Distributed Image Processing using Hadoop MapReduce framework

Binoy A Fernandez (200950006)
Sameer Kumar (200950031)
Distributed Image Processing
Objective

• To demonstrate how the hadoop mapreduce framework can be extended to work with image data for distributed image processing.
Distributed Image Processing
Problem Statement

- Distributed Sobel Edge Detection algorithm
  - Easy to understand and implement, and at the same time, it is computationally expensive.
  - Has the property that the new value to be calculated for a pixel depends not only on that pixel's original value but also on the values of surrounding pixels.
Distributed Image Processing
Basic Idea

• An image can be considered as made up of a two dimensional array of pixels. We can partition the image data into subsets and operate on the data in parallel by distributing subsets to different map tasks.
Distributed Image Processing
Edge Detection Algorithm

Convolution at a single pixel
1. Create kernel $h$ indexed from 0 to $m-1$ horizontally and 0 to $n-1$ vertically and populate it with kernel coefficients
2. Compute kernel half width, $m_2 = \text{floor}(m/2)$
   Compute kernel half height, $n_2 = \text{floor}(n/2)$
3. $\text{sum} = 0$
   $\text{for } k = -n_2 \text{ to } n_2 \text{ loop}$
      $\text{for } j = -m_2 \text{ to } m_2 \text{ loop}$
         $\text{sum} = \text{sum} + h(j + m_2, k + n_2) \, f(x - j, y - k)$
      $\text{end loop}$
   $\text{end loop}$
$g(x, y) = \text{sum}$
Distributed Image Processing
Edge Detection Algorithm

Convolution of an image ignoring the borders

1. Create kernel h indexed from 0 to m-1 horizontally and 0 to n-1 vertically and populate it with kernel coefficients.
2. Compute kernel half width, m2 = floor(m/2)
   Compute kernel half height, n2 = floor(n/2)
3. Create an M x N output image, g
4. for all pixel co-ordinates, x and y, loop
   g(x, y) = 0
   end loop
5. for y = n2 to N-n2-1 loop
   for x = m2 to M-m2-1 loop
     Compute g(x, y) using previous algorithm
   end loop
   end loop
Distributed Image Processing
Implementation Details

- **InputFormat**
  Responsible for,
  1. creating the input splits (`getSplits`)
  2. dividing the splits into records (`getRecordReader`)

- **InputSplit**
  A chunk of the input that is processed by a single map.

- **RecordReader**
  An input split is divided into records, and the map processes each record (key-value pair) in turn. A RecordReader is used to generate record key-value pairs.
Distributed Image Processing
Implementation Details

- **FileInputFormat**
  Base class for all implementations of InputFormat that use files as their data source. Provides,
  - methods to define which files are included as the input.
  - an implementation for generating splits for the input files.
Distributed Image Processing
Implementation Details

- **ImageInputFormat**
  - Reuse the methods provided for defining which files are to be included as input to a job.
  - Define a filter to accept only certain image formats.
  - Define a custom implementation for generating splits for the input files.
Distributed Image Processing
Implementation Details

- **Splitting Logic**
  Assumption: Size of an ImageSplit will not be greater than HDFS block size.

- Consider image of dimensions 20 x 35 pixels. Based on numSplits argument, the image is divided horizontally into splits.

**Performance**: To compute the image splits, we need only the image dimensions. This information can be obtained from the image header thereby avoiding having to read the entire image data into memory.
Distributed Image Processing
Implementation Details

• **ImageReader**
  A record reader implementation for reading sub images based on the computed splits, which is processed independently by the map tasks.

