MapReduce

Hui Chen [a]

[a]CUNY Brooklyn College

September 3, 2025

MapReduce Programming Pattern vs. MapReduce System

Is there a difference?

Motivational Problem

Carry out a large-scale data processing task efficiently.

- Build search index of web programmers
- Sort web pages
- Analyze structure of web

Need 1,000s computers, do hours of computations, process multi-terabyte of data

- how to parallelize the computation,
- how to distribute the data, and
- how to handle failures

which requires large amount of complex code that obscure the original simple computation

Engineering problem: how to make it easy for non-specialist programmers?

MapReduce Job: Example

- split input into M pieces
- Map: calls `Map()` for each split, yield "intermediate" data, a list of k,v pairs
  - Each `Map()` call is a "task"
- Reduce: collect all intermediate values for each key and passes them to a `Reduce()` call
- Final output is a set of k,v pairs from `Reduce()`s

Example: Counting Words

- Map(1, Input1) $\rightarrow$ a,1 b,2
- Map(2, Input2) $\rightarrow$ b,3
- Map(3, Input3) $\rightarrow$ a,2 c,1

Then,

- Reduce(a, [(a, 1), (a, 2)]) $\rightarrow$ a,3
- Reduce(b, [(b, 2), (b, 3)]) $\rightarrow$ b,5
- Reduce(c, [(c, 1)]) $\rightarrow$ c,1

Scalability

N worker computers can process data in parallel, may yield N times throughput

Reducing Complexity

MapReduce system:

- distributes data and code to servers
- tracks which map/reduce task have finished
- shuffles intermediate data from Map tasks to Reduce tasks.
- Balances load over servers/computers
- Recovers from failed servers

Design Consideration

Applications are restricted:

- No interaction or state (other than via intermediate output) among Map/Reduce tasks.
- One Map/Reduce pattern for data flow.
- No real-time or streaming processing.

Distributed File System

- There is a need to split files over many servers, many disks, in a fixed size chunk
- There is need to support parallel read/write
- There is a need to tolerate data access failures (disk failures/network failures)

MapReduce Coordinator

- Send Map tasks to worker servers until all Map tasks complete
  - A Map task splits its output, by hash(key) mod R, into one file to local disk
  - This file will serve as input for a Reduce task
- After all Map tasks have finished, the coordinator starts Reduce tasks
  - Each Reduce task corresponds to one hash bucket of intermediate output
  - Each Reduce task fetches its bucket from every Map worker
  - Each Reduce task writes a separate output file

Evaluation

- What is the performance bottleneck?

Network Use

- Map tasks usually read inputs from local computers – no network use
- Intermedia data are transmitted only once over the network – Reduce workers read from over the network
- Reduce task unit's input is a hash bucket – big network transfers are more efficient

Load Balancing

Keep servers busy

- Many more tasks than workers
- Coordinator assigns new tasks to free workers
- Coordinator gives more tasks to fast servers, and less work to slow servers

Fault Tolerance

We want to hide failures from the application programmer – reruns just the failed Map and Reduce tasks

- Worker crashes: coordinator re-assigns tasks to other workers
- Worker is slow: coordinator re-assigns its task to another worker
- Worker returns incorrect output: too bad, MapReduce system assumes "fail-stop" CPUs and software
- Coordinator crashes: too bad, MapReduce system assumes "fail-stop" CPUs and software

Conclusion

It makes cluster computation easier for programmers.

- Advantage: Scales well and easy to program
- But not the most efficient and flexible

Chambers, Craig, Ashish Raniwala, Frances Perry, Stephen Adams, Robert R. Henry, Robert Bradshaw, and Nathan Weizenbaum. "FlumeJava: easy, efficient data-parallel pipelines." ACM Sigplan Notices 45, no. 6 (2010): 363-375.