Reference

Google Search Statistics

• Google now processes 40,000 search queries every second (on average)
• This is about 3.5 billion searches per day or 1.2 trillion searches per year

• Lots of data to process
Need ....

• A single machine cannot serve all the data

• You need a distributed system to store and process in parallel
However,....

• Parallel programming?
  – Communication between nodes
  – Handle machine failures

• Non-trivial!
Google’s answer

MapReduce and
Underlying infrastructure
MapReduce Overview

• What is it?
  – Programming model used by Google
  – A combination of the Map and Reduce operations
  – Used for analyzing large data sets
MapReduce Overview

• MapReduce is highly scalable and can be used across many computers.

• Many small machines can be used to process jobs that normally could not be processed by a large machine.
Before MapReduce

• Large scale data processing was difficult
  – Managing hundreds or thousands of processors
  – Managing parallelization and distribution
  – I/O scheduling
  – Status and monitoring
  – Fault/crash tolerance
Programming Model

• Programmers specify two functions
  – Map
  – Reduce

• Inspired from map and reduce operations commonly used in functional programming languages like Lisp

• Have multiple workers (processes) on multiple machines run either map or reduce
Map Operation

- Map: \((key_i, value_i) \rightarrow (key_j, value_j)\)
  - Input: A key/value pair
  - Output: A key/value pair

- Evaluation
  - Function defined by user
  - Applies to every value in value input
    - Might need to parse input

- Produces a new list of key/value pairs
  - Can be of different type from input pair
Example Use of Map Operation

• Example Input:
  – Page 1: the weather is good
  – Page 2: today is good
  – Page 3: good weather is good.

• Essentially:
  – The page numbers are keys
  – The page contents are values
Example Use of Map Operation

- The map function is user defined and for this example:
  - Takes each word, \( w \), on the page and creates an intermediate pair
    - \( (w,1) \)

- Assume each page is assigned to a worker (process)
  - Individual pages can be processed in parallel
Example Map output

• Worker 1:
  – (the 1), (weather 1), (is 1), (good 1)

• Worker 2:
  – (today 1), (is 1), (good 1)

• Worker 3:
  – (good 1), (weather 1), (is 1), (good 1)
Reduce Operation

- Reduce: \((\text{key}_{i},[\text{val}_i]) \rightarrow [\text{val}_k]\)
- All the intermediate values for a given output key (resulting from the map operation) are combined together into a list for each intermediate key
  - \((\text{key}_j,[\text{val}_j])\)
- A reduce function is applied to \([\text{val}_j]\) resulting in \([\text{val}_k]\)
Example: Intermediate Key Value Pairs

- (the [1])
- (is [1 1 1])
- (weather [1 1 1])
- (today [1])
- (good [1 1 1 1])

- The above is input to a reduce function which essentially sums the 1’s
MapReduce Workflow

Map
extract something you care about from each record

Reduce
aggregate, summarize, filter, or transform
public static class MapClass extends MapReduceBase
    implements Mapper<LongWritable, Text, Text, IntWritable> {

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(LongWritable key, Text value,
            OutputCollector<Text, IntWritable> output,
            Reporter reporter) throws IOException {
        String line = value.toString();
        StringTokenizer itr = new StringTokenizer(line);
        while (itr.hasMoreTokens()) {
            word.set(itr.nextToken);
            output.collect(word, one);
        }
    }
}
Example Code: Code

• /**
•   * A reducer class that just emits the sum of the input values.
•   */
• public static class Reduce extends MapReduceBase
•   implements Reducer<Text, IntWritable, Text, IntWritable> {
•
•     public void reduce(Text key, Iterator<IntWritable> values,
•       OutputCollector<Text, IntWritable> output,
•       Reporter reporter) throws IOException {
•         int sum = 0;
•         while (values.hasNext()) {
•             sum += values.next().get();
•         }
•         output.collect(key, new IntWritable(sum));
•     }
• }
• }
You write...

• Map, reduce,
• Launching program
  – Defines job
  – Submits job to cluster
More Examples
Example: Distributed Grep (1)

• Find all occurrences of a given pattern in a file (or set of files)

• Input
  – Key: (url+offset)
  – Value: Line

• Map function
  – If line contents match specified pattern, emit (line, 1)
Example: Distributed Grep (2)

- Intermediate key-value pair
  - key: (url+offset)
  - Intermediate value: the list of unique counts

- Reduce function
  - Example of input to reduce is essentially (line, [1,1,1,1])
  - Don’t do anything; just emit line
Example: Count of URL Access Frequency

• Map function
  – Input: <log of web page requests, content of log>
  – Outputs: <URL, 1>

• Reduce function adds together all values for the same URL
Example: Web Structure

• Simple representation of WWW link graph
  – Map
    • Input: (URL, page-contents)
    • Output: (URL, list-of-URLs)

• Who maps to me?
  – Map
    • Input: (URL, list-of-URLS)
    • Output: For each u in list-of-URLS output <u,URL>
  – Reduce: Concatenates the list of all source URLs associated with u and emits ( u, list(URL))
Big Data

• Google knows how to deal with “big data”
• Not just for Google
• Can be used for any data analysis with
  – Large amounts of data that can be split into pieces for processing
Limitations

• Not for real-time analysis
• Not good for streaming data
• Not all data analysis is suitable for MapReduce
  – Example: SVM
• Too many keys – sort is high
Support for MapReduce
Systems Support for MapReduce

- User App
- MapReduce
- Distributed File Systems (Hadoop, Google)
The Infrastructure

• Large clusters of commodity PCs and networking hardware
• Clusters consist of 100/1000s of machines
  – Failures are common
• Google File System (GFS)
  – Distributed file system
  – Provides replication of the data
Infrastructure

- Users submit jobs to a scheduling system
- Possible partitions of data can be based on files, databases, file lines, database records, etc
Execution

• Map invocations are distributed across multiple machines by automatically partitioning the input data into a set of M splits
• The input splits can be processed in parallel by different machines
• Reduce invocations are distributed by partitioning the intermediate key space into R pieces using a hash function: hash(key) mod R
• R is specified by the programmer
MapReduce

Hadoop Program

Master

fork

assign map

fork

assign reduce

Worker

write

Output File 0

Transfer peta-scale data through network

Input Data

Map

local write

reduce

remote read, sort

Reduce

Output Data

split

split

split
Failure in MapReduce

• Failures are the norm in commodity hardware
• Worker failure
  – Detect failure via periodic heartbeats
  – Re-execute task
• Master Failure
  – Single point of failure; resume from execution log
• Robust
  – Google’s experience: Lost 1600 of 1800 machines but finished fine