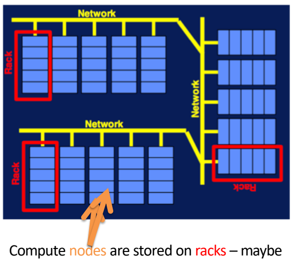
Map-Reduce and Scaling Big Data Processing

Contents

- Understanding Map Reduce
  - Functional programming patterns applied for scalability
- Hadoop
  - Map-reduce in Hadoop
    - Python
    - Java
  - HDFS
- Further reading

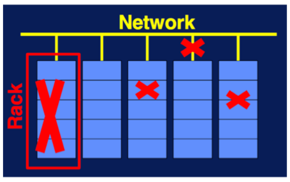
Data centres

- Big Data analysis can be greatly speeded up through parallel and distributed computation.
- Data centres provide parallelism through large **computing clusters** – large collections of commodity hardware, including conventional processors (**"compute nodes"**) connected by Ethernet cables or inexpensive switches



Compute nodes are stored on racks – maybe 8-64 on a rack (5 per rack shown in the diagram).

Fault tolerance



Components (including individual nodes, entire racks and network connections) can fail. The more components a system has, the more likely it is for one to fail.

To avoid having to abort and restart an entire computation every time a component fails, systems should be fault tolerant. This involves:

- file replication at different nodes (e.g., x3 in HDFS)
- division of computation into tasks, such that if one fails to complete it can be restarted without affecting other tasks (e.g., MapReduce)

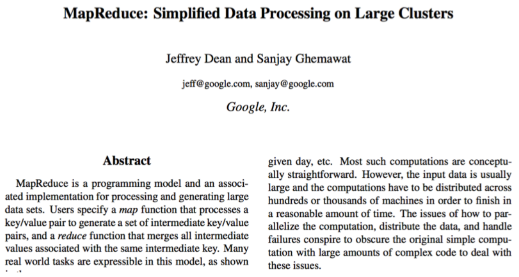
Distributed file systems

- A DFS may be suitable when
  - Files are enormous (maybe a TB or more)
  - Files are rarely updated. They are appended or read sequentially (read or write random access not required)
- Files are divided into **chunks.**
- Typically (e.g., HDFS), chunks are 64MB and are replicated 3 times at 3 different compute nodes (on 3 different racks).
- The **name node** (or master node) for a file stores where the chunks are to be found.
- The name node is replicated and the directory for the file system knows where to find the copies
- The directory itself can be replicated and all users know where the directory copies are.

MapReduce

- a computing paradigm
- developed by Google for indexing web pages and published in 2004
- open-source implementation Hadoop first made available by Yahoo in 2005
- All you write are two functions ***Map()*** and ***Reduce()***
- The system manages the parallel execution and coordination of the tasks that execute Map and Reduce, as well as dealing with the possibility that one of these tasks might fail.

Original 2004 Google Paper



MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

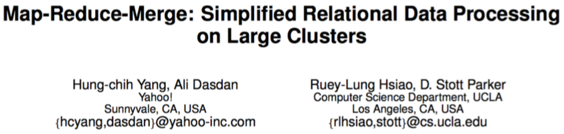
Google, Inc.

Abstract

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a *map* function that processes a key/value pair to generate a set of intermediate key/value pairs, and a *reduce* function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown

given day, etc. Most such computations are conceptually straightforward. However, the input data is usually large and the computations have to be distributed across hundreds or thousands of machines in order to finish in a reasonable amount of time. The issues of how to parallelize the computation, distribute the data, and handle failures conspire to obscure the original simple computation with large amounts of complex code to deal with these issues.

Yahoo 2007



Map-Reduce-Merge: Simplified Relational Data Processing on Large Clusters

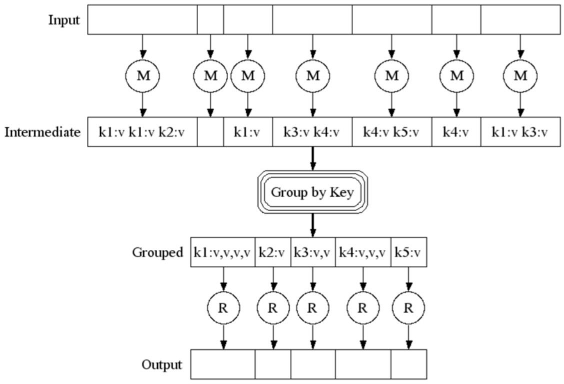
Hung-chih Yang, Ali Dasdan
Yahoo!
Sunnyvale, CA, USA
{hcyang,dasdan}@yahoo-inc.com

Ruey-Lung Hsiao, D. Stott Parker
Computer Science Department, UCLA
Los Angeles, CA, USA
{rlhsiao,stott}@cs.ucla.edu

Class Exercise

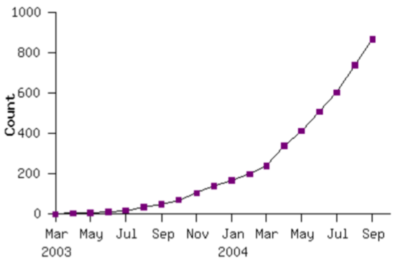
- Find a small piece of paper and write your university and number of siblings on it**.**

Sussex
3



Google's early use of MR

Map Reduce programs in their code repository



A first MapReduce algorithm ...

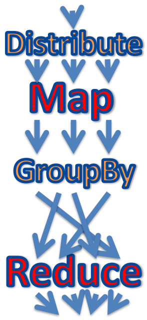
... typically involves counting words

I want to know the frequency of occurrence of the all of the words in a number of texts.

