Lecture 22
GFS and MapReduce

Administration

- Last call for the datacenter tour!
  - Thurs @2:30PM, or Fri @10:30AM
  - Sign up at the Ctools-posted URL
  - Directions have been posted
  - Take a bus, drive, or get a ride
- Alpha Release this Friday @ 6PM
  - Please keep your sites up till 10AM Sat
- Midterm #2
  - Still being graded
  - Early results: better than MT#1

2

GFS and MapReduce

- Search engines were among first apps to address Web's scale directly
  - We spoke about distributing search queries
  - Also, distributing index construction
- Search engines were first, but not last
  - Social networks
  - Web-analysis and intelligence
  - Log processing
- Traditional data systems inappropriate; Google first to build somewhat general solutions

3

A Little Math

- How hard is indexing the Web really?
  - Say, 10B pages in 1 week
  - 10B in ~10K minutes
  - 1M pages every minute!
  - Disk scan rate 50MB/sec
  - Page takes average 25kb
  - 2000 pages/sec, or,
  - 120,000 pages/min *just to read off disk*
- What if we wanted to do it in a day?
  - An hour?
- Want to use clusters of 100s-1000s of machines, with local disks and some network

4

Distributed Programming

- Unlike distributed databases, many CPUs needed. 1000s, not dozens
  - Programmer cannot know how many machines at program-time or runtime
  - Even so, job is very long-lasting compared to most db queries
  - Machines die, machines depart; job must survive
- Also, cannot express program in SQL

5

MapReduce

- MapReduce system provides:
  - Automatic parallelization & distribution
  - Fault-tolerance
  - Status & monitoring tools
  - Clean abstraction for programmers

6

MapReduce

- Many data programs can be written as *map* and *reduce* functions
- *map* transforms **key, value** inputs into new **key', value'**
  - `Map(k, v) => (k', v') list`
- *reduce* receives all the vals for a given key' and can output to disk file
  - `Reduce(k', v' list) => (out-key, out-val) list`

7

Example: Filter

- ```
  Map(k, v)=>
  if (prime(k)) then emit(k, v)
  ```
- ("foo", 7) => ("foo", 7)
- ("bar", 10) => *nothing*

8

Example: Sum

- ```
  Reduce(k, vals)=>
  sum = 0
  foreach int v in vals:
    sum += v
  emit(k, sum)
  ```
- `("A", [42, 100, 312]) => ("A", 454)`
- `("B", [12, 6, -2]) => ("B", 16)`

9

Word counting

- ```
  Map(lineNo, textStr)
      for each word w in textStr:
        emit(w, 1)
  ```
- ```
  Reduce(outKey, interm-vals)
      int result = 0
      for each v in interm-vals:
        result += v
  ```
- `Emit(output-key, result)`

10

Applications

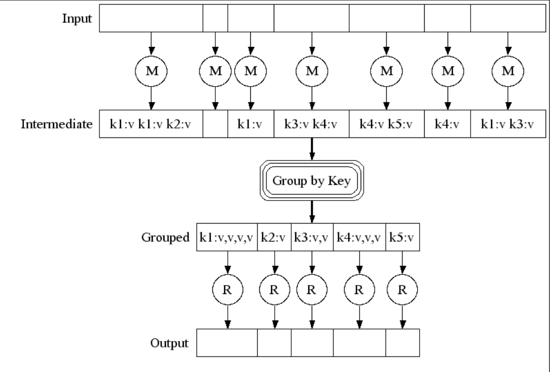
- What else can be an MR program?
  - URL counting in logs, Reverse Web graph
  - Inverted index construction, Sorting
  - Massive image conversion, others

11

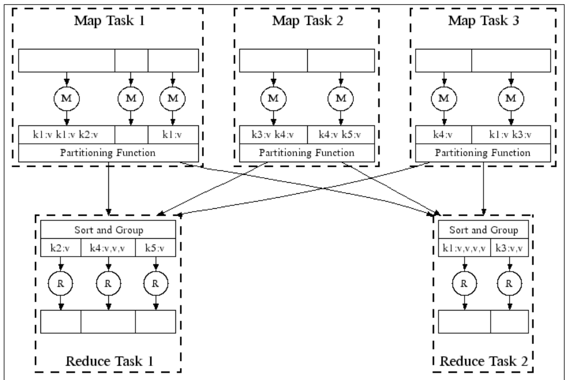
Shuffle/Sort

- What happens between map & reduce?
  - Data collated and grouped for map
- Execution goes as follows:
  - Break input into M chunks
  - Process each chunk w/ *map* process
  - Group map outputs into R chunks
  - Process each chunk w/ *reduce* process
  - *reduce* fn's outputs go to disk

12

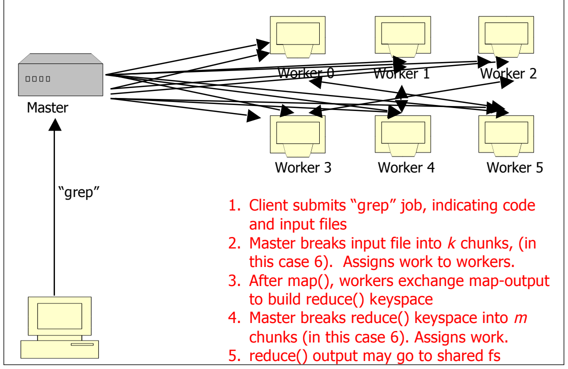


13



14

Map/Reduce Job Processing



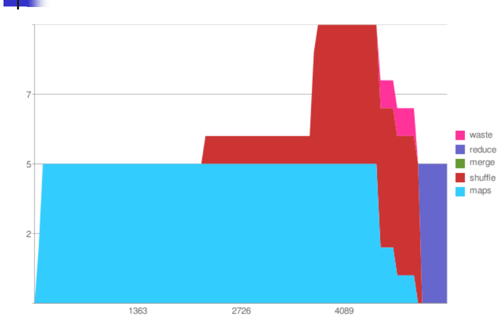
1. Client submits "grep" job, indicating code and input files
2. Master breaks input file into *k* chunks, (in this case 6). Assigns work to workers.
3. After map(), workers exchange map-output to build reduce() keyspace
4. Master breaks reduce() keyspace into *m* chunks (in this case 6). Assigns work.
5. reduce() output may go to shared fs

