Parallel implementation of Apriori algorithm and association of mining rules using MPI

Fall 2012
CSE 633- Parallel Algorithms

By,
Sujith Mohan Vellyattikuzhi
50027269
What is Apriori

- An efficient algorithm in data mining to find the undiscovered relationships between different items.

- Operates on databases containing a set of transactions with each transaction having a number of item sets.

- Aims to find the set of “frequent item-sets” and “association rules” between the items.
### Frequent Item Sets

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bread, Milk</td>
</tr>
<tr>
<td>2</td>
<td>Bread, Diaper, Beer, Eggs</td>
</tr>
<tr>
<td>3</td>
<td>Milk, Diaper, Beer, Coke</td>
</tr>
<tr>
<td>4</td>
<td>Bread, Milk, Diaper, Beer</td>
</tr>
<tr>
<td>5</td>
<td>Bread, Milk, Diaper, Coke</td>
</tr>
</tbody>
</table>

- An item set is a collection of one or more items. Eg \(\{\text{Bread, Milk}\}\)
- Those items which occur more frequent.
- In other words, the item sets whose support is greater than the given support.
Support and Support Count

- Support Count is the number of transactions containing the itemsets. Eg – \{Bread, Milk\} = 3
- Support = Support Count/Total num of transactions  
  eg – \{Bread, Milk\} = 3/5 =0.6
- Frequent Item sets are those whose support is greater than or equal to the specified support.
- It can be 1-itemset, 2-itemset… upto n-itemsets, where n, is the total number of items.
Association rules

- Association rules is used for discovering interesting relationships among the items.

- Confidence of an association rule $X \rightarrow Y = \frac{\text{(# of transactions of } X \cup Y)}{\text{(# of transactions of } X)}$

- An association rule is considered to be a strong association rule if its support and confidence are greater than the specified support and confidence.
Rule Generation

Suppose \( \text{min}\_\text{sup}=0.3, \text{min}\_\text{conf}=0.6, \) 
\( \text{Support}\{\text{Beer, Diaper, Milk}\}=0.4 \)

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bread, Milk</td>
</tr>
<tr>
<td>2</td>
<td>Bread, Diaper, Beer, Eggs</td>
</tr>
<tr>
<td>3</td>
<td>Milk, Diaper, Beer, Coke</td>
</tr>
<tr>
<td>4</td>
<td>Bread, Milk, Diaper, Beer</td>
</tr>
<tr>
<td>5</td>
<td>Bread, Milk, Diaper, Coke</td>
</tr>
</tbody>
</table>

All candidate rules:

- \{Beer\} \rightarrow \{Diaper, Milk\} (s=0.4, c=0.67)
- \{Diaper\} \rightarrow \{Beer, Milk\} (s=0.4, c=0.5)
- \{Milk\} \rightarrow \{Beer, Diaper\} (s=0.4, c=0.5)
- \{Beer, Diaper\} \rightarrow \{Milk\} (s=0.4, c=0.67)
- \{Beer, Milk\} \rightarrow \{Diaper\} (s=0.4, c=0.67)
- \{Diaper, Milk\} \rightarrow \{Beer\} (s=0.4, c=0.67)

All non-empty real subsets

\{Beer\}, \{Diaper\}, \{Milk\}, \{Beer, Diaper\}, \{Beer, Milk\}, \{Diaper, Milk\}

Strong rules:

- \{Beer\} \rightarrow \{Diaper, Milk\} (s=0.4, c=0.67)
- \{Beer, Diaper\} \rightarrow \{Milk\} (s=0.4, c=0.67)
- \{Beer, Milk\} \rightarrow \{Diaper\} (s=0.4, c=0.67)
- \{Diaper, Milk\} \rightarrow \{Beer\} (s=0.4, c=0.67)
The Apriori Algorithm

- \( C_k \): Candidate itemset of size \( k \)
- \( L_k \): frequent itemset of size \( k \)

- \( L_1 = \{ \text{frequent items} \} \);
- for \( k = 1; L_k \neq \emptyset; k++ \) do
  - Candidate Generation: \( C_{k+1} = \) candidates generated from \( L_k \);
  - Candidate Counting: for each transaction \( t \) in database do
    increment the count of all candidates in \( C_{k+1} \) that are contained in \( t \)
  - \( L_{k+1} = \) candidates in \( C_{k+1} \) with \( \text{min}\_\text{sup} \)
- return \( \bigcup_k L_k \);
Candidate-generation: Self-joining

- Given $L_k$, how to generate $C_{k+1}$?

