Cluster Computing (MapReduce)

15th century reconstruction of Ptolemy’s Geography (150 AD)

“reducing a map in late medieval style”, Stable Diffusion, November 21, 7:24AM
Processing Large Data

• Let’s distribute load over many machines
  • 1000s, not 2-16 as in traditional distributed databases
  • 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
MapReduce

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

• MapReduce has become very popular, for lots of good reasons
  • Easy to write distributed programs
  • Built-in reliability on large clusters
  • Bytestreams, not relations
  • “Schema-later”, or “schema-never”
  • Your choice of programming languages
  • Hadoop relatively easy to administer

• Should you use MapReduce instead of a database?
  This was very popular in late-2000s. Today, less so
A Story About MapReduce

• Imagine some fictional comedy sorority or fraternity has instituted a new “entrance” ritual. A student must compute:
  • How common are 1-character words? (‘a’, ‘I’, etc.)
  • How common are 2-character words? (‘an’, ‘be’, ‘is’, etc.)
  • ... up to 10-character words

• ... IN THE ENTIRE MIT LIBRARY
A Story About MapReduce

• A few (real) statistics
• ~6M volumes in the MIT library
• You have one semester
• You can recruit ~1,000 students to help
• In the end, we’ll have 10 numbers:
  • Count of one-character words
  • Count of two-character words
  • ... etc. until 10
A Story About MapReduce

• The next day near Stata:
• Divide the students into groups
• The **Mappers**
  • Thousands of people
• The **Grouper**
  • Just one person for now (in the real MapReduce system, the story is more complicated)
• The **Reducers**
  • Around 10
• The **Controller**
  • *You*
A Story About MapReduce

• Each **mapper** student gets a “reading list” of 6,000 books (welcome to college!)
  • That’s 6M books / ~1k first-year students
• And a notepad
• Instructions: write one line for each word you see in your reading list, along with the number of characters
  • 2, It
  • 3, was
  • 3, the
  • ... etc. many many many many times
A Story About MapReduce

• After the **mappers** are done, they hand their notebooks to the **grouper**
• The **grouper** has a 10 page notebook
• The **grouper** takes the mappers’ notebooks and writes every 1-letter word on page 1, 2-letter word on page 2, etc.
  • Sheet 1: a, a, a, l, a, … many more
  • Sheet 2: if, if, an, if, at … many more
  • …
  • Sheet 10: schnozzles, mozzarella, etc.
A Story About MapReduce

• Now, each of the 10 sheets goes to a reducer
• Each reducer counts the number of words on one sheet, and writes the number in bold letters on the back
• Remember, Sheet 2 has: if, of, it, of, of, if, at, im, is, is, of, of ...
• The reducer writes 2453838307534 on the back
A Story About MapReduce

• Now, the **controller** collects the 10 sheets and reads the back of each sheet, which is the number of 1-character words, 2-character words, etc.

• And you’re done!
A Story About MapReduce

A few observations

• The Mappers can work independently
• The Reducers can work independently
• The Grouper has a lot of work (collating and writing down each individual word on a sheet!) but didn’t have to do any counting (“real work”)
• All Grouper had to do was to look at the Mappers’ outputs and put that word on the appropriate sheet
A Story About MapReduce

• Ideas for optimizations?

• How could you reduce the amount paper used by the mappers?
A Story About MapReduce

• Ideas for optimizations?
  • Take a minute to write them down

• What steps CAN’T be optimized easily?
  • Take another minute
From Story to MapReduce Library

• The work of the Controller (dividing the work) and the Grouper (Grouping the values by key), remains the same
  • MapReduce library provides these
• Grouping is sometimes called ”sort” or “shuffle”
• The work of the mappers and reducers differs with problem
  • This is what you write
Programming Model

• The computation:
• Input key/value pairs
  • e.g., (book_title, book_content)
• Output different key/value pairs
  • e.g., (word_length, occurrences)

• The user of the MapReduce library expresses the computation as two functions....

• CAN YOU GUESS THEIR NAMES?
  • Map and Reduce
Map function

• User's map function takes an input pair and produces a set of intermediate key/value pairs
  
  ```python
  map(book_title, book_content):
    words = book_content.split()
    for word in words:
      word_length = len(word)
      EmitIntermediate(word_length, 1)
  ```

• The MapReduce library groups together all intermediate values associated with the same intermediate key and passes them to the Reduce function
Reduce function

• User's reduce function accepts an intermediate key and a list of values for that key. It merges together these values to form a possibly smaller set of values.

