

GOOGLE'S MAPREDUCE

Happyjack Tech Talk
February 20, 2008

MAP? REDUCE?

- In 2004, Google announced a “new” programming paradigm called MapReduce
- This programming model and implementation makes it easy to process and generate large data sets
- It has become part of Google’s “Cloud Computing” vision

LISP ROOTS

- Map & Reduce are traditional techniques in functional programming
- Map - performs some function on each element of a list
- Reduce - combines all the elements of a list according to a binary function

MAP

```
(define (square n)  
  (* n n))
```

```
(map square '(1 2 3 4 5)) => '(1 4 9 16 25)
```

REDUCE

```
(define (plus a b)  
  (+ a b))
```

```
(foldl plus 0 '(1 2 3 4 5)) => 15  
(5+...(2+(1+0)))
```

MORE REDUCE

```
(define (even? n)  
  (= (remainder n 2) 0))
```

```
(filter even? '(1 2 3 4 5)) => '(2 4)
```

FUNCTIONAL OPERATORS

- In general, much of the power of functional programming comes from using functions (i.e., function variables) to process lists in predetermined ways

LISP WITHOUT (...)

```
def even(x): return x % 2 == 0
```

```
filter(even, range(1, 5)) => [2, 4]
```

```
def square(x): return x * x
```

```
map(square, range(1,5)) => [1, 4, 9, 16]
```

```
def add(x,y): return x + y
```

```
reduce(add, range(1,5)) => 10
```

```
[square(x) for x in range(1,5) if even(x)] => [4, 16]
```

KEY OBSERVATIONS

- So map & reduce are hardly new
- But Google made two key observations:
 - many practical problems can be expressed as map/reduce problems
 - map/reduce can be generalized to process hashtables as well as lists

MAPREDUCE ON HASHES

- In the context of a hash, map / reduce can work as follows:
 - Map processes each entry in the hash and generates one or more (intermediate) key / value pairs
 - Reduce combines all of the values for each key, generating one or more (final) key / value pairs

EXAMPLE 1

- Suppose we want to count the number of occurrences of each word in a bunch of documents
- Map processes many pairs of the form <filename, contents>
- It generates many pairs of the form <word, 1>
 - (Or <word, count> if you prefer...)

EXAMPLE 1, CONT'D

- Reduce processes all of the <word, count> pairs generated by the Map process for each word
 - Then it generates a single <word, total> for that word

EXAMPLE 1, CONT'D

```
void map (string url, string contents) {  
    foreach (word w in contents) {  
        EmitIntermediate (word, 1);  
    }  
}
```

EXAMPLE 1, CONT'D

```
void reduce (string word, list values) {  
    int total = 0;  
    foreach (v in values) {  
        total += (int) v;  
    }  
    Emit (word, total);  
}
```

WHY THE FUSS?

- Why not just write the following:

```
int count (docs) {  
    foreach (d in docs) {  
        foreach (word w in d.contents) {  
            total[w] ++;  
        }  
    }  
}
```

HIDDEN COMPLICATIONS

- The previous code works just fine when
 - we can load all the documents into memory on a single machine
 - a single machine can be expected to process all the documents by itself
- Neither of these assumptions holds in Google's typical use case!

SOLUTION: PARALLELISM

- Google's solution is to use massive parallelism
- Writing parallel code is very complicated, and programmers often get it wrong
- Google uses MapReduce to provide a simple interface to a very complex, distributed infrastructure
- Massive parallelism without pain!

PARALLELISM = HARD

- Easy to parallelize the outer loop -- but what about locking, sharing total array, recovery....

```
foreach (d in docs) in parallel {  
    foreach (word w in d.contents) {  
        total[w] ++;  
    }  
}
```

OTHER EXAMPLES

- We'll get to the distributed implementation of MapReduce in a moment
- But first, let's look at some other examples of MapReduce in action

GREP

- Map: emit a `<file, line>` record whenever the line matches the pattern
- Reduce: do nothing! Pass through the intermediate `<file, line>` records

WEB LINK GRAPH

- Map: emit record $\langle \text{source}, \text{target} \rangle$ for each link found in the source (going to target)
- Reduce: collect all the target links from a given source into a single $\langle \text{source}, \text{target_list} \rangle$ record

REVERSE LINK GRAPH

- Map: emit record $\langle \text{target}, \text{source} \rangle$ for each link found in the source (going to target)
- Reduce: collect all the sources to a given target into a single $\langle \text{target}, \text{source_list} \rangle$ record
 - Note: This is the same reduce as before...