A few nice features

- What about slow, not dead, machines?
  - Speculative execution for stragglers
- What about data placement?
  - Use distributed filesystem across cluster disks; start tasks where the target data already lies
- Isn't the intermediate data size large?
  - Use a "local reducer" called a Combiner at each map
  - Compress data between map and reduce

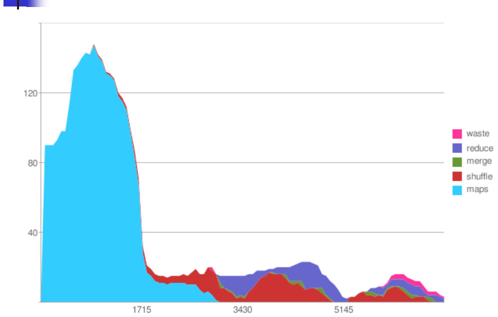
16

MapReduce Timelines



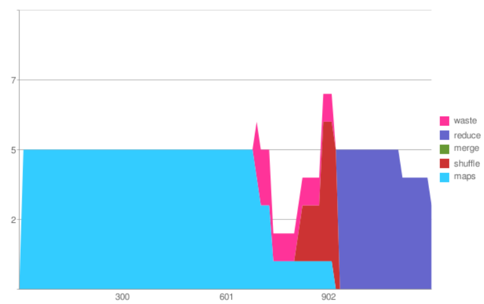
17

MapReduce Timelines



18

MapReduce Timelines



19

Performance

- The TeraSort benchmark measures time to sort 1TB (10B 100-byte records)
  - In 1998, record holder did it in 151 mins
  - In 2009, Yahoo/Hadoop MapReduce did it in 209 seconds
  - Then a few months later, Google did it in 68 seconds (on about 1,000 machines)
- MapReduce also holds the record for sorting a petabyte
  - (That's 10 trillion 100-byte records)
  - 6 hours, 2 mins on 4,000 machines

20

Intermission

- MapReduce has become extremely trendy. Any problem, no matter how unrelated, is now treated as a good target for MapReduce programming

- That said, it has its uses. For a different opinion, do a search for "MapReduce: A Major Step Backwards"

21

Google File System

- The distributed filesystem that stores data across a MapReduce cluster
- As with MR, Google had strange requirements and came up with strange design
- Has not proved as generally-useful as MapReduce

22

Weird Requirements

- High component failure rates
  - Inexpensive commodity components fail all the time
- "Modest" number of HUGE files
  - Just a few million
  - Each is 100MB or larger; multi-GB files typical
- Files are append-only
- Large streaming reads
- High sustained throughput favored over low latency

23

Design solution

- Files stored as chunks
  - Much larger size than most filesystems (default is 64MB)
- Reliability through replication
  - Each chunk replicated across 3+ *Chunkservers*
- Single master coordinates access, metadata
  - Simple centralized management
- No data caching
  - Little benefit due to large data sets, streaming reads
- Familiar interface, but customize the API
  - Simplify the problem; focus on distributed apps

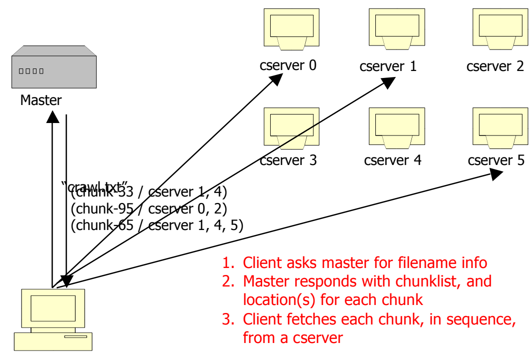
24

Design Details

- Chunks can be copied, replicated
- Chunkservers hold and serve chunks
- Master holds metainfo
  - Filename → chunk list
  - Chunk → chunkserver-location
- Chunkservers report in to master every few seconds
- Data/task co-location is extremely important

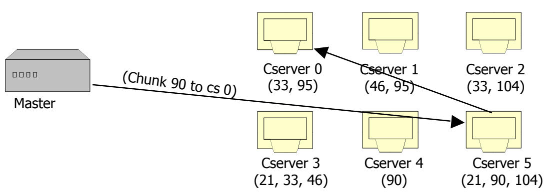
25

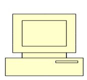
GFS File Read



1. Client asks master for filename info
2. Master responds with chunklist, and location(s) for each chunk
3. Client fetches each chunk, in sequence, from a cserver

GFS Replication





1. Always keep at least *k* copies of each chunk
2. Imagine cserver 4 dies; #90 lost
3. Master loses heartbeat, decrements #90's reference count. Asks cserver 5 to replicate #90 to cserver 0
4. Choosing replication target is tricky