Multi-Core Map-Reduce Implementation for Efficient Resource Management

Mr. BATHINA NARENDRA KUMAR  
Student  
Dept of Computer Science  
Pragati Engineering College  
Surempalem, East Godavari District, A.P., India

Mr. BANDA S N V RAMANAMURTHY  
Associate Professor  
Dept of Computer Science  
Pragati Engineering College  
Surempalem, East Godavari District, A.P., India

Miss. RAYAVARAPU SRI DIVYA  
Assistant Professor  
Dept of Computer Science  
Pragati Engineering College  
Surempalem, East Godavari District, A.P., India

Abstract: Big data analysis is one of the prominent field and its requirement is also increasing due to internet wide spread e-commerce and e-governance. Most of the cases data is unstructured in nature. For example web logs, transaction logs, images, time series data etc. Analyzing such type of data with single system is not possible in general. So Map-Reduce based process is required to work with such big data streams. This paper proposed a solution which is an extension to the existing work proposed in [1] by utilizing modern multi-core functionality for Map-Reduce tasks. We simulate the project using Remote Method Invocation (RMI) using Java. Optimal way of placement of available jobs by reducing the number of jobs in the waiting queue is the major objective of this paper. This can be achieved if the number of jobs requested as well as number of resources they can be utilized are known prior to execution. But, this model will be failed on dynamic demand. For this purpose a novel framework is required to cater dynamic demands. Proposed system considers this objective also.

Keywords: Map-Reduce, RMI, Big Data, Multi-core functionality, Optimizing Data Grouping and Placement

I. INTRODUCTION

Challenges of Existing System includes, Lack of performance and scalability, Lack of flexible resource management, Lack of application deployment support, Lack of quality of service assurance, Lack of multiple data source support.

Current implementations of the Hadoop MapReduce programming model are not optimized to take full advantage of modern multi-core servers. Application performance and scalability can be further improved by optimizing the placement of tasks on multi-core systems based on the specific nature of the MapReduce workload. This project addresses and solves the issue of flexible resource management by take full advantage of modern multi-core servers. The implementation can allow for both multi-threaded and single-threaded tasks, and be able to schedule them intelligently with a view to maximizing cache effectiveness and data locality into consideration.

II. PROPOSED APPROACH

There is two types of requirements we may consider. First is known demand, dynamic demand. In then known demand, resource requirements and types of files accessed by the users are known prior to the execution of any framework. This is somehow easy to handle. For instance consider a case study belongs to medical report processing. Processing of medical reports of collection of villages includes static data. Once the reports are ready they can be uploaded to the cloud. For that point onwards monitoring of access frequency of each file and types of reports most frequently accessed have to be evaluated. Based on that analysis scheduling of jobs and sharing of resources are possible. Here we have to consider some issues. What is size of each report? What type of data to be analyzed in that? And what are the things we need to deliver from that analysis? If these type of things are understood how many mappers and reducers are required to process this task with in allowable or acceptable amount of time.

Bond Energy Algorithm is one of the solutions proposed in [1] to optimally place the data to various nodes to evenly balance the load according to the transaction logs. But in that single core is utilized to handle MapReduce tasks. Utilization of multi-core functionality is will increase the processing speed in each map/reducer followed by good response time. Upgrade the existing single core algorithms to multi-core algorithms to fully utilize the CPU and increase the overall throughput. Each core is treated as a fork-join task. Instead of checking of single core availability, consider all
core-processors utilization in each mapper/reducer node. Optimal data placement in multi-core environment means allocation of map/reduce tasks to each core on each node rather than single node alone. Collectively all core processors’ time is treated as global process time.

According to the work proposed in [1] generation of HDAG (History Data Access Graph) is not easy with single node. Again HDAG generation itself is a MapReduce Job. But this issue is not raised in that work. If the number of data block or files which are shared by number of tasks then divide the whole set of files into multiple partitions and assign them to number of mapper/reducers jobs proportional to the number of partitions. Here each mapper has to generate a local graph component that can be part final HDAG. At last one reducer has to combine each graph component to produce the final HDAG. One Mapper/reducer means more than one processor but not single processor. So if any task requires 2 nodes according to the conventional MapReduce means one single Mapper is sufficient to handle that task in multi core functionality. In this way, optimization means optimization of data placement as well as optimal and efficient use of available resources. Use this functionality static or known demand can be catered.

**Bond Energy Algorithm [2]**

**Input:** The AA matrix

**Output:** The clustered affinity matrix CA which is a perturbation of AA

**Initialization:** Place and fix one of the columns of AA in CA.

- **Iteration:** Place the remaining \( n-i \) columns in the remaining \( i+1 \) positions in the CA matrix. For each column, choose the placement that makes the most contribution to the global affinity measure.

- **Row order:** Order the rows according to the column ordering.

