Towards Engineering a Web-Scale Multimedia Service: A Case Study Using Spark

MMSys 2017

Presented by:
Gylfi Þór Guðmundsson
gylfig@ru.is

Co-authors:
Laurent Amsaleg
Björn Þór Jónsson
Michael J. Franklin
Motivation

• Primary objective:
  - How can a typical multimedia-task harness the power of cloud-based processing?

• Observations:
  - Multimedia collections are growing (*Web-Scale*)
  - Computing power is abundant
    - Automatically Distributed Computational Frameworks (ADCFs)
  - Low-response time can be hard to achieve
  - High throughput services have a greater potential
Design Choices: System

- We choose as our evaluation system: **Content-Based Image Retrieval (CBIR)**
  - Searching based on the image itself
  - The CBIR system must capture, quantify and perform
  - Off-line process (*indexing*) & On-line process (*search*)

![Diagram showing the process of using a query image to retrieve search results using a CBIR system](image-url)
Design Choices: Algorithm

• We use an algorithm called:

\textit{Distributed extended Cluster Pruning (DeCP)}

- Clustering-based
- Deep hierarchical index
- Uses an approximate \(k\)-NN search
- Trades \textit{response time} for \textit{throughput} by batching

• Because:

- It is very simple
- It is prototypical of other CBIR algorithms
- It has previously been adapted for Hadoop
We implement DeCP on an ADCF called:

**Apache SPARK**

- Data resides in Resilient Distributed Datasets (RDDs)
- Transform one RDD into another via operators
- Master and Workers paradigm
- Supports deep pipelines
- Lazy execution allows for optimizations
Design Choices: Dataset

- YLI feature corpus from Yahoo-Flickr's YFCC100M collection
  - Various audio, visual and motion features from 100M images and 800,000 videos
  - Largest dataset publicly available

- We use the 42.9 billion SIFT features and we keep ALL the data
  - Goal is to test at a very large scale
  - No feature aggregation or compression is used
Research Questions

Questions we set out to answer:

1) What is the complexity of the Spark pipeline for typical multimedia-related tasks?

2) How well does background processing scale as collections size and resources grow?

3) How does batch size impact throughput of an online service?
Requirements for the ADCF

R1: Scalability
   - Ability to scale out with additional computing power

R2: Computational flexibility
   - Ability to carefully balance system resources as needed

R3: Capacity
   - Ability to gracefully handle data that vastly exceeds main memory capacity
**Requirements for the ADCF**

*R4: Updates*
- Ability to gracefully update the data structures for dynamic workloads

*R5: Flexible pipelines*
- Ability to easily implement variations of the indexing and/or retrieval process

*R6: Simplicity*
- Evaluate how efficiently the programmers time is spent
Inside the Black Box

Query Image → CBIR System → Search results
DeCP as a CBIR System

- The three steps of DeCP:
  I. Build the index hierarchy
     - Done only once
  II. Cluster the data collection
     - Very CPU intensive
  III. Approximate $k$-NN search
     - Default 1 clusters is searched
Full Featured Search

- **Vote aggregation**
  - Go from many local feature $k$-NNs to image results

- **Search expansion**
  - Searching >1 cluster will produce more $k$-NNs

- **High throughput batch search**
  - Millions of query features from thousands of images
Prior work evaluated DeCP on Hadoop using 30 billion SIFTs on 100+ machines

- The conclusion was a limited success:
  - Scalability limited due to RAM per core issue
  - 2-step M-R pipeline is too rigid to fully implement search

- The six requirements:
  - R1-3 are only partially satisfied
  - R4 is not feasible
  - R5 & R6 are not satisfied
DeCP's Approximate Search

For each feature:
1) Index query feature
2) Retrieve cluster
3) Populate $k$-NN

Clusters reside on disk
Clustered features

Clusters reside on disk

Indexed query features

DeCP's Approximate Search

Retrieve and Scan

At least one $k$-NN created for each local feature

More with search expansion

<table>
<thead>
<tr>
<th>Rank</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>E</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rank</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>2</td>
<td>A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rank</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>F</td>
</tr>
<tr>
<td>2</td>
<td>A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rank</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>C</td>
</tr>
</tbody>
</table>
Clusters reside on disk

