CS6030
Cloud Computing

Ajay Gupta
B239, CEAS
Computer Science Department
Western Michigan University

ajay.gupta@wmich.edu
276-3104

Acknowledgements

- I have liberally borrowed these slides and material from a number of sources including
  - Web
  - MIT, Harvard, UMD, UCSD, UW, Clarkson, . . .
  - Amazon, Google, IBM, Apache, ManjraSoft, CloudBook, . . .
- Thanks to original authors including Dyer, Lin, Dean, Buyya, Ghemawat, Fanelli, Bisciglia, Kimball, Michels-Slettvet,…
- If I have missed any, its purely unintentional. My sincere appreciation to those authors and their creative mind.

Today's Topics

- Class feedback from WMU portal – Course Evaluation
- Functional programming
- MapReduce
- Distributed file system

Functional Programming

- MapReduce = functional programming meets distributed processing on steroids
  - Not a new idea… dates back to the 50's (or even 30's)
- What is functional programming?
  - Computation as application of functions
  - Theoretical foundation provided by lambda calculus
- How is it different?
  - Traditional notions of "data" and "instructions" are not applicable
  - Data flows are implicit in program
  - Different orders of execution are possible
- Exemplified by LISP and ML

Overview of Lisp

- Lisp ≠ Lost In Silly Parentheses
- One may focus on a particular dialect: "Scheme"
- Lists are primitive data types

```
'(1 2 3 4 5)
'((a 1) (b 2) (c 3))
```

- Functions written in prefix notation

```
(+ 1 2) → 3
(* 3 4) → 12
(sqrt (+ (* 3 3) (* 4 4))) → 5
(define x 3) → x
(* x 5) → 15
```


Functions

- Functions = lambda expressions bound to variables

```
(define foo
  (lambda (x y)
    (sqrt (+ (* x x) (* y y)))))
```

- Syntactic sugar for defining functions
  - Above expressions is equivalent to:

```
(define (foo x y)
  (sqrt (+ (* x x) (* y y))))
```

- Once defined, function can be applied:

```
(foo 3 4) → 5
```

Other Features

- In Scheme, everything is an s-expression
  - No distinction between "data" and "code"
  - Easy to write self-modifying code
- Higher-order functions
  - Functions that take other functions as arguments

```
(define (bar f x) (f (f x)))
```

Doesn't matter what *f* is, just apply it twice.

```
(define (baz x) (* x x))
(bar baz 2) → 16
```


Recursion is your friend

- Simple factorial example

```
(define (factorial n)
  (if (= n 1)
      1
      (* n (factorial (- n 1)))))
(factorial 6) → 720
```

- Even iteration is written with recursive calls!

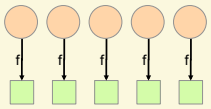
```
(define (factorial-iter n)
  (define (aux n top product)
    (if (= n top)
        (* n product)
        (aux (+ n 1) top (* n product))))
  (aux 1 n 1))
(factorial-iter 6) → 720
```


Lisp → MapReduce?

- What does this have to do with MapReduce?
- After all, Lisp is about processing *lists*
- Two important concepts in functional programming
  - Map: do something to everything in a list
  - Fold: combine results of a list in some way

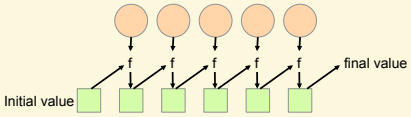
Map

- Map is a higher-order function
- How map works:
  - Function is applied to every element in a list
  - Result is a new list



Fold

- Fold is also a higher-order function
- How fold works:
  - Accumulator set to initial value
  - Function applied to list element and the accumulator
  - Result stored in the accumulator
  - Repeated for every item in the list
  - Result is the final value in the accumulator



Map/Fold in Action

- Simple map example:

```
(map (lambda (x) (* x x))
     '(1 2 3 4 5))
 → '(1 4 9 16 25)
```

- Fold examples:

```
(fold + 0 '(1 2 3 4 5)) → 15
(fold * 1 '(1 2 3 4 5)) → 120
```

- Sum of squares:

```
(define (sum-of-squares v)
  (fold + 0 (map (lambda (x) (* x x)) v)))
(sum-of-squares '(1 2 3 4 5)) → 55
```

Lisp → MapReduce

- Let's assume a long list of records: imagine if...
  - We can parallelize map operations
  - We have a mechanism for bringing map results back together in the fold operation
- That's MapReduce! (and Hadoop)
- Observations:
  - No limit to map parallelization since maps are independent
  - We can reorder folding if the fold function is commutative and associative

