

Distributed Systems

17. MapReduce

Paul Krzyzanowski
pxk@cs.rutgers.edu

Credit

Much of this information is from the Google Code University:

<http://code.google.com/edu/parallel/mapreduce-tutorial.html>

See also:

<http://hadoop.apache.org/common/docs/current/>
for the Apache Hadoop version

Read this (the definitive paper):

<http://labs.google.com/papers/mapreduce.html>

Background

- Traditional programming is serial
- Parallel programming
 - Break processing into parts that can be executed concurrently on multiple processors
- Challenge
 - Identify tasks that can run concurrently and/or groups of data that can be processed concurrently
 - Not all problems can be parallelized

Simplest environment for parallel processing

- No dependency among data
- Data can be split into equal-size chunks
- Each process can work on a chunk
- Master/worker approach
 - Master:
 - Initializes array and splits it according to # of workers
 - Sends each worker the sub-array
 - Receives the results from each worker
 - Worker:
 - Receives a sub-array from master
 - Performs processing
 - Sends results to master

MapReduce

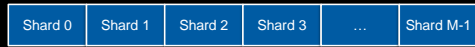
- Created by Google in 2004
 - Jeffrey Dean and Sanjay Ghemawat
- Inspired by LISP
 - Map(function, set of values)
 - Applies function to each value in the set
 - (map 'length '(1) (a) (a b) (a b c)) ⇒ (0 1 2 3)
 - Reduce(function, set of values)
 - Combines all the values using a binary function (e.g., +)
 - (reduce #+ '(1 2 3 4 5)) ⇒ 15

MapReduce

- MapReduce
 - Framework for parallel computing
 - Programmers get simple API
 - Don't have to worry about handling
 - parallelization
 - data distribution
 - load balancing
 - fault tolerance
- Allows one to process huge amounts of data (terabytes and petabytes) on thousands of processors

Step 1: Split input files into chunks (shards)

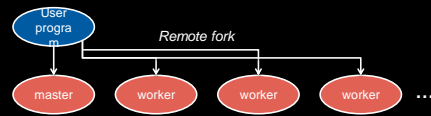
- Break up the input data into M pieces (typically 64 MB)



Input files
Divided into M shards

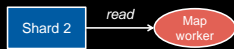
Step 2: Fork processes

- Start up many copies of the program on a cluster of machines
 - 1 master: scheduler & coordinator
 - Lots of workers
- Idle workers are assigned either:
 - map tasks (each works on a shard) – there are M map tasks
 - reduce tasks (each works on intermediate files) – there are R
 - $R = \#$ partitions, defined by the user



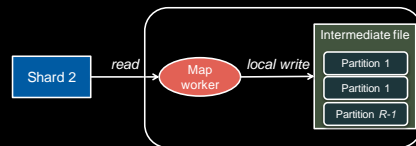
Step 3: Map Task

- Reads contents of the input shard assigned to it
- Parses key/value pairs out of the input data
- Passes each pair to a user-defined *map* function
 - Produces intermediate key/value pairs
 - These are buffered in memory



Step 4: Create intermediate files

- Intermediate key/value pairs produced by the user's *map* function buffered in memory and are periodically written to the local disk
 - Partitioned into R regions by a partitioning function
- Notifies master when complete
 - Passes locations of intermediate data to the master
 - Master forwards these locations to the reduce worker

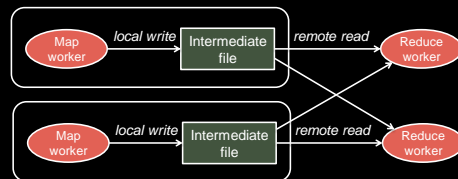


Step 4a. Partitioning

- Map data will be processed by Reduce workers
 - The user's *Reduce* function will be called once per unique key generated by *Map*.
- This means we will need to sort all the (key, value) data by keys and decide which Reduce worker processes which keys – the Reduce worker will do this
- Partition function: decides which of R reduce workers will work on which key
 - Default function: $\text{hash}(\text{key}) \bmod R$
 - Map worker partitions the data by keys
- Each Reduce worker will read their partition from every Map worker

Step 5: Reduce Task: sorting

- Reduce worker gets notified by the master about the location of intermediate files for its partition
- Uses RPCs to read the data from the local disks of the map workers
- When the *reduce* worker reads intermediate data for its partition
 - It sorts the data by the intermediate keys
 - All occurrences of the same key are grouped together



Locality

- Input and Output files are on GFS (Google File System)
- MapReduce runs on GFS chunkservers
- Master tries to schedule *map* worker on one of the machines that has a copy of the input chunk it needs.

Other Examples

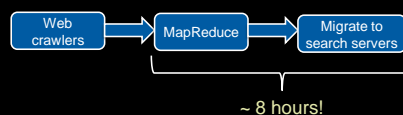
- Distributed grep (search for words)
 - Map: emit a line if it matches a given pattern
 - Reduce: just copy the intermediate data to the output
- Count URL access frequency
 - Map: process logs of web page access; output <URL, 1>
 - Reduce: add all values for the same URL
- Reverse web-link graph
 - Map: output <target, source> for each link to *target* in a page *source*
 - Reduce: concatenate the list of all source URLs associated with a target.
Output <target, list(source)>
- Inverted index
 - Map: parse document, emit <word, document-ID> pairs
 - Reduce: for each word, sort the corresponding document IDs; emits a <word, list(document-ID)> pair. The set of all output pairs is an inverted index

MapReduce Summary

- Get a lot of data
- **Map**
 - Parse & extract items of interest
- **Sort & partition**
- **Reduce**
 - Aggregate results
- Write to output files

All is not perfect

- MapReduce was used to process webpage data collected by Google's crawlers.
 - It would extract the links and metadata needed to search the pages
 - Determine the site's PageRank
- The process took around eight hours.
 - Results were moved to search servers.
 - This was done continuously.



In Practice

- Most data not simple files
 - B-trees, tables, SQL databases, memory-mapped key-values
- Hardly ever use textual data: slow & hard to parse
 - Most I/O encoded with Protocol Buffers

All is not perfect

- Web has become more dynamic
 - an 8+ hour delay is a lot for some sites
- Goal: refresh certain pages within seconds
- MapReduce
 - Batch-oriented
 - Not suited for near-real-time processes
 - Cannot start a new phase until the previous has completed
 - Reduce cannot start until all Map workers have completed
 - Suffers from "stragglers" – workers that take too long (or fail)
 - This was done continuously
- MapReduce is still used for many Google services
- Search framework updated in 2009-2010: Caffeine
 - Index updated by making direct changes to data stored in BigTable
 - Data resides in Colossus (GFS2) instead of GFS

More info

- Good tutorial presentation & examples at:
<http://research.google.com/pubs/pub36249.html>
- The definitive paper:
<http://labs.google.com/papers/mapreduce.html>

The End