Indexed query vectors

DeCP's Approximate Search

Retrieve and Scan

2-NN

<table>
<thead>
<tr>
<th>Rank</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>E</td>
</tr>
</tbody>
</table>

2-NN

<table>
<thead>
<tr>
<th>Rank</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>2</td>
<td>A</td>
</tr>
</tbody>
</table>

2-NN

<table>
<thead>
<tr>
<th>Rank</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>F</td>
</tr>
<tr>
<td>2</td>
<td>A</td>
</tr>
</tbody>
</table>

2-NN

<table>
<thead>
<tr>
<th>Rank</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>C</td>
</tr>
</tbody>
</table>

ID Votes

A: 4
B: 1
C: 1
E: 1
F: 1

Aggregated Results

Clustered features

Clusters reside on disk
Search Pipeline in Hadoop

- Batch-Search for high throughput
- Lookup-table is created a priori using the index
- Mappers create $k$-NNs, Reducers do aggregation
Spark

• A very different ADCF from Hadoop

• Primary advantages for DeCP are:
  • Arbitrarily deep pipelines
    – Easily implement all features and functionality
  
  • Broadcast variables
    – Solves the RAM per core limitation

  • Multiple data sources
    – Allows join operations for maintenance (R4)
.map is a one-to-one transformation

.flatmap is a one-to-any transformation

.groupbykey will do a “Shuffle”

.reduceByKey is like Hadoop's Reduce

.collectAsMap collects distributed data to the Master
Searching on Spark

Search

Point level with search expansion

Figure 2: Spark pipeline for batch k-NN search.
Indexing

With support for search expansion

Search

Point level with search expansion

Figure 2: Spark pipeline for batch $k$-NN search.
Evaluation: Specs

- Hardware: 51 AWS c3.8x1 nodes
  - 800 real cores / 1600 virtual cores
  - 2.8 TB of RAM and 30 TB of SSD space

- Dataset: 42.9 billion SIFT features (~7TB)
  - From almost 100 million Flickr images

- 5-level deep Index with 20 million clusters
  - Index for 300-400 billion features
Evaluation: Indexing

- Indexing all 43 billion features:
  - 5 hours 30 minutes

- Scalability – Hardware:
  - 8.5 billion features on 400 --> 800 cores took 0.59x time

- Scalability - Data:

<table>
<thead>
<tr>
<th>Billion features</th>
<th>Indexing time (seconds)</th>
<th>Relative scaling</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.5</td>
<td>3,287</td>
<td>–</td>
</tr>
<tr>
<td>17.2</td>
<td>5,030</td>
<td>1.53</td>
</tr>
<tr>
<td>26.0</td>
<td>11,943</td>
<td>3.63</td>
</tr>
<tr>
<td>34.5</td>
<td>14,192</td>
<td>4.31</td>
</tr>
<tr>
<td>42.9</td>
<td>19,749</td>
<td>6.00</td>
</tr>
</tbody>
</table>
• DeCP sacrifices response time for throughput
**Summary of Requirements**

**R1: Scalability**

**R2: Computational Flexibility**

**R3: Capacity**

**R4: Updates**

**R5: Flexible pipelines**

**R6: Simplicity**

<table>
<thead>
<tr>
<th></th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
<th>R4</th>
<th>R5</th>
<th>R6</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Spark</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes, full re-shuffle</td>
<td>Yes</td>
<td>Yes, Scala</td>
</tr>
<tr>
<td><strong>Hadoop</strong></td>
<td>Yes, Ram per core limit</td>
<td>Partial</td>
<td>Partial</td>
<td>Infeasible</td>
<td>No</td>
<td>No, Hard to fit the 2-step M-R</td>
</tr>
</tbody>
</table>

25
Conclusions

• Answers to the questions:

1) **What is the complexity of the Spark pipeline for typical multimedia-related tasks?**
   
   *With Spark we could easily implement a fully featured CBIR system based on DeCP*

2) **How well does background processing scale as collections size and resources grow?**
   
   *We pushed the boundaries to near Web-Scale*

3) **How does batch size impact throughput of an online service?**
   
   *We showed that high throughput search is possible even when keeping all the data*