CS 61C: Great Ideas in Computer Architecture (Machine Structures)

Instructors:
Randy H. Katz
David A. Patterson
http://inst.eecs.Berkeley.edu/~cs61c/fa10

Agenda

• Request and Data Level Parallelism
• Administrivia
• Technology Break
• Map-Reduce Examples
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)
Anatomy of a Web Search

- Implementation strategy
  - Randomly distribute the entries
  - Make many copies (aka replicas)
  - Load balance requests across replicas
- Redundant copies of indices and documents
  - Breaks up hot spots, e.g., “Justin Bieber”
  - Increases opportunities for request-level parallelism
  - Makes the system more tolerant of failures

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

<table>
<thead>
<tr>
<th>Process</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Program</td>
<td>MapReduce()   ... wait ...</td>
</tr>
<tr>
<td>Master</td>
<td>Assign tasks to worker machines...</td>
</tr>
<tr>
<td>Worker 1</td>
<td>Map 1 Map 3</td>
</tr>
<tr>
<td>Worker 2</td>
<td>Map 2</td>
</tr>
<tr>
<td>Worker 3</td>
<td>Read 1.1 Read 1.3 Read 1.2 Reduce 1</td>
</tr>
<tr>
<td>Worker 4</td>
<td>Read 2.1 Read 2.2 Read 2.3 Reduce 2</td>
</tr>
</tbody>
</table>

- 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
MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 10 min 18 sec
1707 workers, 1 deaths

<table>
<thead>
<tr>
<th>Type</th>
<th>Shards</th>
<th>Done</th>
<th>Active</th>
<th>Input(MB)</th>
<th>Done(MB)</th>
<th>Output(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>5354</td>
<td>5354</td>
<td>1707</td>
<td>872934.6</td>
<td>406020.1</td>
<td>241058.2</td>
</tr>
<tr>
<td>Shuffle</td>
<td>500</td>
<td>0</td>
<td>500</td>
<td>241058.2</td>
<td>196362.5</td>
<td>196362.5</td>
</tr>
<tr>
<td>Reduce</td>
<td>500</td>
<td>0</td>
<td>0</td>
<td>196362.5</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Counters

<table>
<thead>
<tr>
<th>Variable</th>
<th>Minute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapped (MB/s)</td>
<td>704.4</td>
</tr>
<tr>
<td>Shuffle (MB/s)</td>
<td>371.9</td>
</tr>
<tr>
<td>Output (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>doc-index-hits</td>
<td>5003642228</td>
</tr>
<tr>
<td>doc-indexed</td>
<td>17300709</td>
</tr>
<tr>
<td>dup-in-index-merge</td>
<td>0</td>
</tr>
<tr>
<td>mr-operator-calls</td>
<td>17342493</td>
</tr>
<tr>
<td>mr-operator-outputs</td>
<td>17300709</td>
</tr>
</tbody>
</table>

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 15 min 31 sec
1707 workers, 1 deaths

<table>
<thead>
<tr>
<th>Type</th>
<th>Shards</th>
<th>Done</th>
<th>Active</th>
<th>Input(MB)</th>
<th>Done(MB)</th>
<th>Output(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>5354</td>
<td>5354</td>
<td>1707</td>
<td>872934.6</td>
<td>621608.5</td>
<td>369459.8</td>
</tr>
<tr>
<td>Shuffle</td>
<td>500</td>
<td>0</td>
<td>500</td>
<td>369459.8</td>
<td>326986.8</td>
<td>326986.8</td>
</tr>
<tr>
<td>Reduce</td>
<td>500</td>
<td>0</td>
<td>0</td>
<td>326986.8</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Counters

<table>
<thead>
<tr>
<th>Variable</th>
<th>Minute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapped (MB/s)</td>
<td>706.5</td>
</tr>
<tr>
<td>Shuffle (MB/s)</td>
<td>419.2</td>
</tr>
<tr>
<td>Output (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>doc-index-hits</td>
<td>4982870667</td>
</tr>
<tr>
<td>doc-indexed</td>
<td>17229926</td>
</tr>
<tr>
<td>dup-in-index-merge</td>
<td>0</td>
</tr>
<tr>
<td>mr-operator-calls</td>
<td>17272056</td>
</tr>
<tr>
<td>mr-operator-outputs</td>
<td>17229926</td>
</tr>
</tbody>
</table>

9/20/10
Agenda

- Request and Data-Level Parallelism
- Administrivia
- Technology Break
- Map-Reduce Examples
British Chip Designer Prepares for Wider Demand

By APOLLO VANCE
Published: September 20, 2010

CAMBRIDGE, England — Near the southeastern edge of Cambridge, where this idyllic university town gives way to fields of green, sits the headquarters of ARM Holdings. Neither the modest three-building campus nor its surroundings evoke notions of a thriving hotbed of computing.

But ARM, which designs the low-power chips that go into just about every cellphone sold today, commands a prime position when it comes to one of the next major technological revolutions. This is the so-called Internet of Things, when all sorts of everyday objects will have tiny chips placed inside them and gain the ability to process information and talk to the Web.

In this post-PC era, some analysts say, Intel's familiar jingle — bummum, bum, bum, bum, bam — will fade as the central soundtrack of computing. Instead, people will hear nothing, or rather the understated silence that has accompanied ARM's rise as one of the most important technology companies.

ARM bases its business on licensing chip designs to companies like Apple, Samsung and Qualcomm, which often tweak them to suit their
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

- Map task 1: Maps key k1 hashes to Reduce Task 2.
- Map task 2: Maps key k2 hashes to Reduce Task 1.
- Map task 3: Maps key k3 hashes to Reduce Task 2.
- Map task 4: Maps key k4 hashes to Reduce Task 1.
- Map task 5: Maps key k5 hashes to Reduce Task 1.
- Shuffle.
- Reduce/Combine.
- Collect Results.
Another Example: Word Index (How Often Does a Word Appear?)

Distribute

Map 1
- that is 1, that 2

Map 2
- is that is 1, that 2

Map 3
- is not is not 2, not 2

Map 4
- is that it is 2, it 2, that 1

Shuffle

Reduce 1
- is 1,1,2,2; it 2
- Reduce 1

Reduce 2
- that 2,2,1
- not 2
- Reduce 2

Collect

is 6; 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