  • reduce(word_length, list_of_occurrences):
    sum = 0
    for i in list_of_occurrences:
      sum += i
    Emit(sum)
Example

• input01.txt
  Hello World Bye World

• input02.txt
  Hello Hadoop Goodbye Hadoop

• Task: count the number of words with 1 character, 2 characters, etc. (same as before)

• Spend 2 minutes and think about:
  • What are the inputs to the map steps?
  • What are the outputs of the map steps?
  • What are the inputs to the reduce steps?
  • What are the outputs of the reduce steps?
Example

• What are the inputs to the map steps?
  • Segments of the inputs
• For example,
• First call to map:
  • "input01.txt", "Hello World Bye World"
• Second call to map:
  • "input02.txt", "Hello Hadoop Goodbye Hadoop"
Example

- What are the outputs of the map steps?
- NOTE: order doesn't matter

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
</tr>
</tbody>
</table>
Example

- What are the inputs to the reduce steps?
- Prior to reduce(), MapReduce **groups** together the map() outputs like keys

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
</tr>
</tbody>
</table>
Example

• What are the outputs of the reduce steps?
• `<word_length, occurrences>`

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
</tr>
</tbody>
</table>
Types

• Map and reduce have related types
  • map \((k_1, v_1) \rightarrow \text{list}(k_2, v_2)\)
  • reduce \((k_2, \text{list}(v_2)) \rightarrow \text{list}(v_2)\)

• Final output list can be:
  • Smaller than input list (in the case of computing summary statistics, like word count)
  • Larger than input list (in the case of computing some kind of data structure for downstream use)

• Typically, just zero or one output value is produced per reduce invocation
Exercise: Word Count

• Count the number of occurrences of each word in a collection of web documents, identified by URL
• Exercise: write a map function and a reduce function
Exercise: Word Count

• Count the number of occurrences of each word in a collection of web documents, identified by URL

map(url, content):
    for word in content:
        EmitIntermediate(word, 1);

reduce(word, occurrences):
    Emit(sum(occurrences))
Exercise: Word Count

Inputs to map

• input01.txt
  Hello World Bye World

• input02.txt
  Hello Hadoop Goodbye Hadoop

Outputs of map

Hello 1
World 1
Bye 1
World 1
Hello 1
Hadoop 1
Goodbye 1
Hadoop 1

map(url, content):
  for word in content:
    EmitIntermediate(word, 1);
Exercise: Word Count

Inputs to reduce (grouped by MR)

<table>
<thead>
<tr>
<th>Word</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bye</td>
<td>1</td>
</tr>
<tr>
<td>Goodbye</td>
<td>1</td>
</tr>
<tr>
<td>Hadoop</td>
<td>1</td>
</tr>
<tr>
<td>Hadoop</td>
<td>1</td>
</tr>
<tr>
<td>Hello</td>
<td>1</td>
</tr>
<tr>
<td>Hello</td>
<td>1</td>
</tr>
<tr>
<td>World</td>
<td>1</td>
</tr>
<tr>
<td>World</td>
<td>1</td>
</tr>
</tbody>
</table>

Outputs of reduce

<table>
<thead>
<tr>
<th>Word</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bye</td>
<td>1</td>
</tr>
<tr>
<td>Goodbye</td>
<td>1</td>
</tr>
<tr>
<td>Hadoop</td>
<td>2</td>
</tr>
<tr>
<td>Hello</td>
<td>2</td>
</tr>
<tr>
<td>World</td>
<td>2</td>
</tr>
</tbody>
</table>

What if the number of unique words is small compared to the number of documents? Can you optimize this?

reduce(word, occurrences):
Emit(sum(occurrences))
Exercise: Word Count

- Another solution: sum the words within each doc

```python
map(url, content):
  for word in content:
    if word in counts_hash:
      counts_hash[word] += 1
    else:
      counts_hash[word] = 1

  occurrences = counts_hash.items() #to list
  EmitIntermediate(occurrences);   #list of (k,v)

reduce(word, occurrences):
  Emit(sum(occurrences))
```
Exercise: Word Count

<table>
<thead>
<tr>
<th>Output of map</th>
<th>Output of reduce</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hello</td>
<td>Bye</td>
</tr>
<tr>
<td>World</td>
<td>Goodbye</td>
</tr>
<tr>
<td>Bye</td>
<td>Hadoop</td>
</tr>
<tr>
<td>Hello</td>
<td>Hello</td>
</tr>
<tr>
<td>Hadoop</td>
<td>World</td>
</tr>
<tr>
<td>Goodbye</td>
<td></td>
</tr>
</tbody>
</table>

(same answer as before)

Input:
- input01.txt: Hello World Bye World
- input02.txt: Hello Hadoop Goodbye Hadoop

We’re summing at doc-level (in map()) and corpus-level (in reduce()).

What if we want to find the **average #** of occurrences for each word?

