CS60021: Scalable Data Mining

Sourangshu Bhattacharya

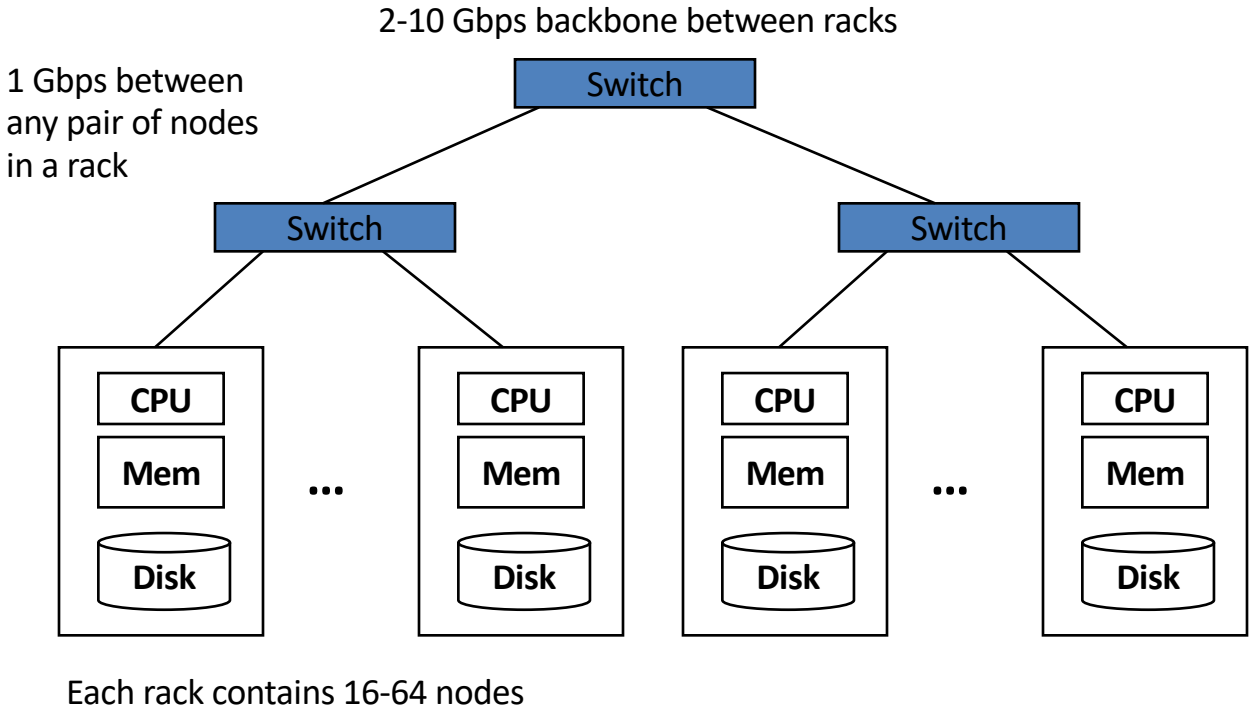
In this Lecture:

- Outline:
  - What is Big Data?
  - Issues with Big Data
  - What is Hadoop ?
  - What is Map Reduce ?
  - Example Map Reduce program.

Motivation: Google Example

- 20+ billion web pages x 20KB = 400+ TB
- 1 computer reads 30-35 MB/sec from disk
  - ~4 months to read the web
- ~1,000 hard drives to store the web
- Takes even more to **do** something useful with the data!
- **Today, a standard architecture for such problems is emerging:**
  - Cluster of commodity Linux nodes
  - Commodity network (ethernet) to connect them

Cluster Architecture



Each rack contains 16-64 nodes

Large-scale Computing

- **Large-scale computing** for **data mining** problems on **commodity hardware**
- **Challenges:**
  - **How do you distribute computation?**
  - **How can we make it easy to write distributed programs?**
  - **Machines fail:**
    - One server may stay up 3 years (1,000 days)
    - If you have 1,000 servers, expect to loose 1/day
    - People estimated Google had ~1M machines in 2011
      - 1,000 machines fail every day!

