CSx55: Distributed Systems [MapReduce]

To Orchestrate a Job in a Cluster

A job comprises many a task
What could be so hard, you ask?
A job's done, when every task wraps up
Circumventing every hiccup
Machines may slowdown or go bust
For no reason nor rhyme
Try to complete, you must
All tasks, at roughly the same time

Shrideep Pallickara
Computer Science
Colorado State University

Frequently asked questions from the previous class survey

- Can Chord, Tapestry, and Pastry "communicate" with each other?
- Chord does not have a leaf set?
- Why does Chord seem to have duplicate entries in its FT?

Matrix Cup Standings [10/1]

Tyler Malone	2.20
Brenner Lattin	2.48
Tommy McRoskey	21.62
Zacharie Guida	22.46
Connor Chapman	25.47
Cameron Mordini	33.44
Parker Jones	44.10

https://www.cs.colostate.edu/~csx55/matrix

Topics covered in today's lecture

- MapReduce

MapReduce

Orchestrating computations over voluminous data: A Google Case Study

- Late 90s work on developing effective search
  - Distributed PageRank algorithm and data structures to process crawled data and rank results
- Google File System: Organize crawled data so that they are amenable to programmatic interactions
- MapReduce: A distributed computational framework for processing large amounts of data
- TensorFlow: A distributed computational framework for training AI models at scale
- Google Translate and Speech recognition
- Medical applications

MapReduce: Topics that we will cover

- Why?
- What it *is* and what it *is not*?
- The core framework and the original Google paper
- Development of programs using Hadoop
  - The dominant MapReduce implementation

MapReduce

- It's a **framework** for processing data residing on a large number of computers
- Very powerful framework
  - Excellent for some problems
  - Challenging or not applicable in other classes of problems

What is MapReduce?

- More a **framework** than a tool
- You are required to **fit** (some folks shoehorn it) your solution into the MapReduce framework
- MapReduce is not a feature, but rather a **constraint**

What does this constraint mean?

- It makes problem solving easier *and* harder
- Clear boundaries for what you can and cannot do
  - You actually need to consider **fewer options** than what you are used to
- But solving problems with constraints requires planning and a ***change*** in your thinking

But what does this get us?

- Tradeoff of being confined to the MapReduce framework?
  - Ability to process data on a large number of computers
  - But, more importantly, **without having to worry about** concurrency, scale, fault tolerance, and robustness

A challenge in writing MapReduce programs

- **Design**!
  - Good programmers can produce bad software due to poor design
  - Good programmers can produce bad MapReduce algorithms
- Only in this case your **mistakes will be amplified**
  - Your job may be distributed on 100s or 1000s of machines and operating on a Petabyte of data

MAPREDUCE

MATERIALS BASED ON

JEFFREY DEAN and SANJAY GHEMAWAT: *MapReduce: Simplified Data Processing on Large Clusters*. OSDI 2004: 137-150

MapReduce

- Programming model
- Associated implementation for
  - Processing & Generating large data sets

Programming model

- Computation takes a set of **input** *key/value* pairs
- Produces a set of **output** *key/value* pairs
- Express the computation as two functions:
  - Map
  - Reduce

Map

- Takes an input pair
- Produces a set of intermediate key/value pairs

Mappers

- If map operations are **independent** of each other, they can be performed in parallel
  - **Shared nothing**
- This is usually the case

MapReduce library

- **Groups** all intermediate values with the same intermediate key
- **Passes** them to the Reduce function

Reduce function

- Accepts intermediate *key* I and
  - Set of *values* for that *key*
- **Merge** these *values* together to get
  - Smaller set of *value*

Counting number of occurrences of each word in a large collection of documents

```
map (String key, String value)
    //key: document name
    //value: document contents

  for each word w in value
     EmitIntermediate(w, "1")
```

The Road Not Taken
Robert Frost

Two roads diverged in a yellow wood,
And sorry I could not travel both
And be one traveler, long I stood
And looked down one as far as I could
To where it bent in the undergrowth;

Then took the other, as just as fair,
And having perhaps the better claim,
Because it was grassy and wanted wear;
Though as for that the passing there
Had worn them really about the same,

And both that morning equally lay
In leaves no step had trodden black.
Oh, I kept the first for another day!
Yet knowing how way leads on to way,
I doubted if I should ever come back.