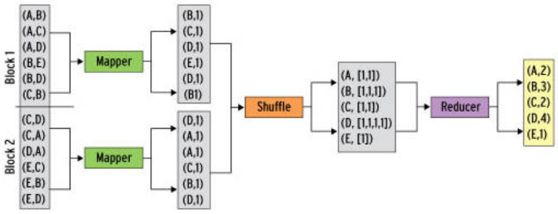
How shall **we** go about computing that distribution efficiently?

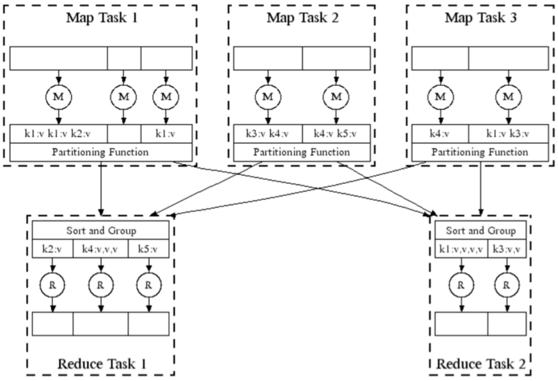
Counting words with MapReduce

- Chunks of the input (individual texts) are distributed to different nodes.
- A worker at each node works out the frequency of each word in its chunk
- Key-value pairs are stored in an intermediate file according to some hash function (the first letter of the word)
- Intermediate files transferred to the node responsible for those key-values pairs
- A worker at that node adds up the associated key-value pairs and produces a single file with the output.



Map/Shuffle/Reduce





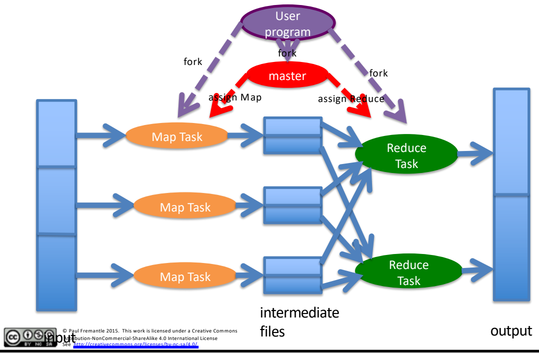
Map Reduce in Real Life

- Analysing web logs
  - Summarise by user / cookie
  - Then aggregate to identify who did what
- Analysing twitter data
  - Who retweeted
  - Who was retweeted the most
- Almost all big data problems can be re-factored into Map Reduce
  - Some more efficiently than others

Tuning

- Fault tolerance
  - Simply re-execute work that fails
- Performance:
  - Partitioning the data
  - Moving the work to near the data

Overview of the Execution of a MapReduce Program

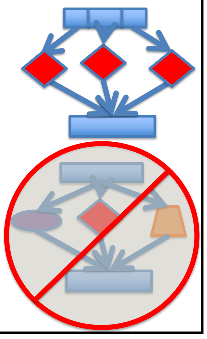


Coping with Node Failures

- If the compute node at which the Master is executing fails, the entire MapReduce job must be restarted.
- Other failures managed by the Master.
- Master periodically *pings* Worker processes to check they are still alive.
- If a Map worker fails, all of the Map tasks assigned to this worker must be redone. The Master reschedules them for a new worker and informs the Reduce tasks where to look for the location of its input
- If a Reduce worker fails, the Master simply reschedules its currently executing Reduce tasks on another reduce worker.

When NOT to use MapReduce

- Don’t use MapReduce for processes which involve relatively little calculation and/or which change the database e.g., interacting with a product database
- MapReduce is good for *data-parallelism* i.e., when the same task needs to be done repeatedly to lots of data
- It is not good for *task-parallelism* i.e., when lots of different tasks need to be done to the same data



Apache Hadoop

- The most famous and popular Map Reduce framework
  - Open Source
    - Written in Java, but supports other languages
  - Runs Map Reduce workloads across a cloud or cluster of machines
  - Supports a distributed filesystem to store data for these jobs
  - Provides reliability when servers in the cluster fail

Components of Hadoop

Map Reduce or Other Workloads

Java, Scala, Python, Apache Pig, Apache Hive, etc

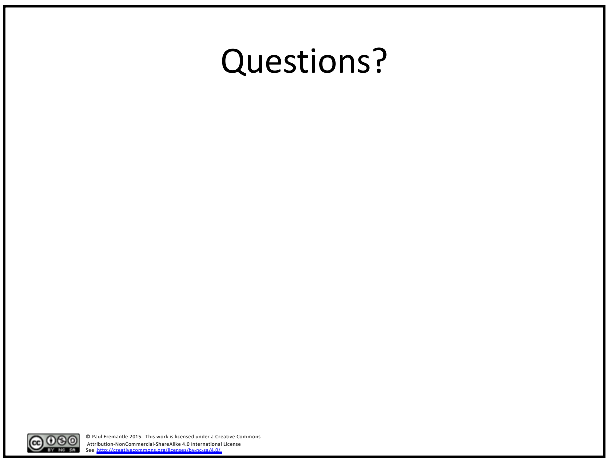
YARN (Yet Another Resource Negotiator)
Cluster Resource Management

Hadoop Distributed File System (HDFS)

Redundant Reliable Distributed File System

Summary

- Understanding the Map Reduce Model
- How is it implemented in Hadoop



Questions?