MapReduce
Simplified Data Processing on Large Clusters
by J. Dean and S. Ghemawat

Stefanos Laskaridis
sl829@cam.ac.uk
Structure

• MapReduce motives

• Programming Model & Architecture

• Comparison with relevant work

• Results

• Critique
Disclaimer

• We will not refer to:
  
  • GFS/HDFS $[2,4]$

  • Hadoop $[4]$
MapReduce Motives
A use case
A LONG LONG TIME AGO (2004), IN A GALAXY NOT SO FAR AWAY, THERE WERE PROGRAMMERS WHO WANTED TO RUN DISTRIBUTED JOBS.

A BIG COMPANY, NAMED GOOGLE, WAS RUNNING MANY OF THOSE.

IMAGINE RUNNING A QUERY OF HOW MANY GOOGLE SEARCHES A USER IN CAMBRIDGE DOES DURING MICHAELMAS TERM.

WHAT WOULD YOU DO?
Approach

- Write a job that would scan through the data and calculate the average.
- You would probably want it to be distributed.

1. Find an interface to the distributed filesystem or distribute the data.
2. Write a parallel program that splits the work in many threads/processes.
3. Make sure that you handle hardware or other failures with minimal data losses.
4. Get intermediate results (may not fit in one machine memory)
5. Write and execute your query
Problem

• Too much focus on preparing the workflow rather than the actual computation.

• Complex code that obscures the actual implementation.

  • Generally harder to understand

  • and maintain
MapReduce (MR)

- A programming model

- Based on 2 functions of functional programming
  
  - `map()`: `(k, v) => list(k1, v1)`
    execute a function for every element in a collection
  
  - `reduce()`: `(k1, v1) => list(v2)`
    aggregate results by key based on a function
Notable Refinements

• Partitioning function
• Ordering guarantee
• Skipping bad records
• Backup tasks
• Distributed counters
• Status information infrastructure (HTTP server)
Failure Semantics

- Master pings workers
- Map worker failure => re-execute map
  - Failed map execution
  - Error after map execution (data still on local disk)
- Reduce worker failure => re-execute reduce
Relevant Work
Relevant Work

“Simplification and distillation of some […] models” [1]
Relevant Work

BSP/MPI
Higher level of abstraction
No transparent fault-tolerance

River
Non-skewed completion times through careful scheduling*

vs

MapReduce

* vs fine-grained task partitioning
Results
Experiment Setup

Grep experiment
- $10^{10}$ 100-byte records (1TB of data)
- Text occurrence: 0.00092336%
- $M=15,000$ (64MB)
- $R=1$

Sorting experiment
- $10^{10}$ 100-byte records (1TB of data)
- 10-byte sort key
- $M=15,000$ (64MB)
- $R=4,000$

Equipment
- 2GHz Intel® Xeon® Processors with HyperThreading
- 160GB IDE disks
- 4GB of memory (2-2.5GB available)
- Gigabit Ethernet link
- 100-200Gbps aggregate bandwidth
Results

• **Grep task**
  Average throughput: ~66GHz

• **Sort task**
  Average throughput: ~11GHz

• Very scalable*

• Backup tasks and fault tolerance do work

• ~81% code reduction for Google’s Web Search service production indexing system

• **Usage**
  • Machine Learning algorithms
  • Clustering for Google News
  • Reports for popular queries for Google Zeitgeist
  • Properties extraction from crawled webpages
  • Graph computations

* s.t. Amdahl's law
Why MapReduce?

- Abstraction for programmer
- Automatic parallelisation
- Almost linear scalability
- Load-balancing
- Fault-tolerance

- Locality optimisation
- Runs on commodity hardware
- Easy large-scale prototyping
Critique
Restrictive Model

• The model of execution is too restrictive.

• The same `map()` and `reduce()` function on all data. Only allows for data parallelisation.

• Inefficient for iterative update algorithms. Need of job pipelining. [6] (e.g. many Machine learning algorithms)
Optimisations

• No distributed data query plan
• No context awareness between different jobs
• Large startup time for job propagation
• No caching or indexing [5,6]
Considerations on the MR Master

• Single point of failure

• At scale, point of congestion for communications
Disk seeks

- Pull-mode remote reads from reducers
- Multiple reduce workers reading different files from the same map worker, leads to high disk seek times [5].
“We don’t really use MapReduce anymore”

– Urs Hölzle [3]
SVP Technical Infrastructure Google
Thank you

Q&A

Stefanos Laskaridis
sl829@cam.ac.uk