**Step 1: self-joining $L_k$**

```sql
INSERT INTO $C_{k+1}$
SELECT $p.item_1, p.item_2, \ldots, p.item_k, q.item_k$
FROM $L_k p, L_k q$
WHERE $p.item_1 = q.item_1, \ldots, p.item_{k-1} = q.item_{k-1}, p.item_k < q.item_k$
```

- **Example**

$L_3 = \{abc, abd, acd, ace, bcd\}$

Self-joining: $L_3 \ast L_3$

- $abcd \leftarrow abc \ast abd$
- $acde \leftarrow acd \ast ace$

$C_4 = \{abcd, acde\}$
Illustrating Apriori Principle

Level 0

null

A
B
C
D
E

Level 1

AB
AC
AD
AE
BC
BD
BE
CD
CE
DE

ABC
ABD
ABE
ACD
ACE
ADE
BCD
BCE
BDE
CDE

ABCD
ABCE
ABDE
ACDE
BCDE

ABCDE

Found to be Infrequent

Pruned Supersets
Output pattern of serial implementation

- Number of transactions = 100 and different items = 200
Interesting patterns in the output

- Output varies according to different inputs of support and confidence and number of item sets present.

- When the support is less, more time is taken to run the program.

- As support increases, the time taken to run the algorithm will be less and gradually comes to constant after a point.
Challenges of apriori algorithm

• More time is taken to generate output for low support values.

• To discover a frequent pattern of size 100, about $2^{100}$ candidates needed to generate.

• Multiple scans of database

• Solution ???? – Parallelize it.
Parallel implementation

• Divide the data sets.

• Each processor $P_i$ will have its own data set $D_i$.

• Each processor $P_i$ reads the values of the data set from a large flat file.

• Each processor do calculation of count of item sets in its own specific processing unit.
How it works

• Support and confidence are given as input to first processor.

• First processor will broadcast the support and confidence to every other processors.

• Each processor generates the first frequent item sets from the input data.

• Then data is divided between different processors.
In subsequent passes

- Each processor $P_i$ develops the complete $C_k$, using the complete frequent itemset $L_{k-1}$ created at the end of pass $k-1$

- Processor $P_i$ develop local support counts for candidates in $C_k$, using its local data partition.

- Then each processor $P_i$ exchanges its local counts to master processor to develop the global $C_k$ counts.
Continued….

• Each processor $P_i$ then computes $L_k$ from $C_k$.

• Each processor $P_i$ independently makes the decision to terminate or continue to next pass.

• The decision will be identical as the processors have all identical $L_k$. 
Flow chart of parallel implementation

1. **START**
2. Input support and confidence to 1st processor
3. Broadcast support and confidence
4. Initially every processor creates 1st level frequent item sets ($L_1$)
5. Data set is divided between different processors
6. If no frequent item set is created
   - Each processor develops local count for each item in candidate set and use All Reduce function to get total of all counts
   - For the next level all processors generate complete candidate set $C_k$ using complete frequent item set $L_{k-1}$
7. **END**
Parallel rule generation

- Generating rules in parallel simply involves partitioning the set of frequent item sets among the processors.
- Each processor generates the rules using the below algorithm
- If a rule Bread, Milk, Coffee -> Diaper does not satisfy the minimum confidence, then no need to consider rules like Bread, Milk -> Coffee, Diaper.
MPI Commands used

• MPI_Comm_rank
• MPI_Bcast
• MPI_AllReduce
• Language used – C++
Test Cases

• Case 1 – To find the output pattern for different values of support and constant number of transactions

• Case 2 – To find the output pattern for different number of transactions with same item sets.

• Case 3 – To find the output pattern for different item sets with same number of transactions.
CASE 1

- Output value for different values of support.
- Both number of item sets and number of transactions are kept constant.
- Various values of support from 40 to 70 are tested.
- Confidence is also kept constant at 50.
Output of parallel implementation

- Number of transactions = 100, different number of items = 200.
- Confidence = 50

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>100</th>
<th>Freq. Items</th>
<th>Assn Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>184.815</td>
<td>139.554</td>
<td>69.596</td>
<td>36.438</td>
<td>19.628</td>
<td>12.244</td>
<td>9.578</td>
<td>7.882</td>
<td>1077</td>
<td>2528</td>
</tr>
<tr>
<td>45</td>
<td>16.1468</td>
<td>10.968</td>
<td>5.42</td>
<td>2.856</td>
<td>1.576</td>
<td>1.11</td>
<td>0.816</td>
<td>0.752</td>
<td>424</td>
<td>614</td>
</tr>
<tr>
<td>50</td>
<td>5.5649</td>
<td>3.762</td>
<td>1.866</td>
<td>0.98</td>
<td>0.552</td>
<td>0.346</td>
<td>0.318</td>
<td>0.282</td>
<td>174</td>
<td>138</td>
</tr>
<tr>
<td>55</td>
<td>0.9734</td>
<td>0.602</td>
<td>0.3</td>
<td>0.162</td>
<td>0.088</td>
<td>0.058</td>
<td>0.058</td>
<td>0.072</td>
<td>64</td>
<td>16</td>
</tr>
<tr>
<td>60</td>
<td>0.345</td>
<td>0.204</td>
<td>0.106</td>
<td>0.052</td>
<td>0.03</td>
<td>0.022</td>
<td>0.012</td>
<td>0.0216</td>
<td>36</td>
<td>4</td>
</tr>
<tr>
<td>65</td>
<td>0.1175</td>
<td>0.0616</td>
<td>0.031</td>
<td>0.015</td>
<td>0.008</td>
<td>0.014</td>
<td>0.016</td>
<td>0.0225</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>70</td>
<td>0.01932</td>
<td>0.01</td>
<td>0.005</td>
<td>0.002</td>
<td>0.001</td>
<td>0.0008</td>
<td>0.008</td>
<td>0.008</td>
<td>7</td>
<td>0</td>
</tr>
</tbody>
</table>
Time taken graph

