CS347: MapReduce
Motivation for Map-Reduce

Distribution makes simple computations complex
- Communication
- Load balancing
- Fault tolerance
- ...

What if we could write “simple” programs that were automatically parallelized?
Motivation for Map-Reduce

Recall one of our sort strategies:

1. Process data & partition
2. Additional processing
Another example: Asymmetric fragment + replicate join

Local join

Result

process data & partition

additional processing

union
From our point of view…

• What if we didn’t have to think too hard about the number of sites or fragments?

• MapReduce goal: a library that hides the distribution from the developer, who can then focus on the fundamental “nuggets” of their computation
Building Text Index - Part I

original Map-Reduce application....

Page stream

Loading

Tokenizing

Sorting

FLUSHING

Intermediate runs

Disk

CS347
Building Text Index - Part II

Intermediate Runs

Merge

Final index

CS347
Generalizing: Map-Reduce
Generalizing: Map-Reduce

Intermediate Runs

Merge

Reduce

Final index

CS347
Map Reduce

- Input: \( R=\{r_1, r_2, \ldots, r_n\} \), functions \( M, R \)
  - \( M(r_i) \rightarrow \{ [k_1, v_1], [k_2, v_2], \ldots \} \)
  - \( R(k_i, \text{valSet}) \rightarrow [k_i, \text{valSet}'] \)
- Let \( S=\{ [k, v] | [k, v] \in M(r) \text{ for some } r \in R \} \)
- Let \( K = \{ k | [k, v] \in S, \text{ for any } v \} \)
- Let \( G(k) = \{ v | [k, v] \in S \} \)
- Output = \( \{ [k, T] | k \in K, T=R(k, G(k)) \} \)
References

Example: Counting Word Occurrences

• map(String doc, String value);
  // doc is document name
  // value is document content
  for each word w in value:
    EmitIntermediate(w, “1”);

• Example:
  – map(doc, “cat dog cat bat dog”) emits
    [cat 1], [dog 1], [cat 1], [bat 1], [dog 1]
Example: Counting Word Occurrences

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

• Example:
  – reduce(“dog”, “1 1 1 1”) emits “4”

Becomes (“dog”, 4)
Mappers

Source data → Split data → Workers

CS347
Mappers

Split data

Workers

CS347
Mappers

Split data

Workers
Mappers

Split data

Workers
Mappers

Split data

Workers

CS347
Mappers

Split data

Workers

CS347
Mappers

Split data

Workers

CS347
Mappers

Split data

Workers
Mappers

Split data

Workers

CS347
Mappers

Split data

Workers

CS347
Mappers

Split data

Workers
Mappers

Split data

Workers
Shuffle

Workers
Reduce

Workers
Another way to think about it:

• Mapper: (some query)
• Shuffle: GROUP BY
• Reducer: SELECT Aggregate()
 Doesn’t have to be relational

- Mapper: Parse data into K,V pairs
- Shuffle: Repartition by K
- Reducer: Transform the V’s for a K into a V_{final}
Process model

Figure 1: Execution overview
Process model

Can vary the number of mappers to tune performance

Reduce tasks bounded by number of reduce shards

Figure 1: Execution overview
Implementation Issues

• Combine function
• File system
• Partition of input, keys
• Failures
• Backup tasks
• Ordering of results
Combine Function

Combine is like a local reduce applied before distribution:
Data flow

worker must be able to access any part of input file; so input on distributed fs

reduce worker must be able to access local disks on map workers

any worker must be able to write its part of answer; answer is left as distributed file

High-throughput network is essential

Figure 1: Execution overview
Partition of input, keys

- How many workers, partitions of input file?

How many splits?

How many workers? Best to have many splits per worker: Improves load balance; if worker fails, easier to spread its tasks.

Should workers be assigned to splits “near” them?

Similar questions for reduce workers.
What takes time?

• Reading input data
• Mapping data
• Shuffling data
• Reducing data

• Map and reduce are separate phases
  – Latency determined by slowest task
  – Reduce shard skew can increase latency
• Map and shuffle can be overlapped
  – But if lots intermediate data, shuffle may be slow
Failures

- Distributed implementation should produce same output as would have been produced by a non-faulty sequential execution of the program.
- General strategy: Master detects worker failures, and has work re-done by another worker.