WORD INDEX

- Map: emit `<word, url>` for each word in a given web document
- Reduce: combine all the `<word, url>` pairs associated with a given word into a single `<word, url_list>`. We can also sort the urls in the list, e.g., by word frequency

IMPLEMENTATION

- Runs on large clusters of commodity PCs
- Machines are typically dual-processor boxes with 2-4 Gigs of memory (and running Linux)
- Machines are connected with Gigabit ethernet, organized into racks of 40-80 machines
- Storage is local to the machines, with lots of replication for performance and reliability

MAPREDUCE

- Initially, the system selects M and R , the number of tasks to run map and reduce, respectively
- Then the input file(s) are split into M pieces, one for each map task
- One master task is in charge of submitting the M map and R reduce jobs to the server farm

THE MAP WORKERS

- A map worker processes its portion of the input file and hands it to the user-defined map function
- As the user-defined map function generates `<key,value>` pairs, they are buffered in memory
- Periodically, the pairs are written out to disk into `R` intermediate datafiles (chosen by `hash(key)%R`)

THE MAP WORKERS

- When a map worker is finished, it will have produced R output files
- Then it notifies the master task that it is finished
- Note: the output files are kept in the worker machine
- If there are any more map tasks, the worker will be assigned another such task by the master

THE REDUCE WORKERS

- A reduce worker is notified when an intermediate data file is ready
- It requests the file from the map worker, and the file is transferred using a special protocol
- When all M intermediate files for a given reduce task are finished, the reduce task combines them into a $\langle \text{key}, \text{value_list} \rangle$ structure (by sorting on the keys first)

THE REDUCE WORKERS

- Once the $\langle \text{key}, \text{value_list} \rangle$ pairs are generated, the user-defined reduce function is called on each
- The workers append the output generated by the user-defined reduce function into a result file
- When the worker is finished, it notifies the master (which may give it another reduce task)

LOAD BALANCING

- Initially, the input file is split into M pieces
- Of course, the split is even, so all of the map tasks are of roughly the same size
- If some machines are faster than others, they will finish earlier, so they will be assigned more map tasks than the slower machines!

FAULT TOLERANCE

- The master process keeps track of which workers performed which map and / or reduce operations
- If any of the machines goes down (i.e., if it fails successive pings), its work is reassigned
- It is almost certain that some machines will fail, but there will be many more machines that can redo that machine's work

FAULT TOLERANCE

- The single point of failure is the master itself
- It keeps its state (which includes the list of assigned work) in a data structure which is periodically checkpointed to disk
- If it dies, the work can be restarted from the last checkpoint

LOCALITY

- The distributed file system (GFS) uses replication for performance and reliability
- MapReduce takes advantage of this by trying to schedule map tasks that are close (in terms of network topology) to their input file

GRACEFUL FAILURE

- Sometimes a machine will run an order of magnitude or two slower than the others
 - Typically, this is because of an incipient hardware failure
- When the MapReduce task is close to finished, such a machine can dominate the execution time

GRACEFUL FAILURE

- MapReduce handles this by special-casing the end of the computation
- Once it gets sufficiently close, it assigns the remaining tasks to more than one machine
- May the fastest one win....

ATTACKING MAPREDUCE

- MapReduce has attracted many enthusiasts
- Naturally, it is also starting to attract some detractors who accuse the “MapReduce community” of over promising

STONEBRAKER'S ATTACK

- MapReduce is a step backward in database access
- MapReduce is a poor implementation
- MapReduce is not new
- MapReduce is missing features
- MapReduce is incompatible with RDBMS tools

STONEBRAKER'S ATTACK

- In some ways, Stonebraker's attack is a category error:
 - MapReduce is not a distributed RDBMS
- On the other hand, you can interpret Stonebraker's words more favorably as follows:
 - If you use a distributed RDBMS instead of MapReduce, you can solve the same problems

STONEBRAKER'S ATTACK

- MapReduce is a step backward in database access
 - always full table scan
- MapReduce is a poor implementation
 - no indices
 - no optimization

STONEBRAKER'S ATTACK

- MapReduce is not new
 - Map & Reduce are functional ideas
 - Implementation has much in common with existing distributed processing techniques

STONEBRAKER'S ATTACK

- MapReduce is missing features
 - Ad-hoc and declarative queries
 - Stonebraker compares it to CODASYL
- MapReduce is incompatible with RDBMS tools
 - No schema manager
 - No backup utilities

STONEBRAKER'S ATTACK

- Stonebraker's point is that distributed RDBMSs were designed to solve just some of the same problems people solve with MapReduce
- They've already run into many, many performance problems
- MapReduce is ignoring that history (and the solutions that the database folks came up with)

FUTURE OF RDBMS?

- RDBMSs have evolved to offer a full range of important (but expensive) features:
 - distributed transactions
 - referential integrity
 - query optimization
- They solve MapReduce problems...and more

RDBMS = OVERKILL?

- For many applications, you need all those things, e.g., banking applications
- For many others, they're completely unnecessary, e.g., web indexing or log reporting
- Maybe we don't need RDBMSs for *all* problems