Best placement? Define contribution of a placement:

\[
cont(A_i, A_k, A_j) = 2 \text{bond}(A_i, A_k) + 2 \text{bond}(A_k, A_j) - 2 \text{bond}(A_i, A_j)
\]

where \( \text{bond}(A_i, A_j) = \sum \text{aff}(A_i, A_j) \text{aff}(A_i, A_j) \)

**Example**

Consider the following AA matrix and the corresponding CA matrix where \( A_1 \) and \( A_2 \) have been placed. Place \( A_3 \):

\[
AA = \begin{bmatrix}
A_1 & A_2 & A_3 \\
45 & 0 & 5 \\
A_2 & 0 & 80 & 5 \\
A_3 & 45 & 5 & 3 \\
A_4 & 0 & 75 & 3 & 78
\end{bmatrix}
\]

\[
CA = \begin{bmatrix}
A_1 & A_2 \\
45 & 0 \\
A_2 & 0 & 80 \\
A_3 & 45 & 5 & 3 \\
A_4 & 0 & 75 & 3 & 78
\end{bmatrix}
\]

Ordering (0-3-1):

\[
cont(A_0, A_3, A_1) = 2 \text{bond}(A_0, A_3) + 2 \text{bond}(A_3, A_1) - 2 \text{bond}(A_0, A_1) = 2 \times 0 + 2 \times 4410 - 2 \times 0 = 8820
\]

Ordering (1-3-2):

\[
cont(A_1, A_3, A_2) = 2 \text{bond}(A_1, A_3) + 2 \text{bond}(A_3, A_2) - 2 \text{bond}(A_1, A_2) = 2 \times 4410 + 2 \times 890 - 2 \times 225 = 10150
\]

Ordering (2-3-4):

\[
cont(A_2, A_3, A_4) = 1780
\]

Ordering (0-3-1):

\[
cont(A_0, A_3, A_1) = 2 \text{bond}(A_0, A_3) + 2 \text{bond}(A_3, A_1) - 2 \text{bond}(A_0, A_1) = 2 \times 0 + 2 \times 4410 - 2 \times 0 = 8820
\]

Ordering (1-3-2):

\[
cont(A_1, A_3, A_2) = 2 \text{bond}(A_1, A_3) + 2 \text{bond}(A_3, A_2) - 2 \text{bond}(A_1, A_2) = 2 \times 4410 + 2 \times 890 - 2 \times 225 = 10150
\]

Ordering (2-3-4):

\[
cont(A_2, A_3, A_4) = 1780
\]

Therefore, the CA matrix has to form:

\[
\begin{bmatrix}
A_1 & A_2 & A_3 \\
45 & 45 & 0 \\
0 & 5 & 80 \\
45 & 53 & 5 \\
0 & 3 & 75
\end{bmatrix}
\]

When \( A_4 \) is placed, the final form of the CA matrix (after row organization) is:

\[
\begin{bmatrix}
A_1 & A_2 & A_3 & A_4 \\
45 & 45 & 0 & 0 \\
0 & 5 & 80 & 3 \\
45 & 53 & 5 & 75 \\
0 & 3 & 75 & 78
\end{bmatrix}
\]
Second objective of this paper is catering of dynamic demand. For this purpose we are proposing a novel and universal solution to the cloud users and service providers. According to this objective data, tasks as well as user requests are all on demand. So resources at one service provider may or may not sufficient enough to handle large scale demand. In that case scalable cloud or cloud elasticity is required. If the resource availability at one service provider is over then the user of that services provider experience the failure or rejection of requests. This situation may not happen for current demands because of lack of awareness, cloud service costs, not fully digitized society etc. But this type of demand surely experienced by both service providers and cloud users in near future. In this type of demand some additional issues are also need to consider to understand our second objective. Some cognitive issues come into picture in general when private data is uploaded to the upload such as security, reliability, service guarantee and service availability. Let us explain each of these issues in detail before moving to proposed solution.

**Security**: Data is of two types sensitive, insensitive. No issue to tackle with insensitive data. For sensitive data such as personal information, medical records, business data etc security is major concern. How can an user upload his/her private data to the cloud? This is fully million dollar question till now. Because securing the cloud and increase the trust on cloud is not a single day or single time process. So considering this aspect is out of scope of this paper.

**Reliability**: This is more important for software-as-a-service model. When a user upload his work and let it be executed at cloud means he is excepting uninterrupted process execution. In case of single node failure may cause the whole work undone. If this situation is happened after 99% of work progress, no one can expect the outcome of the execution.

**Service guarantee and Service Availability**: Assume that one user uploaded his data or request to a service provider. After that service provider dropped or shut down his company. Who will take the responsibility for his data? If demand is more at registered service provider and that provider is unable to cater the current demand so that users of that provider experiencing the service request rejections by the server what will be the impact of this issue on both provider and user. The impact may be in million dollars.

For any one of these issues except security one solution is mutual co-operation. Give and take policy. Service level agreement among cloud service providers is one of the best solutions to handle dynamic demand. If one service provider is experiencing heavy demand then there should be provision to borrow the resources from the other service providers with less demand. Due to this nature cost may little bit increased or may not be increased but users or customers of the cloud cannot experience the service interruption. InterCloud Connectivity is required to handle bulk process tasks. Giant Cloud is required in that case. Each service provider is branch or node under a large cloud tree. It may be visualized as tree but it is a network of clouds means CloudGraph.