CS347
Backup Tasks

• Straggler is a machine that takes unusually long (e.g., bad disk) to finish its work.
• A straggler can delay final completion.
• When task is close to finishing, master schedules backup executions for remaining tasks.

Must be able to eliminate redundant results
Ordering of Results

- Final result (at each node) is in key order

also in key order:

\[[k_1, v_1] \quad [k_3, v_3]\]
Example: Sorting Records

Map: extract k, output [k, record]

Reduce: Do nothing!
Other Issues

• Skipping bad records
• Debugging
MR Claimed Advantages

- Model easy to use, hides details of parallelization, fault recovery
- Many problems expressible in MR framework
- Scales to thousands of machines
MR Possible Disadvantages

• 1-input 2-stage data flow rigid, hard to adapt to other scenarios
• Custom code needs to be written even for the most common operations, e.g., projection and filtering
• Opaque nature of map, reduce functions impedes optimization
Hadoop

• Open-source Map-Reduce system
• Also, a toolkit
  – HDFS – filesystem for Hadoop
  – HBase – Database for Hadoop
• Also, an ecosystem
  – Tools
  – Recipes
  – Developer community
MapReduce pipelines

- Output of one MapReduce becomes input for another
  - Example:
    - Stage 1: Translate data to canonical form
    - Stage 2: Term count
    - Stage 3: Sort by frequency
    - Stage 4: Extract top-k
Make it database-y?

• Simple idea: each operator is a MapReduce
• How to do:
  – Select
  – Project
  – Group by, aggregate
  – Join
Reduce-side join

• Shuffle puts all values for the same key at the same reducer

• Mapper
  – Input: tuples from R; tuples from S
  – Output: (join value, (R|S, tuple))

• Reducer
  – Local join of all R tuples with all S tuples
Map-side join

- Like a hash-join, but every mapper has a copy of the hash table

- Mapper:
  - Read in hash table of R
  - Input: Tuples of S
  - Output: Tuples of S joined with tuples of R

- Reducer
  - Pass through
Comparison

• Reduce-side join shuffles all the data
• Map-side join requires one table to be small
Semi-join?

• One idea:
  – MapReduce 1: Extract the join keys of R; reduce-side join with S
  – MapReduce 2: Map-side join result of MapReduce 1 with R
Platforms for SQL-like queries

- Pig Latin
- Hive
- MapR
- MemSQL
- …
Why not just use a DBMS?

- Many DBMSs exist and are highly optimized

Figure 7: Aggregation Task Results (2.5 million Groups)

- A comparison of approaches to large-scale data analysis. Pavlo et al, SIGMOD 2009
Why not just use a DBMS?

- One reason: loading data into a DBMS is hard

Figure 2: Load Times – Grep Task Data Set (1TB/cluster)

- A comparison of approaches to large-scale data analysis. Pavlo et al, SIGMOD 2009
Why not just use a DBMS?

• Other possible reasons:
  – MapReduce is more scalable
  – MapReduce is more easily deployed
  – MapReduce is more easily extended
  – MapReduce is more easily optimized
  – MapReduce is free (that is, Hadoop)
  – I already know Java
  – MapReduce is exciting and new
Data store

• Instead of HDFS, data could be stored in a database

• Example: HadoopDB/Hadapt
  – Data store is PostgresSQL
  – Allows for indexing, fast local query processing

Batch processing?

• MapReduce materializes intermediate results
  – Map output written to disk before reduce starts

• What if we pipelined the data from map to reduce?
  – Reduce could quickly produce approximate answers
  – MapReduce could implement continuous queries

MapReduce Online. Tyson Condie, Neil Conway, Peter Alvaro, Joe Hellerstein, Khaled Elmeleegy, Russell Sears. NSDI 2010
Not just database queries!

• Any large, partition-able computation
  – Build an inverted index
  – Image thumbnailing
  – Machine translation
  – …

  – Anything expressible in the Map/Reduce functions via general purpose language