Time taken in seconds in logarithmic scale vs Number of Processors.
Speedup for parallel implementation

- Number of transactions = 100, different number of items = 200.
- Confidence = 50

<table>
<thead>
<tr>
<th>Number</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>1</td>
<td>1.324</td>
<td>2.655</td>
<td>5.072</td>
<td>9.416</td>
<td>15.094</td>
<td>19.29</td>
<td>23.447</td>
</tr>
<tr>
<td>50</td>
<td>1</td>
<td>1.479</td>
<td>2.982</td>
<td>5.678</td>
<td>10.081</td>
<td>16.084</td>
<td>17.499</td>
<td>19.734</td>
</tr>
<tr>
<td>60</td>
<td>1</td>
<td>1.691</td>
<td>3.254</td>
<td>6.634</td>
<td>11.5</td>
<td>15.68</td>
<td>28.752</td>
<td>15.972</td>
</tr>
<tr>
<td>65</td>
<td>1</td>
<td>1.907</td>
<td>3.79</td>
<td>7.833</td>
<td>14.687</td>
<td>8.392</td>
<td>7.343</td>
<td>5.222</td>
</tr>
<tr>
<td>70</td>
<td>1</td>
<td>1.932</td>
<td>3.864</td>
<td>9.66</td>
<td>19.32</td>
<td>24.15</td>
<td>2.415</td>
<td>2.415</td>
</tr>
</tbody>
</table>
Speed up graph
Findings from case 1

- As the support increases, the time required to solve the problem will decrease.

- As number of processors increase, the time required to solve the problem will decrease and after some processors it becomes constant.

- In case of higher support, the time taken to solve the problem might increase when the number of processors. This is assumed to be due to the large number of communications happening, when compared to the time taken to solve the problem.
CASE 2

- Output for varying number of transactions
- Different transactions from 1000 to 64000 are taken.
- Support is kept at 55 and Confidence is kept at 50.
Output for varying number of transactions

- Support = 55, Confidence = 50

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>11.3971</td>
<td>6.44</td>
<td>3.312</td>
<td>1.618</td>
<td>0.786</td>
<td>0.396</td>
<td>0.224</td>
<td>0.154</td>
</tr>
<tr>
<td>2000</td>
<td>23.4285</td>
<td>13.754</td>
<td>6.78</td>
<td>3.372</td>
<td>1.634</td>
<td>0.78</td>
<td>0.412</td>
<td>0.258</td>
</tr>
<tr>
<td>4000</td>
<td>47.2866</td>
<td>28.63</td>
<td>13.81</td>
<td>6.836</td>
<td>3.37</td>
<td>1.642</td>
<td>0.812</td>
<td>0.4475</td>
</tr>
<tr>
<td>8000</td>
<td>98.411</td>
<td>59.86</td>
<td>28.752</td>
<td>14.2</td>
<td>7.025</td>
<td>3.425</td>
<td>1.6975</td>
<td>0.8675</td>
</tr>
<tr>
<td>16000</td>
<td>197.839</td>
<td>119.81</td>
<td>59.225</td>
<td>29.77</td>
<td>14.656</td>
<td>7.153</td>
<td>3.57</td>
<td>1.79</td>
</tr>
<tr>
<td>32000</td>
<td>402.78</td>
<td>254.96</td>
<td>132.135</td>
<td>66.16</td>
<td>32.205</td>
<td>15.7</td>
<td>7.645</td>
<td>3.815</td>
</tr>
<tr>
<td>64000</td>
<td>812.917</td>
<td>685.84</td>
<td>356.36</td>
<td>179.117</td>
<td>90.892</td>
<td>46.678</td>
<td>24.86</td>
<td>12.75</td>
</tr>
</tbody>
</table>
Time taken graph for different number of transactions