Big Data Challenges

- Scalability: processing should scale with increase in data.
- Fault Tolerance: function in presence of hardware failure
- Cost Effective: should run on commodity hardware
- Ease of use: programs should be small
- Flexibility: able to process unstructured data

- Solution: Map Reduce !

Idea and Solution

- **Issue: Copying data over a network takes time**
- **Idea:**
  - Bring computation close to the data
  - Store files multiple times for reliability
- **Map-reduce** addresses these problems
  - Elegant way to work with big data
  - **Storage Infrastructure – File system**
    - Google: GFS. Hadoop: HDFS
  - **Programming model**
    - Map-Reduce

Storage Infrastructure

- **Problem:**
  - If nodes fail, how to store data persistently?
- **Answer:**
  - **Distributed File System:**
    - Provides global file namespace
    - Google GFS; Hadoop HDFS;
- **Typical usage pattern**
  - Huge files (100s of GB to TB)
  - Data is rarely updated in place
  - Reads and appends are common

What is Hadoop ?

- A scalable fault-tolerant distributed system for data storage and processing.

- Core Hadoop:
  - Hadoop Distributed File System (HDFS)
  - Hadoop YARN: Job Scheduling and Cluster Resource Management
  - Hadoop Map Reduce: Framework for distributed data processing.

- Open Source system with large community support.
  https://hadoop.apache.org/

What is Map Reduce ?

- Method for distributing a task across multiple servers.
- Proposed by Dean and Ghemawat, 2004.
- Consists of two developer created phases:
  - Map
  - Reduce
- In between Map and Reduce is the Shuffle and Sort phase.
- User is responsible for casting the problem into map – reduce framework.
- Multiple map-reduce jobs can be “chained”.

Programming Model: MapReduce

Warm-up task:

- We have a huge text document
- Count the number of times each distinct word appears in the file
- **Sample application:**
  - Analyze web server logs to find popular URLs

Task: Word Count

Case 1:

- File too large for memory, but all <word, count> pairs fit in memory

Case 2:

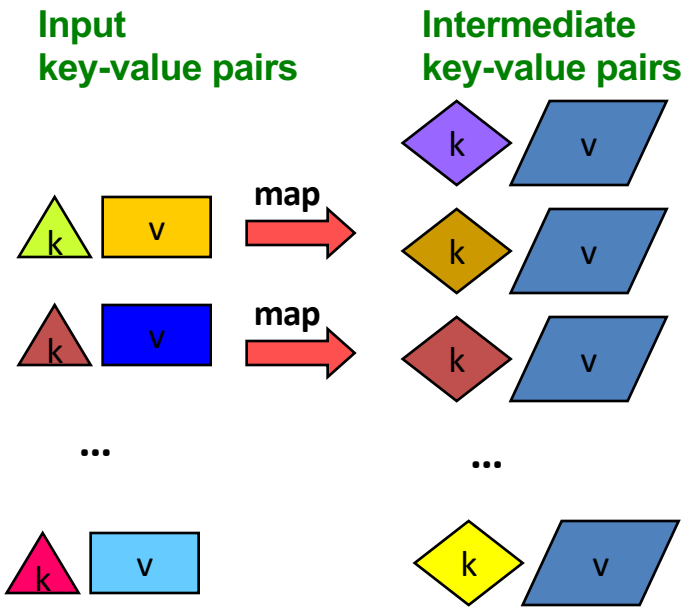
- Count occurrences of words:
  - **`words(doc.txt) | sort | uniq -c`**
    - where **`words`** takes a file and outputs the words in it, one per a line
- Case 2 captures the essence of **MapReduce**
  - Great thing is that it is naturally parallelizable

