Enhancing MapReduce using MPI and an optimized data exchange policy

Hisham Mohamed and Stéphane Marchand-Maillet

Viper group, CVML Laboratory, University of Geneva
September 10, 2012
• Motivation
• MapReduce
• MapReduce overlapping using MPI (MRO-MPI)
• Experiments
  – Wordcount
  – Distributed inverted files.
• Conclusion
Motivation

- Cross Modal Search Engine (CMSE)
Motivation

• Scalability
  – Multimedia Data increases rapidly.
    • Indexing
    • Searching
  
  – High dimensional data.
Our proposed solution

• In CMSE, we need data and algorithm parallelization.

• MapReduce overlapping using MPI (MRO-MPI)
  – C/C++ implementation of MapReduce using MPI.
  – Improving the MapReduce Model.
  – Maintain the usability of the Model.
Outline

• Motivation

• MapReduce
  • MapReduce overlapping using MPI (MRO-MPI)

• Experiments
  – Wordcount
  – Distributed inverted files.

• Conclusion
MapReduce brings a simple and powerful interface for data parallelization, by keeping the user away from the communications and the exchange of data.
The current model for MapReduce has at least three bottlenecks:

- Dependence.
- Multiple Disk access.
- All-to-All communication.
MapReduce Overlapping (MRO)

• Send partial intermediate \((K_m, V_m)\) pairs to the responsible reducers.

• We rule out:
  – The multiple read/write.
  – Shuffling phase is merged with the mapping phase.
  – Reducers do not wait until the mappers finish their work.

• Difficulties:
  – Rate of sending data between Mappers and Reducers.
  – The ratio between the Mappers and Reducers.
MapReduce Overlapping using MPI (MRO-MPI)

• MapReduce
  – Data parallelization.

• Message Passing Interface (MPI)
  – Separate processes with a unique rank.
  – MPI supports point-to-point, one-to-all, all-to-one and all-to-all communications.
  – Communication between processes.

• MapReduce-MPI
  – Based on the original MapReduce Model
MRO-MPI

1. **Chunk**\(_{1-N}\) → **Map**\(_{1-N}\) → **MapHashSet**\(_{h_kL, Plist(<K_m, V_m>)}\)

2. **System**
   - \(h_k = \text{Partitioning Function}(K_m)\)
   - **Send**
     - **Yes**: \(\# \text{pairs} > \text{Threshold}\)
     - **No**: **concatenate**()
     - **MPI_Send**(\(Plist(<K_m, V_m>), h_k\))

3. **Sending data from mapper to the Reducer with rank** \(h_k\)

4. **System**
   - **MPI_Recv**(\(Plist(<K_m, V_m>), MPI\_ANY\_SOURCE\))
   - **ReduceHashSet**\(_{K_m, Plist(V_m)}\)

5. **System**
   - **save processed pairs**
   - **concatenate**()
   - **Reduce Plist(v)**
     - **No**: **concatenate**()
     - **Yes**: **All Data**
     - **No**: **Recv more Plist(v)**

6. **User**
   - **Reduce** \((K_m, Plist(V_m))_{1-L}\)
   - **mapping finished** → **Output\(_{1-L}\)**
MRO-MPI
MRO-MPI

- **User**
  - Chunk\(_{1-N}\) → Map()\(_{1-N}\) → MapHashTable\(_{hk,L,Plist(K_m,V_m)}\)
  - \(hk=\text{Partitioning Function}(K_m)\)
  - MPI_Send(Plist\((K_m,V_m)\),hk)

- **System**
  - Sending data from mapper to the Reducer with rank \(hk\)
  - MPI_Recv(Plist\((K_m,V_m)\), MPI_ANY_SOURCE)
  - ReduceHashTable\(_{K_m,Plist(V_m)}\)
  - save processed pairs
  - Reduce Plist(v)
  - All Data
  - No

- **Output**
  - Reduce \((K_m,Plist(V_m))_{1-L}\)
  - mapping finished
  - Output\(_{1-L}\)
MRO-MPI
MRO-MPI

- Rate of Sending the data
MRO-MPI

1. User provides Chunk\(_{1-N}\) and Map\(_{1-N}\) functions.
2. System calculates partitioning function \(hk\).
3. Map\(\langle K_m, V_m\rangle\) is placed in \(\text{MapHashTable}\langle hk, Plist\langle K_m, V_m\rangle\rangle\).
4. If the number of pairs is greater than the threshold, concatenate and send to the Reducer with rank \(hk\).
5. Send data from mapper to the Reducer with rank \(hk\).
6. System performs MPI\(_{\text{Send}}\) with \(Plist\langle K_m, V_m\rangle, hk\).
7. ReduceHash\(\langle K_m, Plist\langle V_m\rangle\rangle\) is performed in the Reducer.
8. System saves processed pairs and receives more \(Plist\langle v\rangle\) if necessary.
9. User reduces \(K_m, Plist\langle V_m\rangle\) and sets mapping finished.
10. Output is generated.
MRO-MPI

Diagram:

- **Chunk_{1-N}**
  - **K/V**
  - **Map()_{1-N}**
  - **K_m/V_m**

  - **h_k = Partitioning Function(K_m)**
    - **MapHashTable<h_k, Plist(<K_m, V_m>)>**

