Map-Reduce

Mark Greenstreet

CpSc 418 – April 6, 2016

Portrait of a Data Centre

- Sketch of a big data center:
  - Thousands (or tens of thousands) of machines, each with its own disk.
  - File system is distributed across these machines.
    - ★ The file system is redundant: machines **will fail**
  - Use commodity (e.g. Cisco) networks and routers.
    - ★ Each machine has a mainstream network interface (e.g. 10Gb ethernet)
    - ★ Cross-section bandwidth is **way** smaller than the number of machines times the per-machine bandwidth.
- Need to analyse the huge data sets stored on these machines.

Problem Domain: Large-Scale Data Analysis

- Data analysis requires:
  - Fetching the relevant records.
  - Performing analysis of related records.
  - Summarizing the results.
- Example: word frequency in documents
- Example: core curriculum
  - How do 200-level courses impact success in 400-level courses?
  - Look at all transcripts.
  - Analyze relationships for *(2XX, 4YY)* pairs.
- Problem statement: Google noted that each such problem was getting its own custom code:
  - All of that code development is expensive.
  - Code can fail when the system is changed.

The Map-Reduce Pattern

- All data is represented as lists of *(Key, Value)* pairs.
- **map**
  - For each *(Key1, Value1)* pair of the input,
  - Produce a list of *(Key2, Value2)* pairs for the output.
- **reduce**
  - All *(Key2, Value2)* pairs with the same *Key2* are combined using the reduce function.
  - This produces a *(Key2, [colorCodeColorlist of Value2])* result.

Map-Reduce, Curriculum

- The *(Key1, Value1)* pairs:
  - *Key* could be the student number for the transcript
  - *Value* is a list of *(CourseNumber, Grade)* pairs.
- map: for each 400-level course on the transcript, associate it with the grade for that course and a list of all the 200-level courses taken by that student and the grades for those courses.
  - *Key2* is a 400-level course number
  - *Value2* is *(Grade400, [(Course200, Grade200), . . .])*, where
    - *Grade400* is the grade in the 400-level course, *Key2*.
    - *Course200* is the course-number for a 200-level course on the same transcript.
    - *Grade200* is the grade for that course on the same transcript.
    - with one entry in the list for each 200-level course.
- reduce: just returns *(Grade400, [(Course200, Grade200), . . .])*.
- Then, we do another map to compute a linear best-fit for the data.
  - From there, we could do more operations to further analyze the data.

Map-Reduce, Word-Count

From the paper:

- The *(Key1, Value1)* are the document name and document text.
- map: produce a list of *(Word, Count)* pairs, where *Word* is each word that appears in the document, and *Count* is the number of times that it occurs.
- reduce: for each *Word*, add up the *Count* values from all documents.

Programming Model

The user creates a **mapreduce specification object** (C++)

- provide the *map* and *reduce* functions (presumably as methods of the object).
- fill-in fields with the names of the input and output files.
- set other tuning parameters [optional].
- Invoke the `MapReduce` function.

Execution

- The `MapReduce` function spawns *M* map tasks, *R* reduce tasks, and one master.
- Each map task:
  - is assigned a fragment of the input file by the master – these fragments are called **splits**;
  - reads the records from that file;
  - performs the `map` operation;
  - writes the results to a temporary files;
  - informs the master of its progress.
- Each reduce task:
  - is notified of map results by the master;
  - requests the data from the map tasks;
  - performs the reduce operation;
  - writes the result to a file;
  - notifies the master when it is done.
- When all the reduce computations are complete, the master sends a message to wake up the user process, and the `MapReduce` function returns.

Hashing

How do the intermediate results get from the map process to the reduce process?

- When the map-task is spawned it is told how many reduce processes there are.
- Each *(Key2, Value2)* is written to a different file according to *hash(Key2) mod R*.
- The master tells the reduce tasks which file to read from each map task.
- The user can provide their own **hash** function if the default one isn't optimal.
  - If the default hash doesn't balance the workload.
  - There is a reason to cluster certain `Key2` tuples on the same task.

