Roadmap

- Previous class – Core Hadoop
- This class – the MR Implementation
  - MR Runtime a Nutshell
  - Performance Optimizations
Hadoop Map-Reduce

http://hortonworks.com/hdp/

CS236620 Big Data Technology
Historical Perspective

- GFS 2003
- MapReduce 2004
- Yahoo Hadoop 2006
- Open Source 2008-9
- Hadoop 2.0 2013

CS236620 Big Data Technology
Recap

- A simple paradigm for **batch** processing
  - **Data**- and **computation**-intensive jobs

- Simplicity is key for scalability
  - SQL databases just didn’t scale

- No silver bullet
  - Iterative modeling - MPI is better
  - Graph processing - Google Pregel, Appache Giraph
Recap: Crash Course in MR

**Map**(doc)
(docid, tokens[]) = parse(doc)

**foreach** tok (tokens[])
**output**(tok, docid)

**Reduce**(tok, docids[])
**output**(tok, docids[])

Single record $\rightarrow$ Multiple <key, value> pairs

In parallel across many data slices ...

Aggregate values for the same key

In parallel across many keys ...

**Task** Parallelism

CS236620 Big Data Technology
Example: MR job on 2 Machines

Synchronous execution: every R starts computing after all M’s have completed

Input splits (on HDFS)

Output (on HDFS)
Storage

- Job input and output are stored on HDFS
  - Replicated, reliable storage
  - A mapper consumes one or more input splits
  - A reducer typically creates one output split

- Intermediate files reside on local disks
  - Non-reliable
  - A mapper writes one file for each reducer
  - A reducer pulls one file from each mapper

Unless #reducers=0
Mapper versus Reducer (1)

- Mapper (stages happen in parallel)
  - **Scan** (few large) input splits from HDFS [S]
  - **Compute**: window=1, typically selection/projection [U]
  - **Sort**: sort & write intermediate outputs [S]

- Reducer (stages happen sequentially)
  - **Shuffle**: copy (many small) intermediate files [S]
  - **Sort**: merge by key [S]
  - **Compute**: window = group size, typically aggregation [U]
    write-back the output to HDFS [S]
Mapper versus Reducer (2)

- Mapper
  - Low fan-in
  - I/O bound (linear scan)
  - Might be CPU-bound

- Reducer
  - High fan-in
  - Network-bound (shuffle)
  - Might be RAM-bound (sort)

- Require different optimizations
- #mappers and #reducers are independent parameters
Map-Reduce Job Dynamics

- A job proceeds in execution waves
  - Typically, \#tasks >> \#machines
- Synchronous execution
  - Neither of the reducers can start the compute stage before all the mappers complete
- The shuffle and sort stages can overlap
  - Hadoop’s rule of thumb: start scheduling reducers when 75% of the mappers complete
- Data transfer rate over time

CS236620 Big Data Technology
Anatomy of a Map-Reduce Job

Map Phase

Reduce Phase

INIT

EXECUTION

SHUFFLE (local)

SPILLING

Job Start

time

Job Finish

Task Start

set up
map clean

time

Task Finish

Source: http://ercoppa.github.io/HadoopInternals/AnatomyMapReduceJob.html

CS236620 Big Data Technology
Mapper Optimization: Locality

- Let’s exploit HDFS’s redundancy in storage
  - Push the computation as close as possible to the replicas
  - → avoid data transfers over the network

- Applies only to mappers (why?)

- Implied design choice: non-tiered storage & compute

- Scheduler’s priorities (Hadoop)
  - Local replica, then rack-local replica, then all the rest
Combiners

- Often, the reducer does is simple aggregation
  - Sum, average, min/max, ...
  - Commutative and associative functions

\[
\max(X \cup Y \cup Z) = \max(\{\max(X), \max(Y), \max(Z)\})
\]
\[
\sum(X \cup Y \cup Z) = \sum X + \sum Y + \sum Z
\]

- We can do some aggregation at the mapper side
  - ... and eliminate a lot of network traffic!
- Extensively used by query languages on top of MR
Straggler Tasks

Slowest task (straggler) affects the job latency

Input (on HDFS) → M1 → R1 → Output (on HDFS)
Input (on HDFS) → M2 → R1 → Output (on HDFS)
Input (on HDFS) → M3 → R2 → Output (on HDFS)
Input (on HDFS) → M4 → R2 → Output (on HDFS)
Speculative Execution

- Schedule a **backup** task if the original task takes too long to complete
  - Same input(s), different output(s)
- Failed tasks and stragglers get the same treatment
- Let the fastest win
  - After one task completes, kill all the clones
- Challenge: how can we tell a task is late?
Illustration: Speculative Execution

Source: Google Map-Reduce paper, OSDI 2004
A Critique of Map-Reduce

- “Too primitive”
  - Even the simplest tasks require programming
  - Data is non-structured – hard to manage

- “Overlooks many performance optimizations known for long in the database world”
  - E.g., vertical data partitioning

- “Good only for batch processing”
  - E.g., search index can be re-computed, not incrementally updated
Summary

- Large-scale batch processing of read-only data
- M and R have different nature
  - Optimized separately
- Not a silver bullet
Next Class

- Structured Databases atop Map-Reduce
Further Reading

- Google Map-Reduce paper