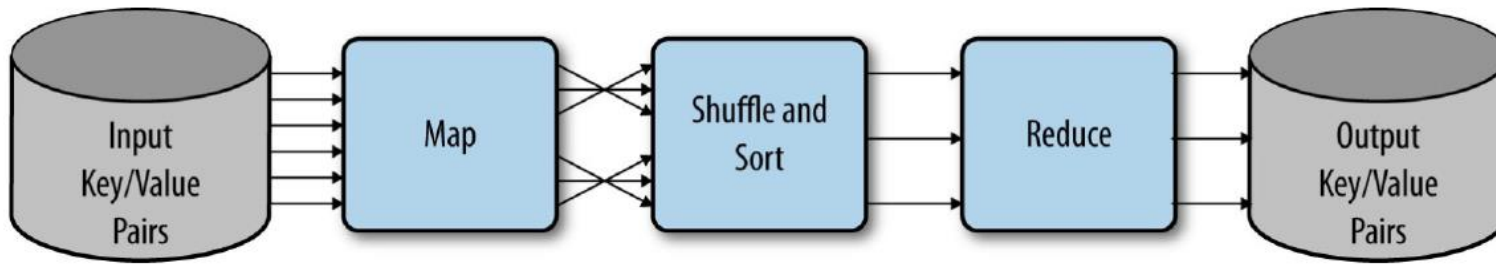


Figure 2-5. Broadly, MapReduce is implemented as a staged framework where a map phase is coordinated to a reduce phase via an intermediate shuffle and sort

The phases shown in [Figure 2-5](#) are as follows:

Phase 1

Local data is loaded into a mapping process as key/value pairs from Hadoop Distributed File System (HDFS)*.

Phase 2

Mappers output zero or more key/value pairs, mapping computed values to a particular key.

Phase 3

These pairs are then sorted and shuffled based on the key and are then passed to a reducer such that all values for a key are available to it.

Phase 4

Reducers then must output zero or more final key/value pairs, which are the output (reducing the results of the map).

*The Hadoop Distributed File System (HDFS) is the primary data storage system used by [Hadoop](#) applications. It employs a NameNode and DataNode architecture to implement a [distributed file system](#) that provides high-performance access to data across highly scalable Hadoop clusters.

Input to WordCount mappers

```
(27183, "The fast cat wears no hat.")  
(31416, "The cat in the hat ran fast.")
```

Mapper 1 output

```
("The", 1), ("fast", 1), ("cat", 1), ("wears", 1),  
("no", 1), ("hat", 1), (".", 1)
```

Mapper 2 output

```
("The", 1), ("cat", 1), ("in", 1), ("the", 1),  
("hat", 1), ("ran", 1), ("fast", 1), (".", 1)
```