MapReduce: Overview

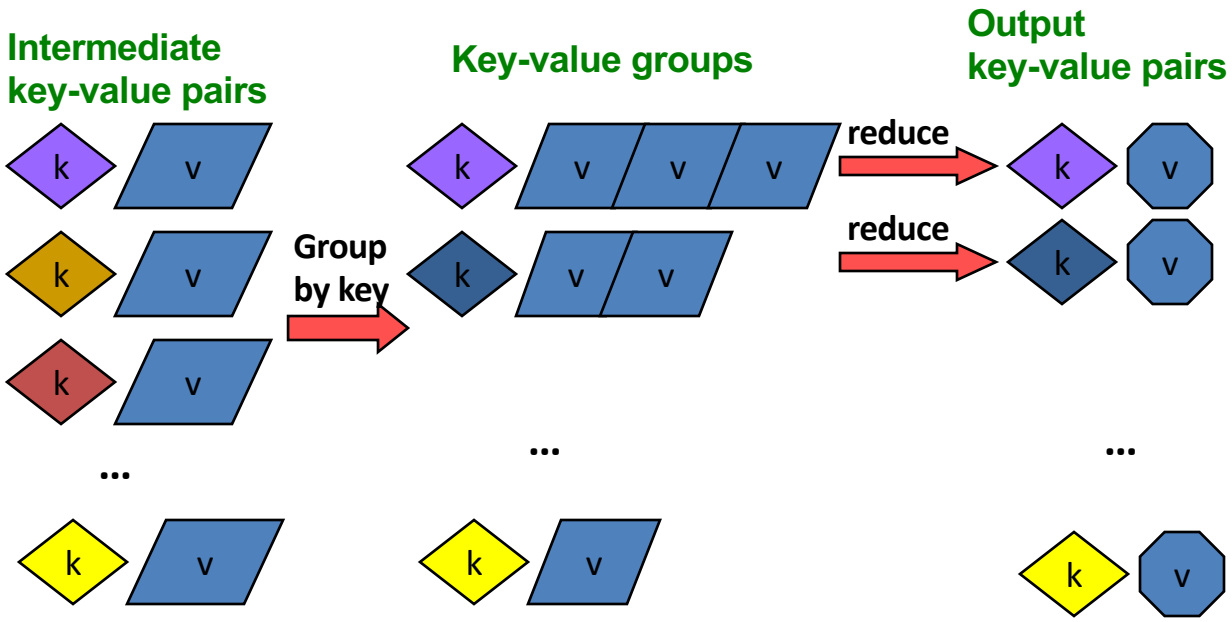
- Sequentially read a lot of data
- **Map:**
  - Extract something you care about
- **Group by key:** Sort and Shuffle
- **Reduce:**
  - Aggregate, summarize, filter or transform
- Write the result