• Key: input image path

• Value: an ImageWritable

• Options :
  • Read the entire image into memory as a BufferedImage and use the getSubImage method.
  • Read only the required portion of the image as defined by the split, into memory.
Distributed Image Processing
Implementation Details

1. Compute the input splits
   - MapReduce Program
   - JobClient
   - Client JVM
   - Client Node
   - Copy Job Resources
   - Shared Filesystem

2. Schedule the map tasks
   - Submit Job
   - JobTracker
   - JobTracker Node

3. Obtain RecordReader for the split
   - Retrieve Input Splits
   - Retrieve Job Resources
   - TaskTracker
   - TaskTracker Node
   - Child JVM
   - Map / Reduce
Distributed Image Processing
Implementation Details

- **ImageWritable**
  - Custom hadoop type which implements hadoop’s serialization framework - Writable interface.
  - Data to serialize/de-serialize
    - Size of the image (in bytes)
    - Start Y co-ordinate relative to input image
    - Height of the image (in pixels)
    - Image path
    - Image data (byte array)
Distributed Image Processing
Implementation Details

- **Serialization**
  How to serialize a BufferedImage using Hadoop's serialization framework?

1. **Serialize RGB colour data**
   Drawback: Color conversion takes place if the default model does not match the image ColorModel.

2. **Serialize raw pixel data as a byte array**
   Drawback: Requires conditional code based on image type.

3. **Serialize in a preferred image format using ImageIO**
   Advantages:
   1) do not have to deal directly with underlying pixel data representation.
   2) intermediate image formats can be specified using custom properties.
   3) default PNG format used provides lossless zip compression.
Distributed Image Processing
Implementation Details

- **OutputFormat**
  Responsible for,
  1. writing output for the job to the file system (getRecordWriter)
  2. checking for validity of output specification of job (checkOutputSpecs)

- **RecordWriter**
  A RecordWriter is used to write the output key-value pairs to an output file.
Distributed Image Processing
Implementation Details

- **FileOutputFormat**
  Base class for all implementations of OutputFormat that use files as their data sink. Provides,
  - methods to define the output specifications for the job.
  - an implementation of checkOutputSpecs.
Distributed Image Processing
Implementation Details

- **ImageOutputFormat**
  - ImageOutputFormat is designed similar to the MultipleOutputFormat abstract class, which allows you to write data to multiple files whose names are derived from the output keys and values.
  - ImageOutputFormat provides a RecordWriter implementation for writing the image to the file system.
  - **Key**: image path, based on which the output file name is generated.
  - **Value**: image and its metadata, written to file system using ImageIO in the preferred output format.
Distributed Image Processing
Implementation Details

- **Map and Reduce function**

- The map function is an implementation of the Sobel edge detection algorithm applied to the input sub image.

- The reduce function uses the metadata associated with each image split to combine them into the final processed output image.
Distributed Image Processing
Test Case

- Input Image
Distributed Image Processing
Test Case

- Output Image
Distributed Image Processing Performance

- **Cluster Details**
  - Master node
    Intel Pentium 4 CPU @ 2.60 GHz, 1.00 GB RAM
  - Slave node
    Intel Core 2 Duo T9550 @ 2.66 GHz, 1.95 GB RAM

- **Input Data Set**
  - 6 hi-resolution PNG images
    Largest: 3000 pixels, Smallest: 1080 pixels (height)

- **Cluster Settings**
  - Maximum concurrent map tasks per tasktracker = 2
  - Number of reduce tasks for job = 2
  - Number of splits per image = 2
  - DFS replication = 2
Distributed Image Processing Performance

- **Wall Clock Time**
  - Serial Execution (Master)
    Begin: 8:38:20, End: 8:43:05
    Time: 4 minutes, 45 seconds
  - Parallel Execution (Cluster)
    Begin: 7:38:05, End: 7:40:55
    Time: 2 minutes, 50 seconds
Distributed Image Processing
Job Details

Live Datanodes : 1

<table>
<thead>
<tr>
<th>Node</th>
<th>Last Contact</th>
<th>Admin State</th>
<th>Configured Capacity (GB)</th>
<th>Used (GB)</th>
<th>Non DFS Used (GB)</th>
<th>Remaining (GB)</th>
<th>Used (%)</th>
<th>Used (%)</th>
<th>Remaining (%)</th>
<th>Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.176.99.90</td>
<td>0</td>
<td>In Service</td>
<td>30.01</td>
<td>0.09</td>
<td>13.9</td>
<td>16.02</td>
<td>0.27</td>
<td></td>
<td>53.4</td>
<td>41</td>
</tr>
</tbody>
</table>

Dead Datanodes : 1

<table>
<thead>
<tr>
<th>Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.176.99.118</td>
</tr>
</tbody>
</table>