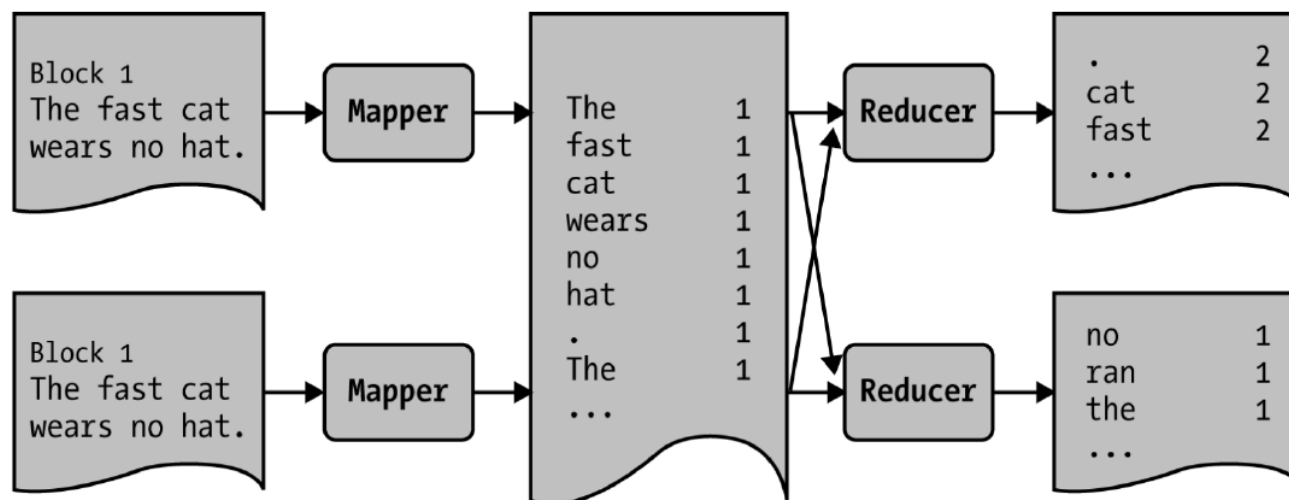


Figure 2-7. Data flow of the word count job being executed on a cluster with two mappers and two reducers



```
# Input to WordCount reducers
# This data was computed by shuffle and sort
```

```
(., [1, 1])
("cat", [1, 1])
("fast", [1, 1])
("hat", [1, 1])
("in", [1])
("no", [1])
("ran", [1])
("the", [1])
("wears", [1])
("The", [1, 1])
```

```
# Output by all WordCount reducers
```

```
(., 2)
("cat", 2)
("fast", 2)
("hat", 2)
("in", 1)
("no", 1)
("ran", 1)
("the", 1)
("wears", 1)
("The", 2)
```

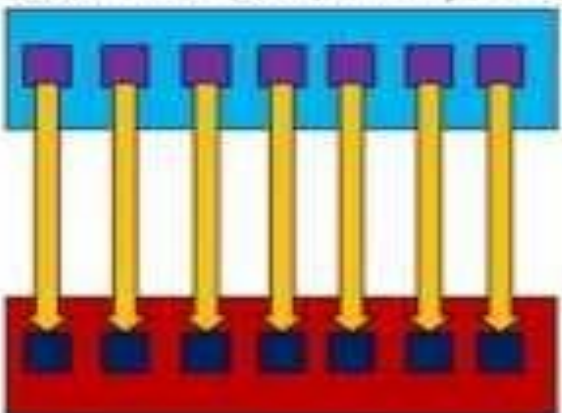
How MapReduce works?



Mapper

Maps input key/value pairs to a set of intermediate key/value pairs.
Maps are the individual tasks that transform input records into **intermediate** records.

Input Data



www.BigDataTrunk.com

9 000077

Simple Example of a Mapper: mapper.py

```
"""mapper.py"""

import sys

# input comes from STDIN (standard input)
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # split the line into words
    words = line.split()
    # increase counters
    for word in words:
        # write the results to STDOUT (standard output);
        # what we output here will be the input for the
        # Reduce step, i.e. the input for reducer.py
        #
        # tab-delimited; the trivial word count is 1
        print '%s\t%s' % (word, 1)
```


