CSCI 339
Distributed Systems

Lecture 15
Project 3 and MapReduce
Apr 6, 2023
Administrative Details

- MapReduce paper summary due now
- Project 3 is out
  - Git repos made...let me know if I made mistakes
- Bookstore
  - Sign up for demo slot with me or Jae
- Midterm has been graded – see Glow
  - Come talk to me if you have questions or concerns!
- Web server grades coming back very soon
Last Class

• Discussed coordination, agreement, and consensus
  • Briefly discussed different algorithms
  • Byzantine general’s problem (I posted a paper)

• Further readings on consensus (final project ideas)
  • Paxos
  • RAFT
Today’s Outline

• Begin talking about MapReduce, Hadoop, and Project 3
• Cluster configuration
Project 3

• IMPORTANT
  • At least one member of your group needs to sign up for Microsoft Azure.
  • You get $200 credit to start. Make sure you kill your VMs at the end of the project.
  • Might need a credit card…let me know if this is problematic.
• Project 3 (code) is due Apr 17th
  • Very little coding required!
    • …difficulty is in understanding Hadoop API and how to use it
    • But if Google developers can handle it, so can you! 😊
    • Pros and cons of abstraction (debugging Hadoop can be difficult)
  • Personal cluster management – important skill!
  • Short writeup required this time – mostly just explain how your mapper(s) and reducer(s) works and reflect, brief evaluation
    • Think about: job completion time, number of worker nodes, number of map/reduce tasks, size of input files, etc
    • Be creative! Learning to evaluate your system is very important.
Project 3 Highlights

• Calculate click-through rate for ads
• Azure VMs
  • https://portal.azure.com/#home
• Hadoop (version 2.10.2)
  • Open source implementation of MapReduce
    • http://hadoop.apache.org/
• High level goals:
  • Introduces you to several “state of the art” technologies that are widely used
• There is a Hadoop learning curve…don’t get frustrated! Watch tutorials and start simple.
Hadoop 2.9.2 and Yarn

Lots of abstractions!!! (And they keep adding more...)
Hadoop “Default” Data Flow

1. HDFS Input
   - Mapper
   - Local Output
   - HTTP Input
   - Reduce
   - HDFS Output

- InputFormat splits and reads files
- SequenceFileOutputFormat writes serialized values
- Map outputs are retrieved over HTTP and merged
- OutputFormat writes a SequenceFile or text
Helpful Tips

• Start with cluster setup tutorial
• Try “trivial” example
• Do most development on lab machines (without fighting with HDFS)
• Once you have things working, move to cluster
• Do not rely on VMs for ”stable storage”
• Stop VMs when not in use
• Delete VMs when completely finished
Moving on…
MapReduce

By:
Jeffrey Dean and Sanjay Ghemawat
Google, Inc.
Motivation: Large Scale Data Processing

- Want to process lots of data ( > 3200 TB)
  - This was invented by Google…
- Want to parallelize across hundreds/thousands of CPUs
  - Way more in reality…
- And want to make it easy
MapReduce

- Automatic parallelization & distribution
- Fault-tolerant
- Provides status and monitoring tools
- Clean abstraction for programmers
Sample Applications

- Distributed grep
- Count of URL access frequency
- Reverse Web-link graph
- Inverted index
- Distributed sort
Programming Model

• Borrows from functional programming
• Users implement interface of two functions:
  • map  \((\text{in\_key}, \text{in\_value}) \rightarrow (\text{out\_key}, \text{intermediate\_value})\text{ list}\)
  • reduce \((\text{out\_key}, \text{intermediate\_value list}) \rightarrow \text{out\_value list}\)
map

- Records from the data source (lines out of files, rows of a database, etc) are fed into the map function as key*value pairs
  - e.g., (document name, document contents).
- map() produces one or more intermediate values along with an output key from the input
  - e.g., For each word in document contents, emit (word, “1”)


reduce

- After the map phase is over, all the intermediate values for a given output key are combined together into a list
  - e.g., combine all (word, “1”) entries for same word
- reduce() combines those intermediate values into one or more final values for that same output key
  - e.g., (word, “1003”)
- (in practice, usually only one final value per key)
Input key*value pairs

Data store 1

map

(key 1, values ...)

(key 2, values ...)

(key 3, values ...)

... Barrier ... : Aggregates intermediate values by output key

(key 1, intermediate values)

(key 2, intermediate values)

(key 3, intermediate values)

reduce

final key 1 values

reduce

final key 2 values

reduce

final key 3 values

Data store n
Parallelism

• map() functions run in parallel, creating different intermediate values from different input data sets
• reduce() functions also run in parallel, each working on a different output key
• All values are processed *independently*
• Bottleneck: reduce phase can’t start until map phase is completely finished.
Example: Count word occurrences

map(String input_key, String input_value):
    // input_key: document name
    // input_value: document contents
    for each word w in input_value:
        EmitIntermediate(w, "1");

reduce(String output_key, Iterator intermediate_values):
    // output_key: a word
    // output_values: a list of counts
    int result = 0;
    for each v in intermediate_values:
        result += ParseInt(v);
    Emit(AsString(result));
Locality

- Master program divvies up tasks based on location of data: tries to have map() tasks on same machine as physical file data, or at least same rack
- map() task inputs are divided into 64 MB blocks: same size as Google File System chunks
Fault Tolerance

- Master detects worker failures
  - Re-executes completed & in-progress map() tasks
  - Re-executes in-progress reduce() tasks
- Master notices particular input key/values cause crashes in map(), and skips those values on re-execution.
  - Effect: Can work around bugs in third-party libraries!
Optimizations

• No reduce can start until map is complete:
  • A single slow disk controller can rate-limit the whole process
• Master redundantly executes “slow-moving” map tasks; uses results of first copy to finish
  • This is the “stragglers” problem

Why is it safe to redundantly execute map tasks? Wouldn’t this mess up the total computation?
Optimizations

- “Combiner” functions can run on same machine as a mapper
- Causes a mini-reduce phase to occur before the real reduce phase, to save bandwidth
MapReduce Conclusions

- MapReduce has proven to be a useful abstraction
- Greatly simplifies large-scale computations at Google
- Functional programming paradigm can be applied to large-scale applications
- “Fun” to use: focus on problem, let library deal w/ messy details
Discussion

• What do you think about MapReduce?
  • Questions/comments? Let me know!
    • PL/database communities were not impressed…

• Any problems or limitations?
  • Only applies to a specific type of computation!
  • https://kyligence.io/blog/is-hadoop-dead-the-future-of-big-data-analysis/

• Hadoop “spinoffs”
  • Bashreduce: https://rcrowley.org/2009/06/27/bashreduce.html
  • Cloudera: https://www.cloudera.com/products/open-source/apache-hadoop.html
  • Teradata (formerly Asterdata):
  • Database community thoughts:
    http://developers.slashdot.org/article.pl?sid=08/01/18/1813248