MapReduce

Tom Anderson
Last Time

- Difference between local state and knowledge about other node’s local state
- Failures are endemic
- Communication costs matter
Why Is DS So Hard?

• System design
  – Partitioning of responsibilities: what should client do, what should server do? Which servers should do what?

• Failures are endemic, partial and ambiguous
  – If the server doesn’t reply, how do you tell if it is (a) the network, (b) the server, or c) neither: they are both just being slow?

• Concurrency and consistency
  – Distributed state, replicated state, caching
  – How do we keep this state consistent?
Why Is DS So Hard?

• Performance
  – Generating a single FB or Google page involves calls to hundreds of different machines
  – Performance can be variable and unpredictable
  – Tail latency: only as fast as the slowest machine

• Implementation and testing
  – Nearly impossible to test/reproduce all failure cases

• Security
  – Adversary can silently compromise machines and manipulate messages
MapReduce

A programming model to help unsophisticated programmers use a data center without thinking about failures and distribution.

– Popular distributed programming framework
– Many related frameworks

Lab 1:

– Help you get up to speed on Go and distributed programming
– Exposure to some fault tolerance
– Motivation for better fault tolerance in later labs
MapReduce Computational Model

For each key k with value v, compute a new set of key-value pairs:

map \((k,v) \rightarrow \text{list}(k',v')\)

For each key \(k'\) and list of values \(v'\), compute a new (hopefully smaller) list of values:

reduce \((k',\text{list}(v')) \rightarrow \text{list}(v'')\)

User writes map and reduce functions. Framework takes care of parallelism, distribution, and fault tolerance.
MapReduce Steps

1. Split document into set of <k1, v1> pairs
2. Run Map(k1, v1) on each element of each split to produce a set of <k2, v2> pairs
3. Coalesce results from each mapper into a (sorted) list for each key
4. Run Reduce(k2, list(v2)) -> list(v2)
   - Optionally run reduce function on results for each key produced by each mapper, to reduce network bw
5. Merge result
MapReduce In Action
Example: grep
find lines that match text pattern

1. Master splits file into M almost equal chunks at line boundaries
2. Master hands each partition to mapper
3. map phase: for each partition, call map on each line of text
   – search line for word
   – output line number, line of text if word shows up, nil if not
4. Partition results among R reducers
   – map writes each output record into a file, hashed on key
Example: grep

5. Reduce phase: each reduce job collects 1/R output from each Map job
   – all map jobs have completed!
   – Reduce function is identity: v1 in, v1 out

6. merge phase: master merges R outputs
Another Example: PageRank

• Compute “importance” of web pages
  – Search result ordering
  – Pages are important if linked by important pages

• Initially: assign every page a default value

• Map:
  – For every page k with outlink l, emit <l, value>

• Reduce:
  – For each target page l, output new value as average of inlink values

• Repeat MapReduce until done
Questions

Suppose we run MapReduce across N workers, with M map partitions and R reducers

• Example: 200K M; 5K R; 2K N

• Number of tasks?

• Number of intermediate files?
Lab 1 Hint

Lab 1 provides code to do all the steps of MapReduce, but on a single node: RunSingle
Questions

How much speedup do we expect on N servers?

What are the bottlenecks to performance?
Question

Why the roughly fixed (64MB) size for the initial size of the input split files?
Question

• How does the master tell a specific worker to do a specific (map, reduce) task?
Remote Procedure Call (RPC)

A request from the client to execute a function on the server.
RPC Implementation
For MapReduce master and worker, who’s the client? who’s the server?
Is an RPC like a normal function call?

Binding
- Client needs a connection to server
- Server must implement the required function

Performance
- Local call: maybe 10 cycles = \( \sim 3 \) ns
- in data center: 10 microseconds \( \Rightarrow \sim 1K \) slower
- in the wide area: millions of times slower

Failures
- What happens if messages get dropped?
- What if client crashes?
- What if server crashes?
- What if server appears to crash but is slow?
- What if network partitions?
MapReduce Fault Tolerance Model

Master is not fault tolerant
  – Assumption: this single machine won't fail during running a mapreduce app

Many workers, so have to handle their failures
  – Assumption: workers are fail stop
  – They can fail and stop
  – They may reboot
  – They don't send garbled weird packets after a failure
What kinds of faults does MapReduce need to tolerate?

• Network:

• Worker:
Tools for Dealing With Faults

• Retry
  – if pkt is lost: resend
  – worker crash: give task to another worker
  – may execute MR job twice! (is this ok? Why?)

• Replicate
  – E.g., input files

• Replace
  – E.g., new worker can be added
Lab 1 Simplifications

• No key in map
• Assume global file system
• No partial failures
  – Files either completely written or not created
  – If restart some failed operation, ok to write to the same filename
DeWitt/Stonebraker Critique

• A giant step backward in the programming paradigm for large-scale data intensive applications
• A sub-optimal implementation, in that it uses brute force instead of indexing
• Not novel at all: represents a specific implementation of well known techniques developed nearly 25 (now 35) years ago
• Missing most of the features that are routinely included in current DBMS
• Incompatible with all of the tools DBMS users have come to depend on"