- Time taken in seconds in logarithmic value

- Number of processors
Speed up table for varying number of transactions

- Support = 55, Confidence = 50

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>1.769</td>
<td>3.441</td>
<td>7.043</td>
<td>14.5</td>
<td>28.781</td>
<td>50.5879</td>
<td>74.007</td>
<td></td>
</tr>
<tr>
<td>4000</td>
<td>1.652</td>
<td>3.424</td>
<td>6.917</td>
<td>14.032</td>
<td>28.798</td>
<td>58.234</td>
<td>105.668</td>
<td></td>
</tr>
<tr>
<td>8000</td>
<td>1.644</td>
<td>3.422</td>
<td>6.93</td>
<td>14.008</td>
<td>28.733</td>
<td>57.974</td>
<td>113.442</td>
<td></td>
</tr>
<tr>
<td>16000</td>
<td>1.651</td>
<td>3.34</td>
<td>6.645</td>
<td>13.498</td>
<td>27.658</td>
<td>55.417</td>
<td>110.524</td>
<td></td>
</tr>
<tr>
<td>32000</td>
<td>1.579</td>
<td>3.048</td>
<td>6.087</td>
<td>12.506</td>
<td>25.524</td>
<td>52.685</td>
<td>105.877</td>
<td></td>
</tr>
<tr>
<td>64000</td>
<td>1.185</td>
<td>2.28</td>
<td>4.53</td>
<td>8.943</td>
<td>17.415</td>
<td>32.69</td>
<td>63.758</td>
<td></td>
</tr>
</tbody>
</table>
Speed up graph for varying number of transactions
Findings from case 2

- As the number of transactions increases, the time taken to solve the problem will also increase.
- As the number of processors increases, the time taken to solve the problem decreases and speed up will also increase.
CASE 3

- Output for different number of item sets and with same number of transactions
- Number of transactions is constant and kept at 1000
- Support = 70 and Confidence = 50
- Item sets having values between 200 and 800 are tested.
### Output for varying number of item sets

- Support = 70, Confidence = 50, Number of transactions = 1000

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>200</strong></td>
<td>0.1461</td>
<td>0.076</td>
<td>0.04</td>
<td>0.02</td>
<td>0.01</td>
<td>0.0075</td>
<td>0.008</td>
<td>0.01</td>
</tr>
<tr>
<td><strong>400</strong></td>
<td>1.2112</td>
<td>0.614</td>
<td>0.296</td>
<td>0.148</td>
<td>0.0729</td>
<td>0.038</td>
<td>0.03</td>
<td>0.02</td>
</tr>
<tr>
<td><strong>600</strong></td>
<td>2.086</td>
<td>1.05</td>
<td>0.525</td>
<td>0.265</td>
<td>0.135</td>
<td>0.065</td>
<td>0.045</td>
<td>0.03</td>
</tr>
<tr>
<td><strong>800</strong></td>
<td>21.2171</td>
<td>10.64</td>
<td>5.21</td>
<td>2.596</td>
<td>1.455</td>
<td>0.762</td>
<td>0.456</td>
<td>0.285</td>
</tr>
</tbody>
</table>
Time taken graph for different item sets

- Time Taken in logarithmic scale in seconds
- Number of processors

Graph showing the time taken for different item sets as a function of the number of processors, with the time taken plotted on a logarithmic scale.
Speed up table for varying number of item sets

- Support = 70, Confidence = 50, Number of transactions = 1000

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>1</td>
<td>1.921</td>
<td>3.65</td>
<td>7.3</td>
<td>14.6</td>
<td>18.25</td>
<td>18.25</td>
<td>14.6</td>
</tr>
<tr>
<td>400</td>
<td>1</td>
<td>1.972</td>
<td>4.091</td>
<td>8.182</td>
<td>16.589</td>
<td>31.868</td>
<td>40.366</td>
<td>60.55</td>
</tr>
<tr>
<td>600</td>
<td>1</td>
<td>1.986</td>
<td>3.973</td>
<td>7.87</td>
<td>15.45</td>
<td>32.09</td>
<td>46.35</td>
<td>69.55</td>
</tr>
<tr>
<td>800</td>
<td>1</td>
<td>1.994</td>
<td>4.072</td>
<td>8.174</td>
<td>14.58</td>
<td>27.91</td>
<td>46.64</td>
<td>74.44</td>
</tr>
</tbody>
</table>
Speed up graph for varying number of item sets
Findings from case 3

- As the number of item sets increases, the time taken to run the program will increase.
- Since high value of support is used, the time taken to run the program might increase when number of processors increases.
- This is mainly because of the large amount of communication happening in the program.
Conclusions

- Was able to identify the benefits of parallelizing.
- When number of processors was increased, corresponding reduction in time taken was clearly seen.
- The output depends on the size as well as the type of input data.
Future Work

• To implement apriori algorithm using Open MP and compare its performance with MPI implementation of the same.
References


2. http://www.cse.buffalo.edu/faculty/azhang/cse601/
Thanks!