Bad Things Happen

- Machines fail, especially if you have thousands of them.
  - If the average lifetime of a machine is five years,
  - then a data center with 10,000 machines has a machine fail every four hours!
- Routers and other network infrastructure can fail, leaving machines isolated.
- Maintenance can take machines off-line.

Fault Tolerance

- Key Idea:
  - The `map` and `reduce` operations are based on functional programming ideas – they don't have side-effects.
  - If a worker crashes, it's as if it never existed.
  - The master can restart the task on another machine.
- The master periodically pings the tasks, and restarts dead ones.
  - If a completed map task fails, it is reexecuted because a reduce task may need the results.
  - If a completed reduce task fails, no action is needed – the results has been stored in the global (redundant, fault-tolerant) file system.

Semantics

- If the *map* and *reduce* functions are deterministic, then the result of `MapReduce` is the same as a sequential execution.
  - **This is really cool!**
  - There **is** a sequential implementation of `MapReduce`:
    - ★ Read all of the *(Key1, Value1)* pairs from the input file.
    - ★ Write all of the *(Key2, [list of Value2])* tuples to an intermediate file.
    - ★ Sort the intermediate file by the *Key2* values.
    - ★ Perform the reduce operation for each *Key2* value and write the results to the output file.
- If the *map* and *reduce* functions are not-deterministic, then
  - It's a bit more complicated, but it's still reasonable.
  - If the reduce tasks are non-deterministic, then the result for each *Key2* is the result from some sequential implementation.
  - The paper doesn't talk about non-determinism for *map*, but I expect it would be similar.

Work Stealing

- Sometimes a worker is slow – **stragglers**.
- If the `MapReduce` task is near completion, the master assigns straggler tasks to idle processors.
- These are called **backup tasks**.
- Either the original or the backup process can complete the task.
- In practice, this **work stealing** by backup tasks:
  - Only adds a few percent to the total compute resources used.
  - Can result in substantial performance improvements:
    - The paper reported a 44% slow-down when the sort benchmark was run without backup tasks.

Performance Issues

- The master attempts to schedule map tasks on processor that holds the split being processed, or are nearby (by the network connections).
- Reduce tasks must read intermediate results from many map tasks
  - In general, every reduce task could read from every map task.
  - This could easily saturate the cross-section bandwidth of the data center.
- For good performance, the map tasks should be filters that output much less data than they read.
  - This isn't always possible.
  - But it's good to find ways to represent the intermediate data as compactly as possible.
- Partial reduction can also lower the bandwidth needed by map reduce
  - Each map task does a reduce for each *Key2*.
  - Changes the semantics of Map Reduce – but no change if reduce is associative and commutative.

MapReduce targets big data:

- Reading large disk files takes seconds.
- Communication between standard linux machines with generic networks takes milliseconds.
- The task needs to be big enough to justify these overheads:
  - We've just increased λ by a few orders of magnitude.
  - MapReduce makes sense if the task is disk-limitted and harnassing a few thousand disks provides the necessary disk bandwidth.
    - ★ Think of it as "disk parallelism" instead of "CPU parallelism".
    - ★ Note: big-data companies like Amazon, Facebook and Google are moving to using FLASH memory and DRAM instead of disks, exactly because of these I/O bottlenecks.
  - Or, if you have a really huge data set, and the compute time dominates all of these overheads.

Results

- Good performance on $\sim$ 2000 machines: `grep` and `sort`.
- In widespread use at Google.
- Continuing to grow in use, especially through its open source implementation: Hadoop.

More Examples

Searching for 'map reduce' on `http://scholar.google.ca`:

- *Hive: a warehousing solution over a map-reduce framework*
- *Map-Reduce for Machine Learning on Multicore*
- *Twister: a runtime for iterative MapReduce*
- *Upper and lower bounds on the cost of a map-reduce computation*
- *MapReduce Algorithms for Big Data Analysis*

Summary

- MapReduce is a parallel programming pattern
  - Data are represented by lists of *(Key, Value)* pairs.
  - A user provided **map** function takes each *(Key1, Value1)* pair of the input, and produces a (possibly empty) list of *(Key2, Value2)* pairs.
  - A user provided **reduce** function is applied to the list of *Value2* values for each *Key2*.
- Details of the parallel implementation are handled by the MapReduce API:
  - Creating workers processes, sending messages between machines, etc.
  - Handling failures and slow nodes.
- Performance is often bandwidth limited
  - Locality matters: perform *map* on the machine with the data.
  - If *map* is an effective filter (bytes out $\ll$ bytes in), then we can reduce the impact of network congestion.

