Agenda

• Request and Data Level Parallelism
• Administrivia
• Technology Break
• Map-Reduce Examples

Request-Level Parallelism

• Hundreds or thousands of requests per second
  – Not your laptop or cell-phone, but popular Internet services like Google search
  – Such requests are largely independent
    • Little read-write (aka “producer-consumer”) sharing
    • Mostly involve read-only databases
    • Rarely involve read-write data sharing or synchronization across requests
• Computation easily partitioned within a request and across different requests

Google Query-Serving Architecture

Anatomy of a Web Search

• Google “David A. Patterson”
  – Direct request to “closest” Google datacenter
  – Front-end load balancer directs request to one of many clusters within the datacenter
  – Within cluster, select one of many Google Web Servers (GWS) to handle the request and compose the response pages
  – GWS communicates with Index Servers to find documents that contain the search words, “David”, “Patterson”
  – Return document list with associated relevance score
Anatomy of a Web Search

- In parallel,
  - Spell checker: “Did you mean David Paterson?”
  - Ad system: books by Patterson at Amazon.com
  - Images of David Patterson
- Use docids to access indexed documents
- Compose the page
  - Result document extracts (with keyword in context) ordered by relevance score
  - Sponsored links (along the top) and advertisements (along the sides)

Data-Parallel “Divide and Conquer” (Map-Reduce Processing)

- Map:
  - Slice data into “shards”, distribute these to workers, compute sub-problem solutions
  - map(in_key, in_value) -> list(out_key, intermediate value)
  - Processes input key/value pair
  - Produces set of intermediate pairs

- Reduce:
  - Collect and combine sub-problem solutions
  - reduce(out_key, list(intermediate_value)) -> list(out_value)
  - Combines all intermediate values for a particular key
  - Produces a set of merged output values (usually just one)

Google Uses MR For ...

- E.g.:
  - Extracting the set of outgoing links from a collection of HTML documents and aggregating by target document
  - Stitching together overlapping satellite images to remove seams and to select high-quality imagery for Google Earth
  - Generating a collection of inverted index files using a compression scheme tuned for efficient support of Google search queries
  - Processing all road segments in the world and rendering map tile images that display these segments for Google Maps
  - Fault-tolerant parallel execution of programs written in higher-level languages across a collection of input data

Map-Reduce Processing Time Line

- Master assigns map + reduce tasks to worker machines
- As soon as a map task finishes, it can be assigned a new map or reduce task
- Data shuffle begins as soon as a given Map finishes
- Reduce task begins as soon as all data shuffles finish
Agenda

- Request and Data-Level Parallelism
- Administrivia
- Technology Break
- Map-Reduce Examples
Agenda

- Data Level Parallelism
- Administrivia
- Technology Break
- Map-Reduce Examples

Map-Reduce Processing Example: Count Word Occurrences

- Pseudo Code:

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

  reduce(String output_key, Iterator intermediate_values):
  // output_key: a word
  // output_values: a list of counts
  int result = 0;
  for each v in intermediate_values:
    result += ParseInt(v);
  Emit(AsString(result));
  ```

Map-Reduce Processing: Execution
Another Example: Word Index (How Often Does a Word Appear?)

Distribute

that that is that that
Map 1
is 1, that 2

that is not that
Map 2
is 1, that 2, not 2

that is that it is
Map 3
is 2, it 2, that 1

Shuffle

that 1, 2, 2
is 1; it 2
Reduce 1

that 2, 2, 1
is 1; it 2
Reduce 2

not 2; that 5

Collect

is 1; it 2; not 2; that 5

Map-Reduce Failure Handling

• On worker failure:
  – Detect failure via periodic heartbeats
  – Re-execute completed and in-progress map tasks
  – Re-execute in progress reduce tasks
  – Task completion committed through master
• Master failure:
  – Could handle, but don’t yet (master failure unlikely)
• Robust: lost 1600 of 1800 machines once, but finished fine

Map-Reduce Redundant Execution

• Slow workers significantly lengthen completion time
  – Other jobs consuming resources on machine
  – Bad disks with soft errors transfer data very slowly
  – Weird things: processor caches disabled (!)
• Solution: Near end of phase, spawn backup copies of tasks
  – Whichever one finishes first "wins"
• Effect: Dramatically shortens job completion time

Map-Reduce Locality Optimization

• Master scheduling policy:
  – Asks GFS for locations of replicas of input file blocks
  – Map tasks typically split into 64MB (== GFS block size)
  – Map tasks scheduled so GFS input block replica are on same machine or same rack
• Effect: Thousands of machines read input at local disk speed
• Without this, rack switches limit read rate

Summary

• Request-Level Parallelism
  – High request volume, each largely independent of the other
  – Use replication for better request throughput and availability
• Map-Reduce Data Parallelism
  – Divide large data set into pieces for independent parallel processing
  – Combine and process intermediate results to obtain final result