Outline stays the same, **Map** and **Reduce** change to fit the problem

MapReduce: The Map Step



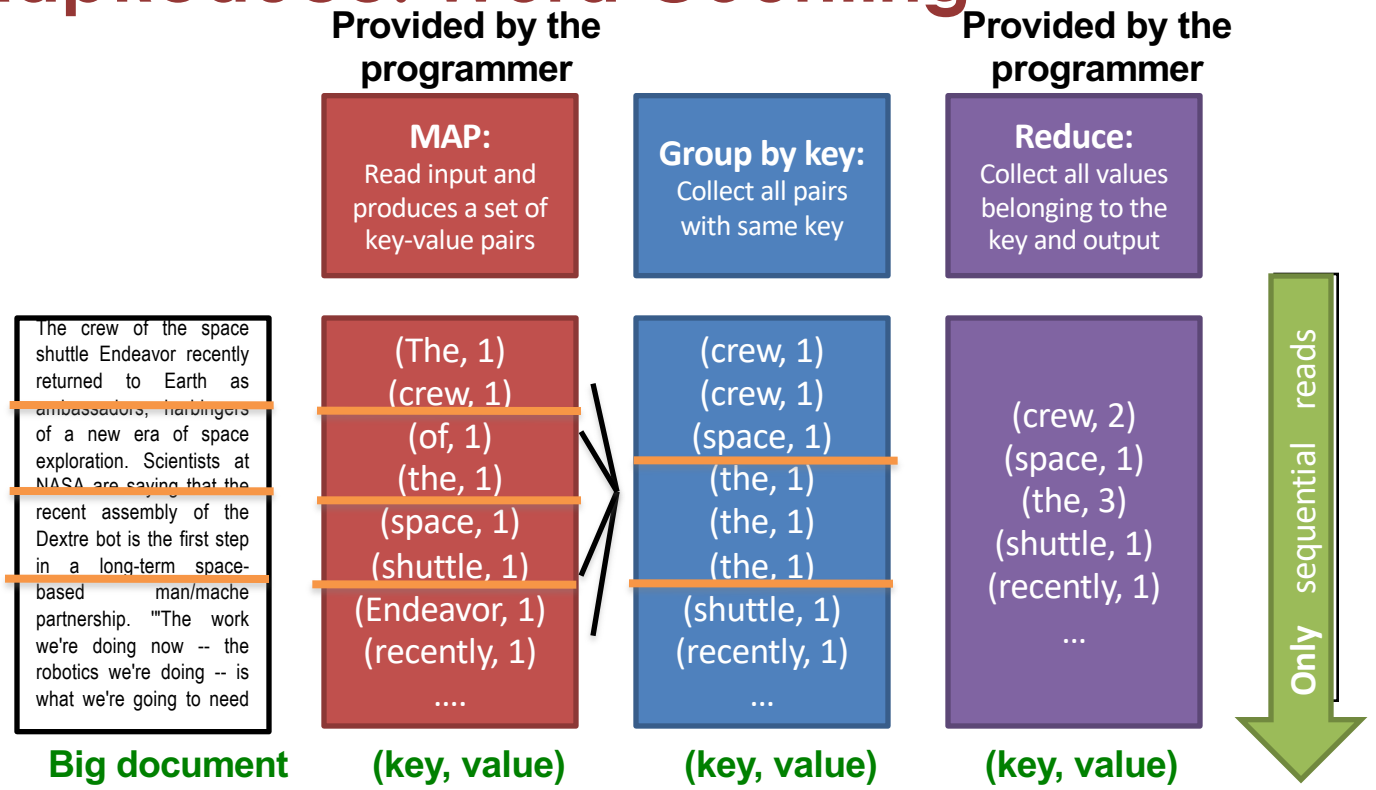
MapReduce: The Reduce Step



More Specifically

- **Input:** a set of key-value pairs
- Programmer specifies two methods:
  - **Map(k, v)** $\rightarrow$ <k', v'>*
    - Takes a key-value pair and outputs a set of key-value pairs
      - E.g., key is the filename, value is a single line in the file
    - There is one Map call for every *(k,v)* pair
  - **Reduce(k', <v'>*)** $\rightarrow$ <k', v''>*
    - **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'*

MapReduce: Word Counting



Word Count Using MapReduce

```
map(key, value):
// key: document name; value: text of the document
   for each word w in value:
    emit(w, 1)


reduce(key, values):
// key: a word; value: an iterator over counts
    result = 0
    for each count v in values:
        result += v
    emit(key, result)
```

Map Phase

- User writes the mapper method.
- Input is an unstructured record:
  - E.g. A row of RDBMS table,
  - A line of a text file, etc
- Output is a set of records of the form: <key, value>
  - Both key and value can be anything, e.g. text, number, etc.
  - E.g. for row of RDBMS table: <column id, value>
  - Line of text file: <word, count>