Typical Problem

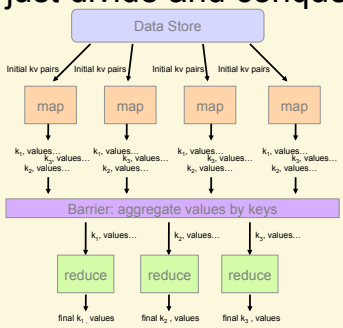
- Iterate over a large number of records
- Map: Extract something of interest from each
- Shuffle and sort intermediate results
- Aggregate intermediate results
- Reduce: Generate final output

Key idea: provide an abstraction at the point of these two operations

MapReduce

- Programmers specify two functions:
  map (k, v) → <k', v'>*
  reduce (k', v') → <k', v'>*
  - All v' with the same k' are reduced together
- Usually, programmers also specify:
  partition (k', number of partitions) → partition for k'
  - Often a simple hash of the key, e.g. hash(k') mod n
  - Allows reduce operations for different keys in parallel
- Implementations:
  - Google has a proprietary implementation in C++
  - Hadoop is an open source implementation in Java (lead by Yahoo)

It's just divide and conquer!



Recall these problems?

- How do we assign work units to workers?
- What if we have more work units than workers?
- What if workers need to share partial results?
- How do we aggregate partial results?
- How do we know all the workers have finished?
- What if workers die?

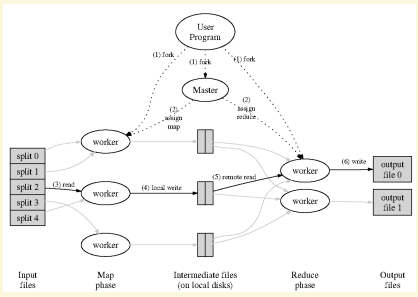
MapReduce Runtime

- Handles scheduling
  - Assigns workers to map and reduce tasks
- Handles "data distribution"
  - Moves the process to the data
- Handles synchronization
  - Gathers, sorts, and shuffles intermediate data
- Handles faults
  - Detects worker failures and restarts
- Everything happens on top of a distributed FS (later)

"Hello World": Word Count

```
Map(String input_key, String input_values):
    // input_key: document name
    // input_values: document contents
    for each word w in input_values:
        EmitIntermediate(w, "1");

Reduce(String key, Iterator intermediate_values):
    // key: a word, same for input and output
    // intermediate_values: a list of counts
    int result = 0;
    for each v in intermediate_values:
        result += ParseInt(v);
    Emit(key, AsString(result));
```

Source: Dean and Ghemawat (OSDI 2004)

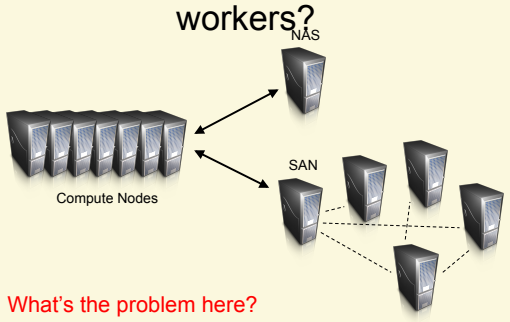
Bandwidth Optimization

- Issue: large number of key-value pairs
- Solution: use "Combiner" functions
  - Executed on same machine as mapper
  - Results in a "mini-reduce" right after the map phase
  - Reduces key-value pairs to save bandwidth

Skew Problem

- Issue: reduce is only as fast as the slowest map
- Solution: redundantly execute map operations, use results of first to finish
  - Addresses hardware problems...
  - But not issues related to inherent distribution of data

How do we get data to the workers?



What's the problem here?

Distributed File System

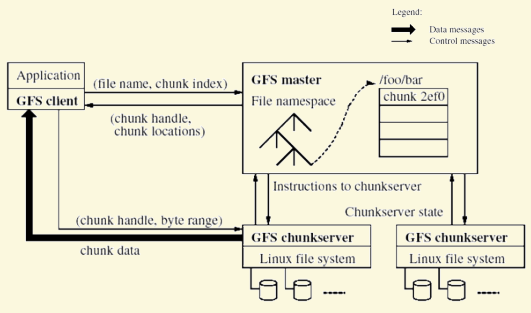
- Don't move data to workers… Move workers to the data!
  - Store data on the local disks for nodes in the cluster
  - Start up the workers on the node that has the data local
- Why?
  - Not enough RAM to hold all the data in memory
  - Disk access is slow, disk throughput is good
- A distributed file system is the answer
  - GFS (Google File System)
  - HDFS for Hadoop