Hadoop job_200911241818_0003 on bifernan-idc

User: biferman
Job Name: SobelDriver
Job File: https://biferman-idc:9100/tmp/hadoop-biferman/mapred/system/job_200911241818_0003/job.xml
Job Setup: Successful
Status: Succeeded
Started at: Tue Nov 24 19:38:21 IST 2009
Finished at: Tue Nov 24 19:40:54 IST 2009
Finished In: 2mins, 32sec
Job Cleanup: Successful

<table>
<thead>
<tr>
<th>Kind</th>
<th>% Complete</th>
<th>Num Tasks</th>
<th>Pending</th>
<th>Running</th>
<th>Complete</th>
<th>Killed</th>
<th>Failed/Killed Task Attempts</th>
</tr>
</thead>
<tbody>
<tr>
<td>map</td>
<td>100.00%</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td>12</td>
<td>0</td>
<td>0 / 1</td>
</tr>
<tr>
<td>reduce</td>
<td>100.00%</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0 / 0</td>
</tr>
</tbody>
</table>
## Distributed Image Processing

### Job Details

<table>
<thead>
<tr>
<th></th>
<th>Counter</th>
<th>Map</th>
<th>Reduce</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parsed SubImages</td>
<td>12</td>
<td></td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>Total SubImages</td>
<td>12</td>
<td></td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td><strong>File Systems</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HDFS bytes read</td>
<td>52,629,836</td>
<td></td>
<td>0</td>
<td>52,629,836</td>
</tr>
<tr>
<td>HDFS bytes written</td>
<td>0</td>
<td>37,397,249</td>
<td>37,397,249</td>
<td></td>
</tr>
<tr>
<td>Local bytes read</td>
<td>0</td>
<td>31,801,646</td>
<td>31,801,646</td>
<td></td>
</tr>
<tr>
<td>Local bytes written</td>
<td>31,802,402</td>
<td>31,801,646</td>
<td>63,604,048</td>
<td></td>
</tr>
<tr>
<td><strong>Job Counters</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Launched reduce tasks</td>
<td>0</td>
<td></td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Launched map tasks</td>
<td>0</td>
<td></td>
<td>0</td>
<td>13</td>
</tr>
<tr>
<td>Data-local map tasks</td>
<td>0</td>
<td></td>
<td>0</td>
<td>13</td>
</tr>
<tr>
<td><strong>Output</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Processed Images</td>
<td>0</td>
<td></td>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>
## Distributed Image Processing
### Job Details

### All Tasks

<table>
<thead>
<tr>
<th>Task ID</th>
<th>Complete</th>
<th>Status</th>
<th>Start Time</th>
<th>Finish Time</th>
<th>Errors</th>
<th>Counters</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_200911241818_0003_m_000000</td>
<td>100.00%</td>
<td>hdfs://bifernar-idc:9100/user/bifernar/input/image1.png:0+1500</td>
<td>24-Nov-2009 19:38:23</td>
<td>24-Nov-2009 19:38:30 (38sec)</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>task_200911241818_0003_m_000001</td>
<td>100.00%</td>
<td>hdfs://bifernar-idc:9100/user/bifernar/input/image1.png:1500+1500</td>
<td>24-Nov-2009 19:38:33</td>
<td>24-Nov-2009 19:40:07 (1min, 34sec)</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>task_200911241818_0003_m_000002</td>
<td>100.00%</td>
<td>hdfs://bifernar-idc:9100/user/bifernar/input/image2.png:0+1089</td>
<td>24-Nov-2009 19:38:34</td>
<td>24-Nov-2009 19:38:50 (16sec)</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>task_200911241818_0003_m_000003</td>
<td>100.00%</td>
<td>hdfs://bifernar-idc:9100/user/bifernar/input/image2.png:1089+1089</td>
<td>24-Nov-2009 19:38:35</td>
<td>24-Nov-2009 19:38:50 (1min, 11sec)</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>
Distributed Image Processing Enhancements

- Handle global and local boundary conditions
- Handle images larger than HDFS block size
- Evaluate performance of multi-threaded map runner
Thanks!