Review 1

- How does map-reduce distribute work between map tasks?
- How does map-reduce distribute work between reduce tasks?
- How does map-reduce handle machine or failures?
- How does map-reduce handle slow (i.e. straggler) machines?

Review 2

- What are the requirements for the type-signatures of the *map* and *reduce* functions a map-reduce?
  - Why did this force me to use an extra "map" step in my "Curriculum design" example? In other words, why couldn't *reduce* compute the linear regression?
- Let's say that I have a table of airline flights. Each entry is of the form

  ```
  (DepartCity, DepartTime, ArriveCity, ArriveTime)
  ```

  I want to fly from Vancouver to Timbuktu, but there are no direct flights. Let's say I want to find the fastest route with one stop. How could I do this using map reduce?
  - Hint: use the intermediate city as *Key2*.
  - For simplicity, assume that all times are GMT (no need for time-zone conversion).
  - How does *map* filter out irrelevant flights?

Supplementary Material: Percolation and HW4

Why?

- Monte-Carlo simulation is an important application of parallel computing.
  - Monte-Carlo simulation is named after the gambling center in Monaco.
  - The idea is to simulate a randomized model.
  - Statistical properties are computed from the results of a large number of simulations.
- Examples of Monte-Carlo simulation
  - Investment portfolio analysis.
  - Modeling the spread of epidemics.
  - Climate modeling.
  - Machine learning.
- Percolation is an example of a very simple, random process, that we can study by Monte Carlo simulation

What is Percolation?

- The original problem arose from modeling the seepage of fluids through rock.
  - Rocks have microscopic pores.
    - ★ For some rocks, water, oil, or gasses can slowly diffuse through them.
    - ★ Other rocks are impervious to such flow, even though they have tiny pores.
    - ★ Why?
  - It has to do with the density and structure of the pores.
  - Percolation matters for oil and gas exploration, building construction, and many other areas.
- We can formulate the percolation question as a question about graphs.
  - Then look for simple examples to understand the key ideas.
- Percolation by Grimmett is an excellent book,
  - but it is in no way needed for this course!

The percolation model for HW4

- Given positive integers, W and H (for the width and height), and $0 \leq p \leq 1$ (the marking probability), we'll construct a directed graph, $G = \{V, E\}$ where
  - $V = \{v_{i,j} | 0 \leq i < h, 0 \leq j < w$ is the set of vertices.
  - $E = \{(v_{i,j}, vi + 1, j), (v_{i,j}, v_{i+1}, (j + 1) \mod w) | 0 \leq i < h, 0 \leq j < w\}$ is the set of edges.
  - Every vertex, $v_{0,j}$ is "marked".
  - All other vertices are marked with probability p.
- What is the probability that there is a path from a vertex in row 0 to a vertex in row $h - 1$ for which every vertex on the path is marked?
- I'll draw a picture on the board and add it to the final version of the slides.

Simulating Percolation

- Create an array of W vertices, and set them all to 1.

```
for i = 1 to H-1 {
   for j = 0 to W-1 {
      vv[i,j] = (v[i-1,j] | v[i-1, (j-1) mod w]) & bernoul
      v = vv;
   }
}
```

- Where `bernoulli(p)` is a random variable that is 1 with probability `p` and 0 with probabily 1-p.
- Each call to `bernoulli` produces an independent random variable with this distribution.
- There is a path from a vertex in row 0 to a vertex in row $H-1$ iff there is some `j` at the end of the simulation such that `v[j]` is 1.

More stuff

- The critical probability.
- Using GPUs to make the code run fast.
  - The random-number generation bottleneck.
    - ★ Use separate kernels for random-number generation and network simulation.
  - The memory bandwidth bottleneck.
    - ★ Pack random bits into integers.
  - The vertex update bottleneck.
    - ★ Use bitwise logical operators.
  - Now, what are the performance issues?