MapReduce and Hadoop

Aaron Birkland
Cornell Center for Advanced Computing

January 2012
Motivation

• Simple programming model for Big Data
  – Distributed, parallel – but hides this
• Established success at petabyte scale
  – Internet search indexes, analysis
  – Google, yahoo facebook
• Recently: 8000 nodes sort 10PB in 6.5 hours
• Open source frameworks with different goals
  – Hadoop, phoenix
• Lots of research in last 5 years
  – Adapt scientific computation algorithms to MapReduce, performance analysis
A programming model with some nice consequences

• Map(D) → list(Ki, Vi)
• Reduce(Ki, list(Vi)) → list(Vf)
• Map: “Apply a function to every member of dataset” to produce a list of key-value pairs
  – Dataset: set of values of uniform type D
    • Image blobs, lines of text, individual points, etc
  – Function: transforms each value into a list of zero or more key,value pairs of types Ki, Vi
• Reduce: Given a key and all associated values, do some processing to produce list of type Vf
• Execution over data is managed by a MapReduce framework
Canonical example: Word Count

- \( D = \text{lines of text} \)
- \( K_i = \text{Single Words} \)
- \( V_i = \text{Numbers} \)
- \( V_f = \text{Word/count pairs} \)
- \( \text{Map}(D) \rightarrow \text{list}(K_i, V_i) \)
- \( \text{Reduce}(K_i, \text{list}(V_i)) \rightarrow \text{list}(V_f) \)

\[
\text{Map}(D) \rightarrow \text{list}(K_i, V_i)
\]
\[
\text{Reduce}(K_i, \text{list}(V_i)) \rightarrow \text{list}(V_f)
\]

- Map(D) = Emit pairs containing each word and the number 1
- Reduce(Ki, list(Vi)) = Sum all the numbers in the list associated with the given word. Emit the word and the resulting count
Canonical example: Word Count

Map(D) → list(Ki, Vi)
Reduce(Ki, list(Vi)) → list(Vf)

Somehow need to group by keys so Reduce can be given all associated values!

1/19/2012
Opportunities for Parallelism?

absence of evidence

is not evidence of

absence

(absence, 1) -> (absence, 1)
(of, 1) -> (absence, 1)
(evidence, 1) -> (of, 1)
(is, 1) -> (of, 1)
(not, 1) -> (evidence, 1)
(of, 1) -> (evidence, 1)
(is, 1) -> (not, 1)

(absence, 2) -> (of, 2)
(evidence, 2) -> (is, 1)
(not, 1) -> (not, 1)

Promising

Worrisome

Promising
Opportunities for Parallelism

• Map and Reduce functions are independent
  – No explicit communication between them
  – Grouping phase between Map and Reduce is the only point of data exchange
• Individual Map, Reduce results depend only on input value.
  – Order of data, execution does not matter in the end.
• Input data read in parallel
• Output data written in parallel
Parallel, Distributed execution

absence of evidence
is not evidence of
(absence, 1)
(of, 1)
(evidence, 1)

(absence, 1)
(of, 1)
(evidence, 1)

(is, 1)
(not, 1)
(evidence, 1)
(of, 1)

(absence, 1)
(of, 1)
(evidence, 1)

(is, 1)
(not, 1)
(evidence, 1)
(of, 1)

(absence, 1)
(of, 1)
(evidence, 1)

(is, 1)
(not, 1)
(evidence, 1)
(of, 1)

(absence, 2)
(of, 1)

(is, 1)

(evidence, 1)

(absence, 2)
(of, 1)

(is, 1)

(evidence, 2)

(absence, 2)
(of, 1)

(is, 1)

(evidence, 2)
Full Parallel Pipeline
Full Parallel Pipeline

Split – Divide data into parallel streams
• Use features of underlying storage technology
  • File sharding, locality information, parallel data formats
Full Parallel Pipeline

Read – Chop data into iterable units

- Most common in MapReduce world – Lines of Text
- Can be arbitrary simple or complex – integer arrays, pdf documents, mesh fragments, etc.
Full Parallel Pipeline

Map – Apply a function, return a list of keys/values
Full Parallel Pipeline

Combine – (optional) execute a “mini-reduce” on some set of map output
  • For optimization purposes
  • May not be possible for every algorithm
Group – Group all results by key, collapse into a list of values for each key
- Need **all** intermediate values before this can complete
- Automatically performed by MapReduce framework
Full Parallel Pipeline

Partition – Send grouped data to reduce processes
- Typically, just a dumb hash to evenly distribute
- Opportunities for balancing or other optimization.
Full Parallel Pipeline

Reduce – Run a computation over each aggregated result, produce a final list of values
Full Parallel Pipeline

Write – Move Reduce results to their final destination
• Could be storage, or another MapReduce process!
Programming considerations

You *must* provide:
• Map, Reduce functions

You *may* provide:
• Combine, if it helps
• Partition function, if it matters

Framework must provide:
• Grouping and data shuffling

Framework may provide:
• Read, Write
  – For simple data such as lines of text
• Split
  – For parallel storage or data formats it knows about
Benefits

• Presents an easy-to-use programming model
  – No synchronization, communication by individual components. Ugly details hidden by framework.