  - **concatenate()**
    - **Yes**
    - **# pairs > Threshold**
    - **No**

  - **MPI_Send(Plist(<K_m, V_m>), h_k)**

  - **Sending data from mapper to the Reducer with rank h_k**

  - **MPI_Recv(Plist(<K_m, V_m>), MPI_ANY_SOURCE)**

  - **ReduceHashTable<K_m, Plist(V_m)>**

  - **save processed pairs**
  - **Reduce Plist(v)**
    - **No**
    - **concatenate()**
      - **Yes**
      - **All Data**

  - **Recv more Plist(v)**
    - **Yes**
    - **mapping finished**

  - **User**
    - **Reduce (K_m, Plist(V_m))_{1-L}**
      - mapping finished

  - **Output_{1-L}**

**Time**
MRO-MPI

- Same simple interface
  - Extra parameters:
    - Rate of sending data.
    - Number of Mappers to Reducers.
    - Data type.
Outline

• Motivation
• MapReduce
• MapReduce overlapping using MPI (MRO-MPI)
• Experiments
  – Wordcount
  – Distributed inverted files.
• Conclusion
WordCount

• WordCount:
  – Reads text files and counts how often words occur.
  – Input data size varies from 0.2Gb to 53Gb from project Gutenberg.
WordCount

- MRO-MPI: 24 as mappers and 24 as reducers.
- MR-MPI: 48 cores are used as mappers then as reducers.
- Hadoop: 48 reducers and the number of mappers varies according to the number of partial input files.

X-axis: Data size in gigabytes. Y-axis: log 10 of the running time. Values in the table show the running time in seconds. Values above the columns shows the size of each chuck.
Outline

• Motivation
• MapReduce
• MapReduce overlapping using MPI (MRO-MPI)
• Experiments
  – Wordcount
  – Distributed inverted files.
• Conclusion
Inverted Files

- **Inverted Files** is an indexing structure composed of two elements: the *vocabulary* and the *posting lists*.
  - Vocabulary
  - Posting lists

Name= Doc1  #id=1

Computer security known as information security as applied to computers and networks...........

Name= Doc1  #id=2

MapReduce has been used as a framework for distributing larger corpora............

Name= Doc1  #id=3

Protesters have been clashing with security forces. No information............

......
......
......
......

**Vocabulary**

- apply
- clash
- Corpora
- Compute
- framework
- force
- information
- large
- MapReduce
- networks
- protest
- security

**Posting Lists**

- <1,tf-idf>,
- ...
- ...
- <1,tf-idf>
- ...
- ...
- <2,tf-idf>
- ...
- ...
- <1,tf-idf>,<3,tf-idf>
- ...
- ...
- <1,tf-idf>,<3,tf-idf>
Inverted Files – tf-idf

- **tf-idf** - weighting scheme (SMART system, 1988):
  - Used to evaluate how important a word in a document with respect to other documents in the corpus.
  - **Term Frequency (tf):**
    \[ tf_{ij} = \frac{freq_{ij}}{\max_i freq_{ij}} \]
    - \( freq_{ij} \): number of occurrence of term \( t_i \) in document \( d_j \).
  - **Inverse Document Frequency (idf):**
    \[ idf_i = \log \frac{N}{n_i} \]
    - \( n_i \): number of documents where \( t_i \) appears.
    - \( N \): total number of documents.
  - Weighting scheme:
    \[ W_{ij} = tf_{ij} \times idf_i \]
MRO-MPI for inverted files

• Mappers:
  – \((K_m, V_m) = (\text{term}, (\text{document name}, \text{tf}))\).

• Reducers:
  – Distributes the data based on their lexicographic order, each reducer being responsible for a certain range of words.
  – Similar terms are saved into the same database, reducer nodes can calculate the correct \(\text{tf-idf}\) value.
Distributed inverted files

- 9,319,561 text (XML) excerpts related to 9,319,561 images from 12 million ImageNet corpus.
- Data size: 36GB of XML data.
- Hadoop: 40 minutes with 26 Reducers.
- Double speedup because of sending the data while the map functions is working.
- The best ratio between the mappers and reducers is found to be:

\[ 2M \geq R \geq M. \]

<table>
<thead>
<tr>
<th>Map/Reduce</th>
<th>0</th>
<th>4</th>
<th>13</th>
<th>26</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>81.6</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>-</td>
<td>39.2</td>
<td>46.6</td>
<td>51.7</td>
</tr>
<tr>
<td>10</td>
<td>-</td>
<td>41.5</td>
<td>39.8</td>
<td>39.2</td>
</tr>
<tr>
<td>13</td>
<td>-</td>
<td>66</td>
<td>28.8</td>
<td>24</td>
</tr>
<tr>
<td>20</td>
<td>-</td>
<td>111.9</td>
<td>19.6</td>
<td>18.5</td>
</tr>
<tr>
<td>22</td>
<td>-</td>
<td>113.9</td>
<td>16.45</td>
<td>14.2</td>
</tr>
<tr>
<td>44</td>
<td>-</td>
<td>118.5</td>
<td>125.2</td>
<td>45.8</td>
</tr>
</tbody>
</table>
Conclusion

• We proposed MRO-MPI for intensive data processing.

• Maintain the simplicity of MapReduce.

• High speedup with the same number of nodes.
Questions ?