I shall be telling this with a sigh
Somewhere ages and ages hence:
Two roads diverged in a wood, and I—
I took the one less traveled by,
And that has made all the difference.

```
two   1
roads     1
diverged  1
in    1
a     1
yellow    1
wood      1
and   1
sorry     1
i     1
could     1
not   1
travel    1
both  1
and   1
be    1
one   1
…
```

Counting number of occurrences of each word in a large collection of documents

```
reduce (String key, Iterator values)
    //key: a word
    //value: a list of counts

    int result = 0;
      for each v in values
        result += ParseInt(v);
    Emit(AsString(result));
```

Sums together all counts emitted for a particular word

The Road Not Taken
Robert Frost

a	3
about	1
ages	2
all	1
and	9
another	1
as	5
back	1
be	2
because	1
bent	1
better	1
black	1
both	2
by	1
claim	1
come	1
could	2
day	1
difference	1
diverged	2
…	

MapReduce specification object contains

- Names of
  - Input
  - Output
- Tuning parameters

Map and reduce functions have associated types drawn from different domains

map(k1, v1) → list(k2, v2)

reduce(k2, list(v2)) → list(v2)

What's passed to-and-from user-defined functions?

- Strings
- User code converts between
  - String
  - Appropriate types

Examples

History is Philosophy teaching by example.

Thucydides

Programs expressed as MapReduce computations: Distributed Grep

- Map
  - Emit line if it matches specified pattern
- Reduce
  - Just copy intermediate data to the output
    - The reducer here is an identity function

Counts of URL access frequency

- Map
  - Process logs of web page requests
  - Output <URL, 1>
- Reduce
  - Add together all values for a particular URL
  - Output <URL, total count>

Reverse Web-link Graph

- Map
  - Outputs <target, source> pair for each target URL found in page source
- Reduce
  - Concatenate list of all sources for a target URL
  - Output <target, *list*(source)>

Term-Vector per Host

- Summarizes important terms that occur in a set of documents <word, frequency>
- For each input document, the Map
  - Emits <hostname, term vector>
- Reduce function
  - Has all per-document vectors for a given host
  - Add term vectors; discard away infrequent terms
    - <hostname, term vector>

Inverted Index

- Map
  - Parse each document
  - Emit <word, document ID>
- Reduce
  - Accept all pairs for a given word
  - Sort document IDs
  - Emit <word, list(document ID)> pair

Implementation

Implementation

- Machines are **commodity** machines
- **GFS** is used to manage data stored on the disks