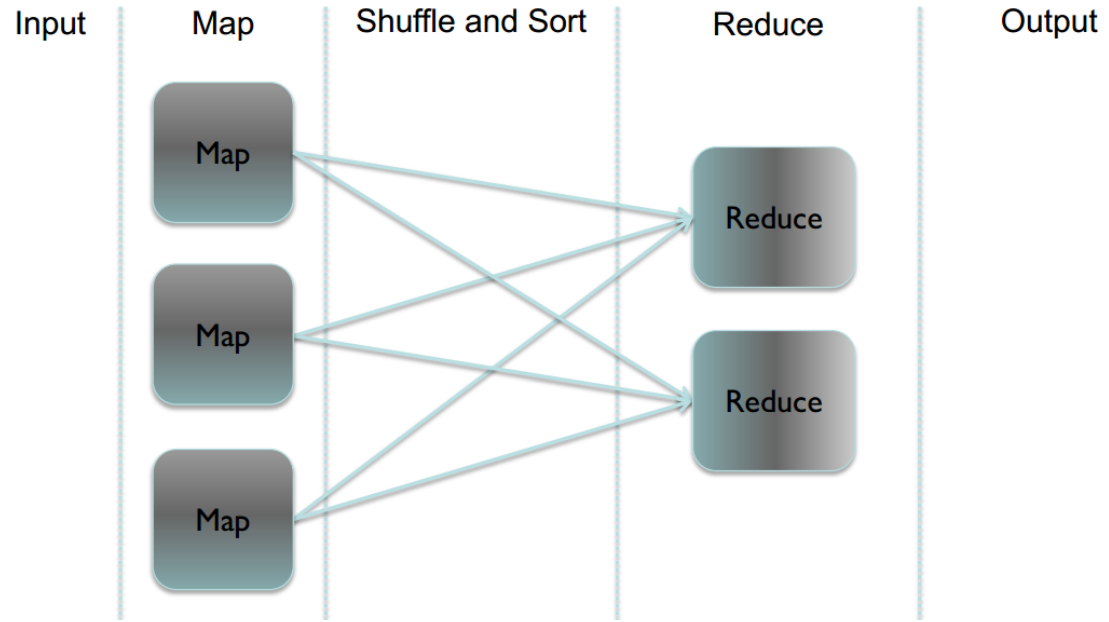
Shuffle/Sort phase

- Shuffle phase ensures that all the mapper output records with the same key value, goes to the same reducer.
- Sort ensures that among the records received at each reducer, records with same key arrives together.