GFS: Assumptions

- Commodity hardware over "exotic" hardware
- High component failure rates
  - Inexpensive commodity components fail all the time
- "Modest" number of HUGE files
- Files are write-once, mostly appended to
  - Perhaps concurrently
- Large streaming reads over random access
- High sustained throughput over low latency

GFS: Design Decisions

- Files stored as chunks
  - Fixed size (64MB)
- Reliability through replication
  - Each chunk replicated across 3+ chunkservers
- Single master to coordinate access, keep metadata
  - Simple centralized management
- No data caching
  - Little benefit due to large data sets, streaming reads
- Simplify the API
  - Push some of the issues onto the client



Source: Ghemawat et al. (SOSP 2003)

GFS Single Master

- We know this is a:
  - Single point of failure
  - Scalability bottleneck
- GFS solutions:
  - Shadow masters
  - Minimize master involvement
    - Never move data through it, use only for metadata (and cache metadata at clients)
    - Large chunk size
    - Master delegates authority to primary replicas in data mutations (chunk leases)
- Simple, and good enough!

GFS Master's Responsibilities (1/2)

- Metadata storage
- Namespace management/locking
- Periodic communication with chunkservers
  - Give instructions, collect state, track cluster health
- Chunk creation, re-replication, rebalancing
  - Balance space utilization and access speed
  - Spread replicas across racks to reduce correlated failures
  - Re-replicate data if redundancy falls below threshold
  - Rebalance data to smooth out storage and request load

GFS Master's Responsibilities (2/2)

- Garbage Collection
  - Simpler, more reliable than traditional file delete
  - Master logs the deletion, renames the file to a hidden name
  - Lazily garbage collects hidden files
- Stale replica deletion
  - Detect "stale" replicas using chunk version numbers

GFS : Metadata

- Global metadata is stored on the master
  - File and chunk namespaces
  - Mapping from files to chunks
  - Locations of each chunk's replicas
- All in memory (64 bytes / chunk)
  - Fast
  - Easily accessible
- Master has an operation log for persistent logging of critical metadata updates
  - Persistent on local disk
  - Replicated
  - Checkpoints for faster recovery

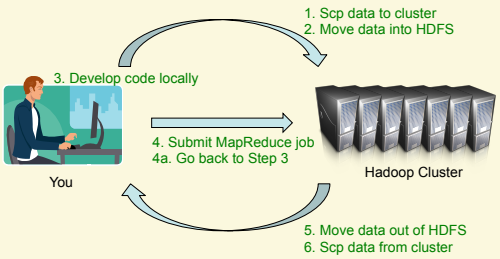
GFS: Mutations

- Mutation = write or append
  - Must be done for all replicas
- Goal: minimize master involvement
- Lease mechanism:
  - Master picks one replica as primary; gives it a "lease" for mutations
  - Primary defines a serial order of mutations
  - All replicas follow this order
  - Data flow decoupled from control flow

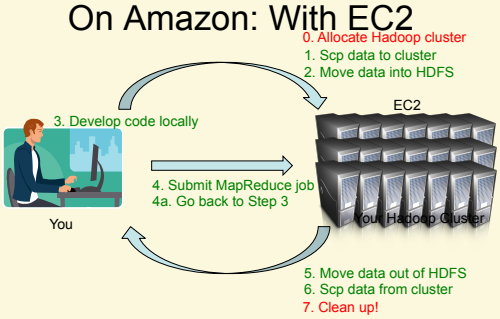
Parallelization Problems

- How do we assign work units to workers?
- What if we have more work units than workers?
- What if workers need to share partial results?
- How do we aggregate partial results?
- How do we know all the workers have finished? How is MapReduce different?
- What if workers die?

From Theory to Practice

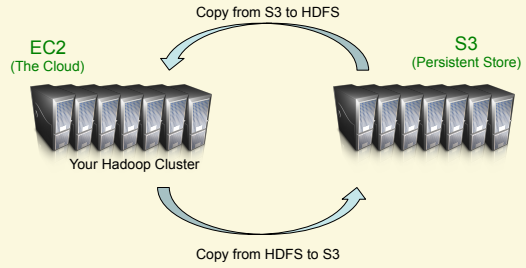


On Amazon: With EC2



Uh oh. Where did the data go?

On Amazon: EC2 and S3



Next time

- More on MapReduce
- Synchronization and coordinating partial results
- Use of complex keys and values
- Examples

Questions?