A Hierarchical Framework for Cross-Domain MapReduce Execution

Yuan Luo¹, Zhenhua Guo¹, Yiming Sun¹,
Beth Plale¹, Judy Qiu¹, Wilfred W. Li²

¹ School of Informatics and Computing, Indiana University
² San Diego Supercomputer Center, University of California, San Diego

ECMLS Workshop of HPDC 2011, San Jose, CA, June 8th 2011
Background

• The MapReduce programming model provides an easy way to execute embarrassingly parallel applications.

• Many data-intensive life science applications fit this programming model and benefit from the scalability that can be delivered using this model.
A MapReduce Application from Life Science: AutoDock Based Virtual Screening

• AutoDock:
  – a suite of automated docking tools for predicting the bound conformations of flexible ligands to macromolecular targets.

• AutoDock based Virtual Screening:
  – Ligand and receptor preparation, etc.
  – A large number of docking processes from multiple targeted ligands
  – Docking processes are data independent

Image source: NBCR
Challenges

• Life Science Applications typically contains large dataset and/or large computation.
• Only small clusters are available for mid-scale scientists.
• Running MapReduce over a collection of clusters is hard
  – Internal nodes of a cluster is not accessible from outside.
Solutions

• Allocating a large Virtual Cluster
  – Pure Cloud Solution

• Coordinating multiple physical/virtual clusters.
  – Physical clusters
  – Physical + Virtual clusters
  – Virtual clusters
Hierarchical MapReduce

Gather computation resources from multiple clusters and run MapReduce jobs across them.
Features

• Map-Reduce-GlobalReduce Programming Model
• Focus on Map-Only and Map-Mostly Jobs
  - map-only, map-mostly, shuffle-mostly, and reduce-mostly *
• Scheduling Policies:
  – Computing Capacity Aware
  – Data Locality Aware (development in progress)

## Programming Model

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>((k^i, v^i))</td>
<td>((k^m, v^m))</td>
</tr>
<tr>
<td>Reduce</td>
<td>((k^m, [v^m_1, ..., v^m_n]))</td>
<td>((k^r, v^r))</td>
</tr>
<tr>
<td>Global Reduce</td>
<td>((k^r, [v^r_1, ..., v^r_n]))</td>
<td>((k^o, v^o))</td>
</tr>
</tbody>
</table>

![Diagram of the Programming Model](image)
Procedures

1) A job is submitted into the system.
2) global controller to local clusters.
3) Intermediate pairs are passed to the Reduce tasks.
4) Local reduce outputs (including new key/value pairs) are send back to the global controller.
5) The Global Reduce task takes key/value pairs from local Reducers, performs the computation, and produces the output.
Computing Capacity Aware Scheduling

- \( Max\text{Mapper}_i = \rho_i \times \text{NumCore}_i \)
  - \( \rho_i \) is defined as maximum numbers of mappers per core.
- \( \gamma_i = Max\text{Mapper}_i - \text{MapperRun}_i \)
  - \( \gamma_i \) is the number of available Mappers on \( \text{Cluster}_i \)
- \( \text{Weight}_i = \frac{\gamma_i \times \theta_i}{\sum_{i=1}^{N} \gamma_i \times \theta_i} \)
  - \( \theta_i \) is the computing power of each cluster;
- \( \text{JobMap}_{x,i} = \text{Weight}_i \times \text{JobMap}_x \)
  - \( \text{JobMap}_{x,i} \) is the number of Map tasks to be scheduled to \( \text{Cluster}_i \) for job \( x \).
MapReduce to run multiple AutoDock instances

1) **Map**: AutoDock binary executable + Python script summarize_result4.py to output the lowest energy result using a constant intermediate key.

2) **Reduce**: Sort the values corresponding to the constant intermediate key by the energy from low to high, and outputs the results.

3) **Global Reduce**: Sorts and combines local clusters outputs into a single file by the energy from low to high.