• Execution managed by a framework
  – Failure recovery (Maps/Reduces can always be re-run if necessary)
  – Speculative execution (Several processes operate on same data, whoever finishes first wins)
  – Load balancing

• Adapt and optimize for different storage paradigms
Drawbacks

• Grouping/partitioning is serial!
  – Need to wait for all map tasks to complete before any reduce tasks can be run
• Some algorithms may be hard to conceptualize in MapReduce.
• Some algorithms may be inefficient to express in terms of Map Reduce
Hadoop

- Open Source MapReduce framework in Java
  - Spinoff from Nuch web crawler project
- HDFS – Hadoop Distributed Filesystem
  - Distributed, fault-tolerant, sharding
- Many sub-projects
  - Pig: Data-flow and execution language. Scripting for MapReduce
  - Hive: SQL-like language for analyzing data
  - Mahout: Machine learning and data mining libraries
    - K-means clustering, Singular Value Decomposition, Bayesian classification
Hadoop

• User provides java classes for Map, Reduce functions
  – Can subclass or implement virtually every aspect of MapReduce pipeline or scheduling
• Streaming mode to STDIN, STDOUT of external map, reduce processes (can be implemented in any language)
  – Lots of scientific data that goes beyond lines of text
  – Lots of existing/legacy code that can be adapted/wrapped into a Map or Reduce stage.

```
stream -input /dataDir/dataFile
-file myMapper.sh -mapper "myMapper.sh"
-file myReducer.sh -reducer "myReducer.sh"
-output /dataDir/myResults
```
HDFS

• Data distributed among compute nodes
  – Sharding: 64MB chunks
  – Redundancy
• Small number of large files
• Not quite POSIX file semantics
  – No random write, append
• Write-once read many
• Favor throughput over latency
• Streaming/sequential access to files
HDFS

File metadata

Sharding

NameNode

DataNode

Replication

1/19/2012
HDFS + MapReduce
HDFS + MapReduce

• Assume failure-prone nodes
  – Data and computation recovery through redundancy
• Move computation to data
  – Data is local to computation, direct-attached storage to each node
• Sequential reads on large blocks
• Minimal contention
  – Simultaneous maps/reduces on a node can be controlled by configuration
Hadoop + HDFS vs HPC
Hadoop in HPC environments

• Access to local storage can be problematic
  – Local storage may not be available at all
  – Even if so, long-term HDFS usually not possible

• HPC relies on global storage (e.g. Lustre) via high-speed interconnect.
  – What is meaning of “locality” in inherently non-local (but parallel) storage?
Hadoop @ TACC

• On *Longhorn* visualization cluster
• Special, local, persistent /hadoop filesystem on some machines
  – 48 nodes with 2TB HDFS storage/node
  – 16 nodes with 1TB HDFS storage/node, extra large memory (144GB memory)
• Modified hadoop distribution
  – Starts HDFS on allocated nodes
• Special Hadoop queue
• By request only
• Details at [https://sites.google.com/site/tacchadoop](https://sites.google.com/site/tacchadoop)
Dark Matter Halo Detection on Longhorn

- 200 TB simulated astronomy data
- Explore algorithms for identifying halo candidates by analyzing star density
- Compared compute parallel vs HDFS+MapReduce Data Parallel
  - ~57k data points/node/hr for compute parallel
  - ~600k data points/node/hr data parallel with MapReduce + HDFS
BLAST at NERSC

• Streaming API
• Issues with data format.
  – FASTA: sequence represented on multiple lines
    >some sort of header
    ACTGCATCATCATCATCAT
    GGGCTTACATCATCATCAT
  – Lots of effort re-implementing basic data handling components (Reader, possibly Split)
  – Eventually re-formatted data so each sequence was on own line
• Overall performance: Not significantly better than existing parallel methods
Still much to learn

- Most established patterns are from web and text processing (inverted indexes, ranking, clustering, etc)
- Scientific data and algorithms much more varied
  - Papers describing an existing problem applied to MapReduce are common
- When does HDFS provide benefit over traditional global shared FS?
  - Tends to do poorly for small tasks, can be a crossover point that needs to be found
- Lots of tuning parameters
  - Data skew and heterogeneity may lead to long, inefficient jobs.
Why Hadoop?

• If you find the programming model simple/easy
• If you have a data intensive workload
• If you need fault tolerance
• If you have dedicated nodes available
• If you like Java
• If you want to experiment.