Presentation 2

MapReduce: simplified data processing on large clusters

Dean, Jeffrey, and Sanjay Ghemawat

Communications of the ACM 51, no. 1 (2008): 107-113

Pig latin: a not-so-foreign language for data processing

Olston, Christopher, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins

In Proceedings of the 2008 ACM SIGMOD international conference on Management of data, pp. 1099-1110. ACM, 2008.

John Berlin

October 6, 2016

Old Dominion University
Introduction to Information Retrieval
CS734/834

In The Beginning There Was Big Data

- **What Is It?**
  - Crawled Documents
  - Log Files
  - Databases
- **How Big Is Big Data?**
  - More Data Than A Single Computer Can Handle
  - > 1TB
- **How Do We Process It?**
  - 100 Computers with 1/100 Of The Data?
  - Distribute It All Over The Network?
  - How Are We Going To Coordinate Our Machines?
  - Do Our Programmers Need To Learn A New Language/Tool For This?

"Data: My programming may be inadequate to the task.
Counselor Troi: We're all more than the sum of our parts, Data. You'll have to be more than the sum of your programming."
In Theory, TNG

https://twitter.com/wtrsld/status/424272437929205760

MapReduce: Simplified Data Processing on Large Clusters

Dean & Ghemawat’s Contribution

- **Created A Framework That Abstracted Away The Complexity From**
  - Parallelization Of The Computation
  - Distribution Of The Data
- **“Devised” A Programming Model To Harness The Framework**
  - Write Smaller Code Modules → Quick Big Data Processing

The MapReduce Programming Model

- **Computation By Two Functions**
  - Map: $(K_1,V_1) \rightarrow List(K_2,V_2)$
  - Reduce: $(K_2, List(V_2)) \rightarrow List(V_3)$
- **Functions Bound By A Contract**
  - Take A Single Input Format
  - Produce Single Output Format
- **Need Only These Two Functions To Start Map And Reducing**

```
map(String key, String value):
  // key: document name
  // value: document contents
  for each word w in value:
    EmitIntermediate(w, "1");

reduce(String key, Iterator values):
  // key: a word
  // values: a list of counts
  int result = 0;
  for each v in values:
    result += ParseInt(v);
  Emit(AsString(result));
```

Word Count Example
section 2.1, p2

MapReduce Is Really Functional Programming λ

- **Pure Functions**
  - Contract Ensures This
  - Input Is Not Modified
  - Produces Same Output Given The Same Input
- **Higher Order Functions**
  - Output Of One Function Is The Input Of The Next
  - Composable: $Reduce(Map(Reduce(Map(K_1,V_1))))$ $Reduce(Map(Map(K_1,V_1)))$
- **Map Only Transforms**
  - Given Input Produces Zero Or More K,V Pairs
- **Reduce Only Accumulates**
  - Combines A List Of V For a Given K

```
map(String key, String value):
  // key: document name
  // value: document contents
  for each word w in value:
    EmitIntermediate(w, "1");

reduce(String key, Iterator values):
  // key: a word
  // values: a list of counts
  int result = 0;
  for each v in values:
    result += ParseInt(v);
  Emit(AsString(result));
```

MapReduce Framework

- **Input Split Into M Splits**
  - Split Size Typically 16-64MB
- **Map Function Distributed**
  - Splits M = Number Of Mappers
- **Reduce Function Distributed**
  - Distribution Based On User Configured Number R
  - Feed Data Based On Partitioning
  - Partition Function Typically: Hash(key) Mod R

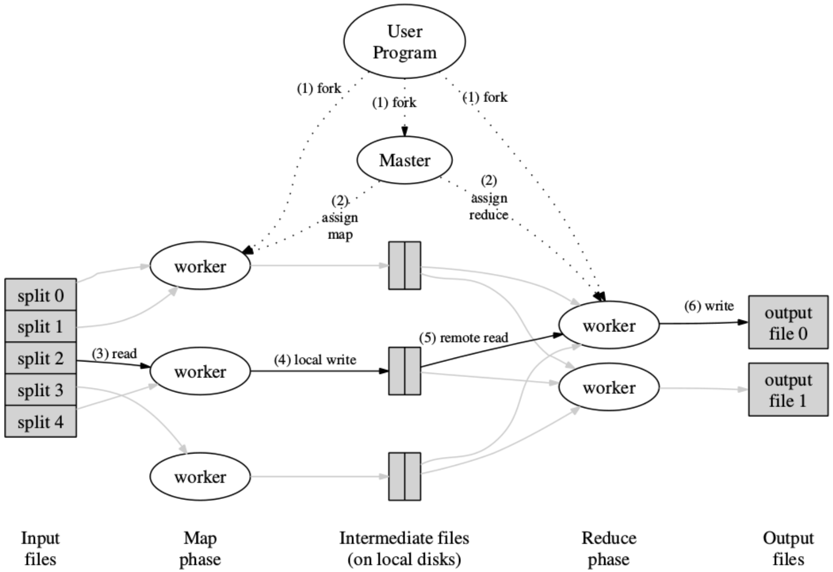

Figure 1: Execution overview

Behind The Scenes Framework Execution

- **Data Lives On The Worker Machines**
  - Split Up On Start Of A MR Job
- **Map Tasks**
  - Master Schedules Map Tasks
  - On End Are Rescheduled If Machine Still Has Unprocessed Data
  - On Failure Master Starts Another On Same Machine Or Machine Close To The Data i.e Same Local Network Switch
- **Reduce Tasks**
  - Constantly Work As Input Is Sorted And Uniquely Keyed
  - Reads Data From Mapper File System Via Network
  - Writes To Final Output Destination

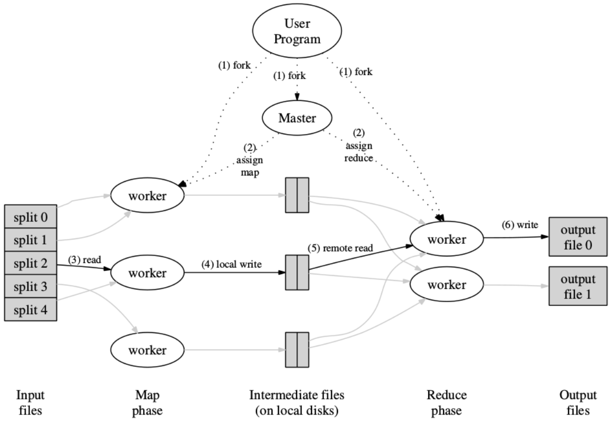

Figure 1: Execution overview