### III. EXPERIMENTAL RESULTS

**Table 1 Duration of Turbine Sensor stream execution**:

<table>
<thead>
<tr>
<th>Throughput Stream</th>
<th>Seed</th>
<th>Query Start Time</th>
<th>Query End Time</th>
<th>Duration (sec)</th>
<th>ST1 Start Time</th>
<th>ST1 End Time</th>
<th>ST2 Start Time</th>
<th>ST2 End Time</th>
</tr>
</thead>
</table>
Table 2 Query Execution Time

<table>
<thead>
<tr>
<th>Stream ID</th>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
<th>Q5</th>
<th>Q6</th>
<th>Q7</th>
<th>Q8</th>
<th>Q9</th>
<th>Q10</th>
<th>Q11</th>
<th>Q12</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>97.1</td>
<td>1.9</td>
<td>15.9</td>
<td>3.0</td>
<td>18.8</td>
<td>10.8</td>
<td>14.5</td>
<td>18.8</td>
<td>162.2</td>
<td>11.8</td>
<td>93.8</td>
<td>51.9</td>
</tr>
<tr>
<td>1</td>
<td>485.2</td>
<td>48.2</td>
<td>127.0</td>
<td>30.9</td>
<td>78.6</td>
<td>21.3</td>
<td>136.9</td>
<td>163.5</td>
<td>387.9</td>
<td>63.5</td>
<td>240.9</td>
<td>129.1</td>
</tr>
<tr>
<td>2</td>
<td>601.7</td>
<td>27.6</td>
<td>113.2</td>
<td>45.2</td>
<td>94.2</td>
<td>23.7</td>
<td>56.1</td>
<td>90.8</td>
<td>404.1</td>
<td>66.7</td>
<td>194.8</td>
<td>427.8</td>
</tr>
<tr>
<td>3</td>
<td>508.9</td>
<td>41.6</td>
<td>100.4</td>
<td>48.2</td>
<td>92.9</td>
<td>62.8</td>
<td>125.2</td>
<td>87.9</td>
<td>390.8</td>
<td>60.9</td>
<td>129.5</td>
<td>153.0</td>
</tr>
<tr>
<td>4</td>
<td>551.9</td>
<td>9.6</td>
<td>46.2</td>
<td>37.5</td>
<td>86.7</td>
<td>77.3</td>
<td>63.6</td>
<td>121.2</td>
<td>654.2</td>
<td>78.4</td>
<td>39.8</td>
<td>103.8</td>
</tr>
<tr>
<td>5</td>
<td>527.9</td>
<td>25.9</td>
<td>79.0</td>
<td>59.1</td>
<td>79.6</td>
<td>42.7</td>
<td>101.2</td>
<td>80.3</td>
<td>379.7</td>
<td>59.6</td>
<td>136.7</td>
<td>157.0</td>
</tr>
<tr>
<td>6</td>
<td>597.3</td>
<td>20.5</td>
<td>103.1</td>
<td>36.9</td>
<td>80.3</td>
<td>26.2</td>
<td>150.0</td>
<td>103.1</td>
<td>490.6</td>
<td>52.7</td>
<td>466.9</td>
<td>178.1</td>
</tr>
<tr>
<td>7</td>
<td>513.9</td>
<td>97.1</td>
<td>108.2</td>
<td>38.4</td>
<td>88.7</td>
<td>20.7</td>
<td>90.0</td>
<td>81.9</td>
<td>428.4</td>
<td>61.0</td>
<td>107.8</td>
<td>91.5</td>
</tr>
<tr>
<td>Minimum</td>
<td>97.1</td>
<td>1.9</td>
<td>15.9</td>
<td>8.0</td>
<td>18.8</td>
<td>10.8</td>
<td>14.5</td>
<td>18.8</td>
<td>162.2</td>
<td>11.8</td>
<td>39.8</td>
<td>51.9</td>
</tr>
<tr>
<td>Maximum</td>
<td>601.7</td>
<td>97.1</td>
<td>127.0</td>
<td>59.1</td>
<td>94.2</td>
<td>77.3</td>
<td>150.0</td>
<td>163.5</td>
<td>654.2</td>
<td>78.4</td>
<td>466.9</td>
<td>427.8</td>
</tr>
<tr>
<td>Average</td>
<td>485.5</td>
<td>34.1</td>
<td>86.6</td>
<td>38.0</td>
<td>77.5</td>
<td>35.7</td>
<td>92.2</td>
<td>93.4</td>
<td>412.2</td>
<td>56.8</td>
<td>176.3</td>
<td>159.3</td>
</tr>
</tbody>
</table>

IV. CONCLUSION

Proposed System is two fold. Firstly, this project addresses and solves the issue of flexible resource management by take full advantage of modern multi-core servers. The implementation can allow for both multi-threaded and single-threaded tasks, and be able to schedule them intelligently with a view to maximizing cache effectiveness and data locality into consideration. The default random data placement in a Map-Reduce cluster does not take into account data grouping semantics. Second is catering of dynamic demands with InterCloud Communication with CloudGraph.

V. FUTURE WORK

More attention required to be paid on CloudGraph framework. In depth analysis is required to understand the real time issues, requirements and resources.

VI. REFERENCES