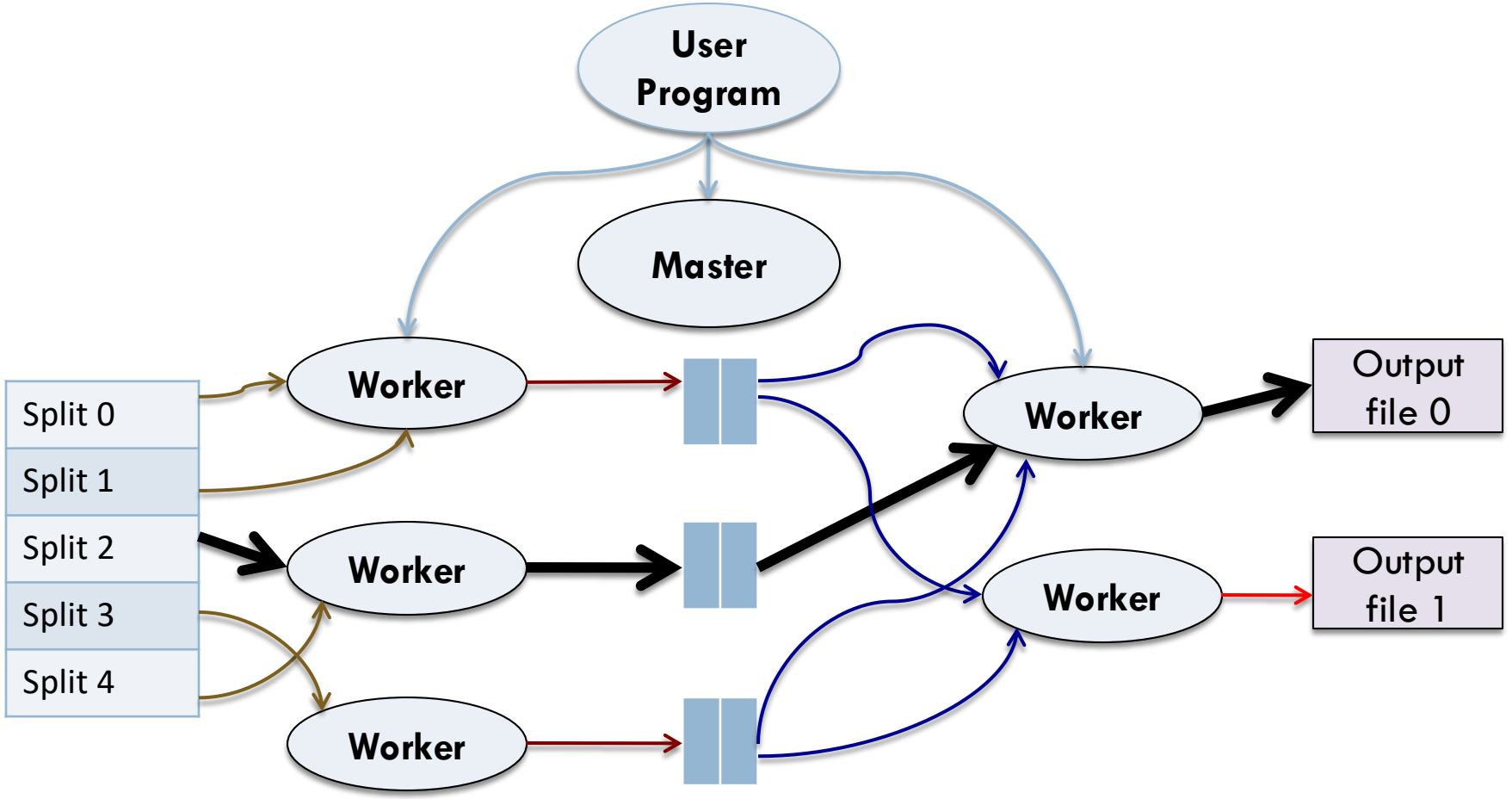
Execution Overview – Part I

- *Maps* distributed across multiple machines
- Automatic partitioning of data into M splits
- Splits are processed **concurrently** on different machines

Execution Overview – Part II

- Partition *intermediate* key space into R pieces
- E.g. hash(key) **mod** R
- User specified parameters
  - **Partitioning** function
  - **Number** of partitions (R)

Execution Overview



Execution Overview: Step 1
The MapReduce library

- Splits input files into **M** pieces
  - 16-64 MB per piece
- Starts up **copies** of the program on a cluster of machines

Execution Overview: Step II
Program copies

- One of the copies is a **Master**
- There are **M** map tasks and **R** reduce tasks to assign
- Master
  - Picks *idle* workers
  - Assigns each worker a map or reduce task

Execution Overview: Step III
Workers that are assigned a map task

- Read contents of their input split
- Parses <*key, value*> pairs out of the input data
- Pass each pair to user-defined *Map* function
- Intermediate <*key, value*> pairs from *Maps*
  - Buffered in Memory

Execution Overview: Step IV
Writing to disk

- Periodically, **buffered pairs** are written to disk
- These writes are partitioned
  - By the partitioning function
- **Locations** of buffered pairs on local disk
  - *Reported* back to Master
  - Master *forwards* these locations to reduce workers

Execution Overview: Step V
Reading Intermediate data

- Master notifies *Reduce* worker about locations
- Reduce worker reads buffered data from the **local disks** of *Maps*
- Read *all* intermediate data; sort by intermediate key
  - All occurrences of the same key are grouped together
  - Many different keys map to the same *Reduce* task

Execution Overview: Step VI
Processing data at the Reduce worker

- Iterate over sorted intermediate data
- For each unique key pass
  - *Key* + set of *intermediate values* to Reduce function
- Output of the Reduce function is appended
  - To output file of the reduce partition

Execution Overview: Step VII
Waking up the user

- After all Map & Reduce tasks have been completed
- Control returns to the user code

Master Data Structures

- For each Map and Reduce task
  - **State**: {*idle*, *in-progress*, *completed*}
  - Worker **machine** identity
- For each completed Map task store
  - **Location** and **sizes** of $\mathbf{R}$ intermediate file regions
- Information pushed incrementally to *in-progress* Reduce tasks

The contents of this slide-set are based on the following references

- *Distributed Systems: Concepts and Design. George Coulouris, Jean Dollimore, Tim Kindberg, Gordon Blair. 5th Edition. Addison Wesley. ISBN: 978-0132143011.* [Chapter 10]
- Jeffrey Dean, Sanjay Ghemawat: *MapReduce: Simplified Data Processing on Large Clusters*. OSDI 2004: 137-150
- Jeffrey Dean, Sanjay Ghemawat: MapReduce: simplified data processing on large clusters. Commun. ACM 51(1): 107-113 (2008)