Simple Example of a Reducer: reducer.py

```
#!/usr/bin/env python
"""reducer.py"""

from operator import itemgetter
import sys

current_word = None
current_count = 0
word = None

# input comes from STDIN
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()

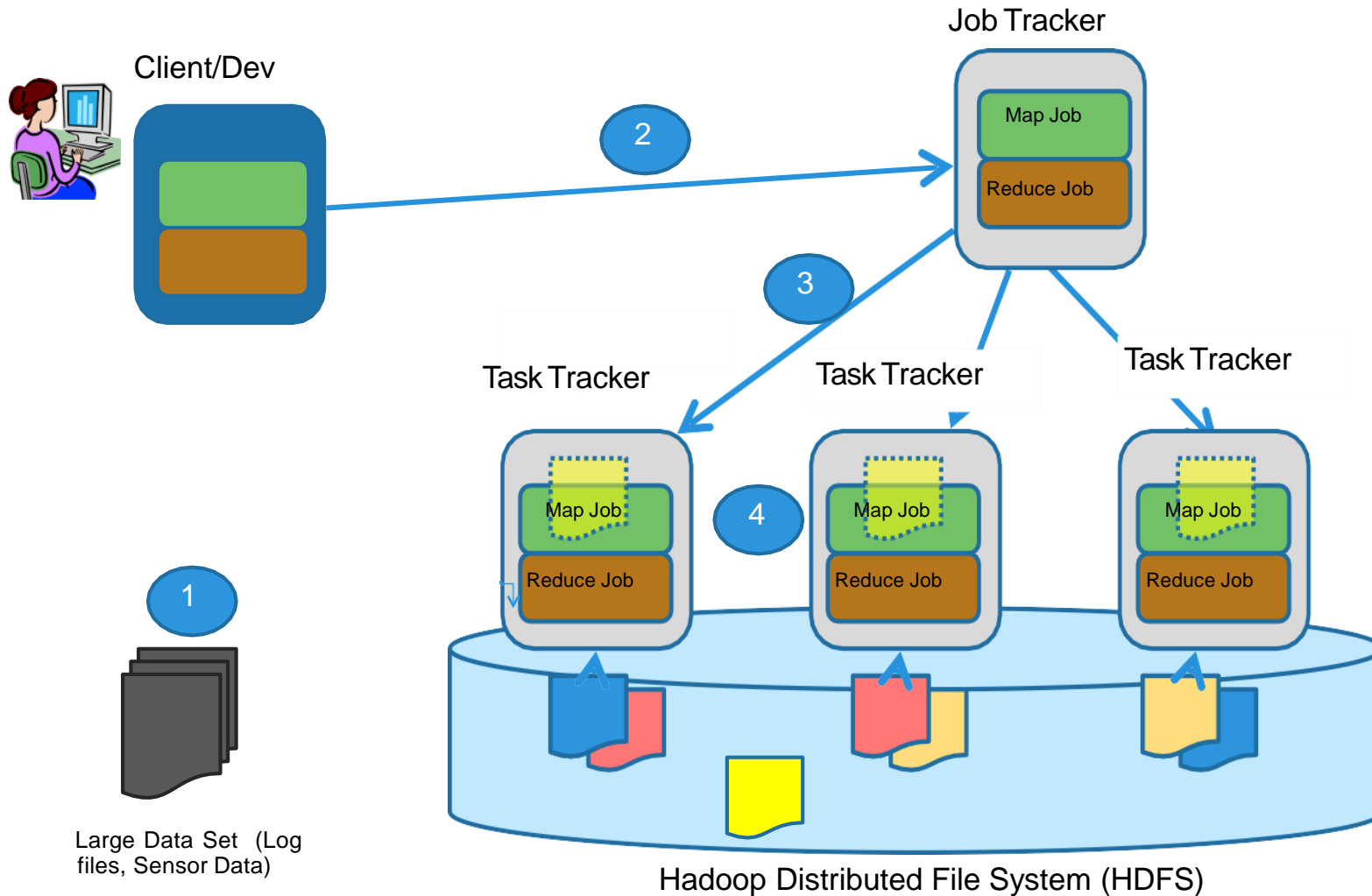
    # parse the input we got from mapper.py
    word, count = line.split('\t', 1)

    # convert count (currently a string) to int
    try:
        count = int(count)
    except ValueError:
        # count was not a number, so silently
        # ignore/discard this line
        continue

    # this IF-switch only works because Hadoop sorts map output
    # by key (here: word) before it is passed to the reducer
    if current_word == word:
        current_count += count
    else:
        if current_word:
            # write result to STDOUT
            print '%s\t%s' % (current_word, current_count)
            current_count = count
            current_word = word

# do not forget to output the Last word if needed!
if current_word == word:
    print '%s\t%s' % (current_word, current_count)
```

Putting it all Together: MapReduce and HDFS



Example: Social Triangles (e-discovery)

Suppose you have 517,424 emails from an energy company - case that you need to investigate.

The social network may be implied by all To:From pairings in the emails, with reflexivity accounted for.

Date: Thu, 4 May 2000 09:55:00 -0700 (PDT)
From: walt.zimmerman@enron.com
To: michael.burke@enron.com, dana.gibbs@enron.com, lori.maddox@enron.com,
susan.ralph@enron.com Subject: Update on Steve Todoroff Prosecution--
CONFIDENTIAL/SUBJECT TO ATTORNEY-CLIENT PRIVILEGE Cc:
steve.duffy@enron.com, stanley.horton@enron.com, jdegeeter@velaw.com

Almost one month ago, Special Agent Carl Wake of the FBI called me about the Steve Todoroff investigation. He indicated that the FBI had recently learned of the article about EOTT's NGL theft that appeared in the business section of the Houston Chronicle. Mr. Wake said it might be a matter the FBI would like to investigate. I told Mr. Wake that EOTT was currently working with the Harris County District Attorney on the prosecution of this matter, and I thanked him for the FBI's interest. He told me that the FBI might want to work with the Harris County District Attorney in investigating this matter, and he stated that there may be investigative information that the FBI can obtain more quickly than the Harris County District Attorney. Mr. Wake requested a copy of the materials we had provided to the Harris County District Attorney.