**AutoDock MapReduce input fields and descriptions**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ligand_name</td>
<td>Name of the ligand</td>
</tr>
<tr>
<td>autodock_exe</td>
<td>Path to AutoDock executable</td>
</tr>
<tr>
<td>input_files</td>
<td>Input files of AutoDock</td>
</tr>
<tr>
<td>output_dir</td>
<td>Output directory of AutoDock</td>
</tr>
<tr>
<td>autodock_parameters</td>
<td>AutoDock parameters</td>
</tr>
<tr>
<td>summarize_exe</td>
<td>Path to summarize script</td>
</tr>
<tr>
<td>summarize_parameters</td>
<td>Summarize script parameters</td>
</tr>
</tbody>
</table>
Experiment Setup

- Cluster Nodes Specifications.
  - FG: FutureGrid, IU: Indiana University

<table>
<thead>
<tr>
<th>Cluster</th>
<th>CPU</th>
<th>Cache size</th>
<th># of Core</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hotel (FG)</td>
<td>Intel Xeon 2.93GHz</td>
<td>8192KB</td>
<td>8</td>
<td>24GB</td>
</tr>
<tr>
<td>Alamo (FG)</td>
<td>Intel Xeon 2.67GHz</td>
<td>8192KB</td>
<td>8</td>
<td>12GB</td>
</tr>
<tr>
<td>Quarry (IU)</td>
<td>Intel Xeon 2.33GHz</td>
<td>6144KB</td>
<td>8</td>
<td>16GB</td>
</tr>
</tbody>
</table>

- PBS allocated 21 nodes per cluster
  - 1 namenode, 20 datanode
- set $\rho_i = 1$ so that
  - $MaxMapper_i = \rho_i \times NumCore_i$
- AutoDock Version 4.2 on each cluster
- 6,000 ligands and 1 receptor.
- $ga\_num\_evals = 2,500,000$
Evaluation

\( \gamma \)-weighted dataset partition:
set \( \theta_i = C \), where \( C \) is a constant, \( \gamma_1 = \gamma_2 = \gamma_3 = 160 \)
\( Weight_i = 1/3 \)

**The average global reduce time taken after processing 6000 map tasks (ligand/receptor docking) is 16 seconds.**
Data Movement cost can be ignored in comparison with the computation cost
Local cluster MapReduce execution time based on different number of map tasks.
\( \gamma\beta \)-weighted dataset partition:

\[ \theta_1 = 2.93 \text{ (Hotel)}, \quad \theta_2 = 2.67 \text{ (Alamo)}, \quad \theta_3 = 2 \text{ (Quarry)} \]

\[ \gamma_1 = \gamma_2 = \gamma_3 = 160 \]

\( \text{Weight}_1 = 0.3860, \quad \text{Weight}_2 = 0.3505, \quad \text{Weight}_3 = 0.2635 \)
Conclusion and Future Work

• A hierarchical MapReduce framework as a solution to run MapReduce over a collection of clusters.
• “Map-Reduce-Global Reduce” model.
• Computing Capacity Aware Scheduling
• AutoDock as an example.
• Performance Evaluation showed the workload are well balanced and the total makespan was kept in minimum.

• Performance Test for Large Dataset Applications.
  – Data transfer overhead
  – Bring Computation to Data
  – Share File System that uses local storage
  – Change $\theta_i$ in the current scheduling policy

• Replace ssh+scp glue
  – Meta-scheduler?
  – Better data movement solution
    • gridftp?
    • Distributed file system?
Acknowledgements

• This work funded in part by
  – Pervasive Technology Institute of Indiana University
  – Microsoft

• Special thanks to Dr. Geoffrey Fox for providing early access to FutureGrid.
Thanks!
Questions?

Yuan Luo, http://www.yuanluo.net
Indiana University School of Informatics and Computing
http://www.soic.indiana.edu
Indiana University Data to Insight Center
http://pti.iu.edu/d2i
Backup Slides

Two-way Data Movement Cost: $\gamma \theta$-weighted Partitioned Input Datasets & Local MapReduce Outputs

Clusters

- Hotel
- Alamo
- Quarry

Data Stage-In

Data Stage-Out

Time (in seconds)

0  3  6  9  12  15  18  21