Reduce phase

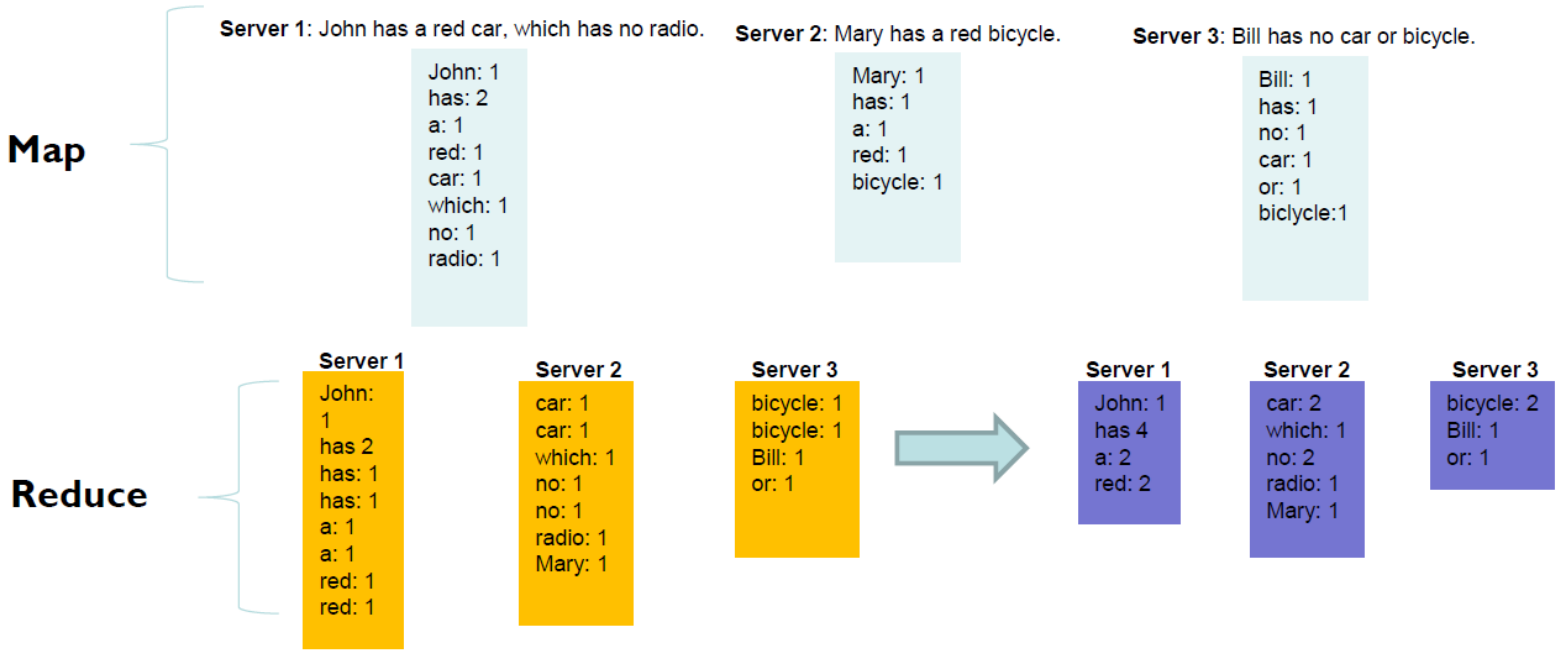
- Reducer is a user defined function which processes mapper output records with some of the keys output by mapper.
- Input is of the form <key, value>
  - All records having same key arrive together.
- Output is a set of records of the form <key, value>
  - Key is not important

Parallel picture

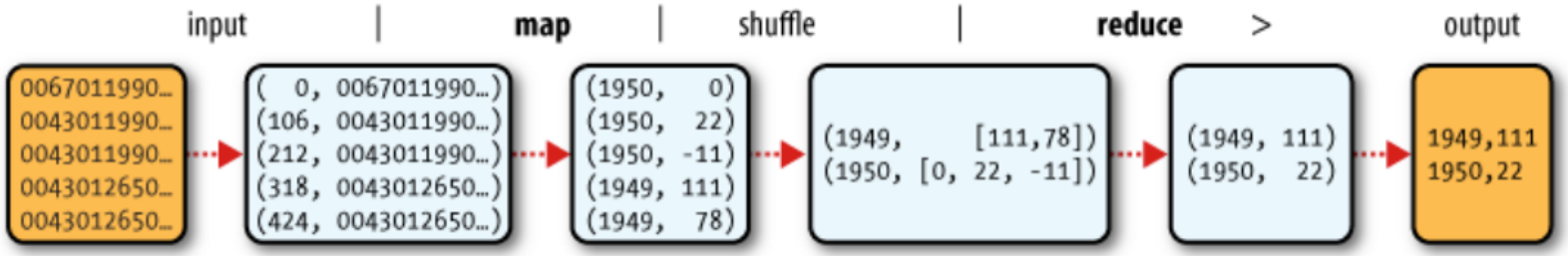


Example

Word Count: Count the total no. of occurrences of each word



Map Reduce - Example



What was the max/min temperature for the last century ?

Hadoop Map Reduce

- Provides:
  - Automatic parallelization and Distribution
  - Fault Tolerance
  - Methods for interfacing with HDFS for colocation of computation and storage of output.
  - Status and Monitoring tools
  - API in Java
  - Ability to define the mapper and reducer in many languages through Hadoop streaming.

References:

- Jure Leskovec, Anand Rajaraman, Jeff Ullman. **Mining of Massive Datasets.** *2nd edition. - Cambridge University Press.* *http://www.mmds.org/*

- Tom White. **Hadoop: The definitive Guide.** Oreilly Press.