What about **median**?
Study Break 1! (take 3-5 mins)

• Write mapper and reducer functions for computing the dot product of two large vectors
  • Assume we have prepared A and B for you: (1,(Ai,Bi))

\[ A \cdot B = \sum_{i=1}^{n} A_i B_i = A_1 B_1 + A_2 B_2 + \cdots + A_n B_n \]
Dot product

- Write mapper and reducer functions for computing the dot product of two large vectors

$A \cdot B = \sum_{i=1}^{n} A_i B_i = A_1 B_1 + A_2 B_2 + \cdots + A_n B_n$

map(1, (ai, bi)):
  product = ai * bi
  EmitIntermediate(1, product)

reduce(1, product_list):
  Emit(1, sum(product_list))
Study Break 2! (take 3-5 mins)

• Write mapper and reducer functions for distributed search (AKA grep)
  • Print any line of a big input file that contains an input pattern as a substring
Linear search (grep)

• Write mapper and reducer functions for distributed search (AKA grep)
  • Print any line of a big input file that contains an input pattern as a substring

```python
map(filename, content):
    for line in content:
        if pattern in line:
            EmitIntermediate(1, line)

reduce(1, lines):
    for line in lines:
        Emit(1, line)
```
MapReduce vs the RDBMS

• **Schemas**: MR doesn’t have them, for better and worse

• **Functions**: MR doesn’t have a query language, but permits flexible UDFs

• **Execution and optimization**: MR has optimizations, but limited schemas mean limited options

• **Failure recovery**: MR can lose machines and keep going. Distributed RDBMS traditionally restarts queries

• **Transactions**: MR always yields new data. It never modifies data in place. Unclear semantics if the input data changes during processing.
Executing MapReduce

- MapReduce execution consists of 3 main stages:
  - Map
  - Shuffle/Sort (aka Group)
  - Reduce
- In stage 1, partition input data and run \texttt{map()} on many machines
- Then group intermediate data by intermediate key
- In stage 2, partition intermediate data by key and run \texttt{reduce()} on many machines
- Output is whatever \texttt{reduce()} emits
- We have local storage and shared storage
Shuffle/Sort

• What happens between map & reduce?
  • Data collated and grouped for map
  • Default: hash(key) MOD R

• This step is similar to the RDBMS shuffle join
  • What’s the join key? The intermediate mapper output key

• Execution goes as follows:
  • Break input into M chunks
  • Process each chunk w/ map process
  • Group-by map output keys
  • Place key-groups into R chunks
  • Process each chunk w/ reduce process
  • reduce fn’s outputs go to disk
Architecture
1. Client submits “grep” job, indicating code and input files
1. Client submits “grep” job, indicating code and input files
2. Controller breaks input file into $k$ chunks, (in this case 6). Assigns work to workers.
1. Client submits “grep” job, indicating code and input files
2. Controller 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
Client submits "grep" job, indicating code and input files
2. Controller 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. Controller breaks reduce() keyspace into $m$ chunks (in this case 6). Assigns work.
1. Client submits “grep” job, indicating code and input files
2. Controller 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. Controller breaks reduce() keyspace into $m$ chunks (in this case 6). Assigns work.
5. reduce() output may go to shared fs
Applications

• What else can be a MapReduce program?
  • URL counting in logs
  • Inverted index construction for search engines, Sorting
  • Massive image conversion, others
Robustness

• How do we know if a machine goes down?
  • Heartbeat messages tell master which machines are online

• What happens to the job with MapReduce?

• What happens without MapReduce? (say, in an RDBMS)
Robustness

• What happens when a machine dies?

• **With** MapReduce

• If a map() worker dies
  • Just restart that task on a different box
  • You lose the map() work, but no big deal

• If a reduce() worker ides
  • Restart the reducer, using output from source mappers
Robustness

• What happens when a machine dies?

• **Without** MapReduce, in a traditional RDBMS
• Query is restarted
• Not so hot if your job is in hour 23

• *Recovery in the face of partial failure* is maybe MapReduce’s most important contribution
A few nice features

• What about slow, not dead, machines?
  • Speculative execution for stragglers
  • Kill the 2nd-place finisher

• What about data placement?
  • Spread input files 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
(a) Normal execution  
(b) No backup tasks  
(c) 200 tasks killed
Key observations

• Scalability and fault-tolerance achieved by optimizing the execution engine once
  • Use it many times by writing different map and reduce functions for different applications

• Stateless mapper

• Stateless reducer
Key observations

• Map and reduce functions inspired by functions of the same name in Lisp programming language

• Functional programming
  • Computation as the evaluation of mathematical functions

• Functions have no side effects
  • AKA "pure" functions
  • AKA stateless
  • Does not change state outside itself

• Easy to parallelize!
Further Reading

• Some researchers disagree with MapReduce's popularity: “MapReduce: A Major Step Backwards”
  • https://homes.cs.washington.edu/~billhowe/mapreduce_a_major_step_backwards.html