In order to avoid damage to the good rapport we have established with Assistant District Attorney Bill Moore, I asked John DeGeeter to call Bill Moore and advise him of the contact that had been made by the FBI. Bill Moore agreed to call Carl Wake and work with Mr. Wake on his request for the materials provided by EOTT.

Carl Wake called me again yesterday. He has been working with Bill Moore. Mr. Wake stated it was too early to speculate as to what charges would be brought. He did say that our materials clearly indicated federal wire fraud and possibly mail fraud. He said that where there is wire fraud, there is usually money laundering.

The purpose of Mr. Wake's call yesterday was to inquire about the status of some interview summaries that John DeGeeter and I have prepared and collected at the request of Bill Moore. Mr. Wake requested that EOTT send a copy of the summaries to him when we sent the summaries to Bill Moore. Those summaries were sent out today.

I gathered from my calls with Carl Wake that the FBI is very interested in taking an active part in this investigation. In order to build on the relationship we have established with Bill Moore, we will continue to direct our inquiries about the investigation to Mr. Moore until he tells us to do otherwise.

Social Triangle: First Directed Edge

Mapper1

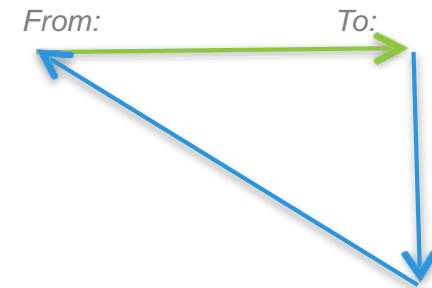
Maps two regular expression searches:

To: Michael, Dan, Lori, Susan

From: Walt

Emits the outbound directed edge of the social graph:

<Key, Value> = <Walt, [Michael, Dan, Lori, Susan]>



Reducer1

Gets the output from the mapper with different values

<Key, Value> = <Walt, [Michael, Dan, Lori, Susan]>

<Key, Value> = <Walt, [Lori, Susan, Jeff, Ken]>

Unions the values for the second directed edge:

<Key, Value> = <Walt, [Dan, Jeff, Ken, Lori, Michael, Susan]>

Data is reduced by about a third.

Social Triangle: Second Directed Edge

Mapper2

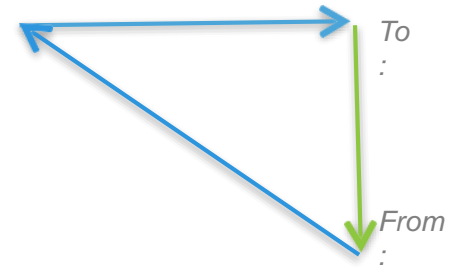
Reverses the previous Map:

To: Dan, Jeff, Ken, Lori, Michael, Susan

From: Walt

Emits the inbound directed edge of the social graph:

<Key, Value> = <Dan, Walt>; <Jeff, Walt>; .. <Susan, Walt>; etc



Reducer2

Gets the output from the mapper with different values

<Key, Value> = <Susan, Walt>

<Key, Value> = <Susan, Jeff>

Unions the values for the third directed edge:

<Key, Value> = <Susan, [Jeff, Ken, Walt]>

Data again reduced by about a third.

Social Triangle: Third Directed Edge

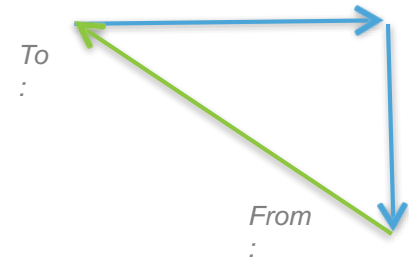
Mapper3

Join [inbound] and [outbound] lists by Key

Walt, [Jeff, Ken, Lori, Susan], [Jeff, Lori, Stanley, Walt]

Emits <Person, Person> pair with level of association:

<Key, Value> = <Walt:Jeff, reciprocal>; <Walt:Stanley, directed>, etc



Reducer3

Reducer unions the output of the mappers and presents rules:

<Key, Value> = <Walt:Jeff, reciprocal>

<Key, Value> <Walt:Stanley, directed>

The third reducer can shape the data any way that serves the business objective.