Design Pattern for Scientific Applications in DryadLINQ CTP

DataCloud-SC11
Hui Li
Yang Ruan, Yuduo Zhou
Judy Qiu, Geoffrey Fox
Motivation

• One of latest release of DryadLINQ was published on Dec 2010
• Investigate usability and performance of using DryadLINQ language to develop data intensive applications
• Generalize programming patterns for scientific applications in DryadLINQ
Big Data Challenge

- Open MPI: Open Source HPC
- Hadoop (MapReduce)
- HBase
- Windows HPC Server 2008
- Windows Azure
- Twitter
- Facebook
- Gmail
- Google
- Peta $10^{15}$
- Tera $10^{12}$
- Giga $10^9$
- Mega $10^6$
MapReduce Processing Model

- Scheduling
- Fault tolerance
- Workload balance
- Communication
Disadvantages in MapReduce

• Rigid and flat data processing paradigm are difficult to express semantic of relational operation, such as Join.

• Pig or Hive can solve above issue to some extent but has several limitations:
  – Relational operations are converted into a set of MapReduce tasks for execution
  – For some equal join, it needs to materialize entire cross product to the disk

• Sample: MapReduce PageRank
Microsoft Dryad Processing Model

Directed Acyclic Graph (DAG)

Processing vertices

Outputs

Channels (file, pipe, shared memory)

Inputs
Microsoft DryadLINQ Programming Model

- Higher level programming interface for Dryad
- Based on LINQ model
- Unified data model
- Integrated into .NET language
- Strong typed .NET objects

<table>
<thead>
<tr>
<th>Common LINQ providers</th>
<th>Base class</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;T&gt;</td>
<td>Strong typed .NET objects</td>
</tr>
<tr>
<td>LINQ-to-objects</td>
<td>IEnumerable&lt;T&gt;</td>
</tr>
<tr>
<td>PLINQ</td>
<td>ParallelQuery&lt;T&gt;</td>
</tr>
<tr>
<td>DryadLINQ</td>
<td>DistributedQuery&lt;T&gt;</td>
</tr>
</tbody>
</table>
Pleasingly Parallel Programming Patterns

Sample applications:
1. SAT problem
2. Parameter sweep
3. Blast, SW-G bio

Issue:
Scheduling resources in the granularity of node rather than core

DryadLINQ

Query

Subquery
User defined function

User defined function

Application Program

Legacy Code
Hybrid Parallel Programming Pattern

Query

DryadLINQ

subquery

User defined function

PLINQ

TPL

Sample applications:
1. Matrix Multiplication
2. GTM and MDS
Solve previous issue by using PLINQ, TPL, Thread Pool technologies
Distributed Grouped Aggregation Pattern

Sample applications:
1. Word count
2. PageRank

Three aggregation approaches:
1. Naïve aggregation
2. Hierarchical aggregation
3. Aggregation tree
## Implementation and Performance

### Hardware configuration

<table>
<thead>
<tr>
<th></th>
<th>TEMPEST</th>
<th>TEMPEST</th>
<th>TEMPEST-CNXX</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPU</strong></td>
<td>Intel E7450</td>
<td>Intel E7450</td>
<td></td>
</tr>
<tr>
<td><strong>Cores</strong></td>
<td>24</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td>24.0 GB</td>
<td>50.0 GB</td>
<td></td>
</tr>
<tr>
<td><strong>Memory/Core</strong></td>
<td>1 GB</td>
<td>2 GB</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>STORM-CN01,CN03</th>
<th>STORM-CN04,CN05</th>
<th>STORM-CN06,CN07</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPU</strong></td>
<td>AMD 2356</td>
<td>AMD 8356</td>
<td>Intel E7450</td>
</tr>
<tr>
<td><strong>Cores</strong></td>
<td>8</td>
<td>16</td>
<td>24</td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td>16 GB</td>
<td>16 GB</td>
<td>48 GB</td>
</tr>
<tr>
<td><strong>Memory/Core</strong></td>
<td>2 GB</td>
<td>1 GB</td>
<td>2 GB</td>
</tr>
</tbody>
</table>

1. We use DryadLINQ CTP version released in December 2010
2. Windows HPC R2 SP2
3. .NET 4.0, Visual Studio 2010
Pairwise sequence comparison using Smith Waterman Gotoh

1. Pleasingly parallel application
2. Easy to program with DryadLINQ
3. Easy to tune task granularity with DryadLINQ API

```
Var SWG_Blocks = create_SWG_Blocks(AluGeneFile, numOfBlocks, BlockSize)
Var inputs= SWG_Blocks.AsDistributedFromPartitions();
Var outputs= inputs.Select(distributedObject => SWG_Program(distributedObject));
```
Workload balance in SW-G

1. SWG tasks are heterogeneous in CPU time.
2. Coarse task granularity brings workload balance issue.
3. Finer task granularity has more scheduling overhead.

Simply solution: tune the task granularity with DryadLINQ API.

Related API:
- AsDistributedFromPartition()
- RangePartition<Type t>()

Fig. 5: CPU and Scheduling Time of the Same SW-G Job with Various Partition Granularities.
Parallel MM algorithms:
1. Row partition
2. Row/Column partition
3. Fox algorithm

Multiple core parallel technologies:
1. PLINQ,
2. TPL,
3. Thread Pool

**Pseudo Code of Fox algorithm:**

Partitioned matrix A, B to blocks

For each iteration i:
1) broadcast matrix A block \((k,j)\) to row \(k\)
2) compute matrix C blocks, and add the partial results to the previous result of matrix C block
3) roll-up matrix B block by column
Matrix multiplication performance results
1core on 16 nodes V.S 24 cores on 16 nodes

![Graph 1](image1.png)

![Graph 2](image2.png)

![Graph 3](image3.png)

![Graph 4](image4.png)
Three distributed grouped aggregation approaches
1. Naïve aggregation
2. Hierarchical aggregation
3. Aggregation Tree

Foreach iteration
{
1. join edges with ranks
2. distribute ranks on edges
3. groupBy edge destination
4. aggregate into ranks
}
PageRank performance results
Naïve aggregation v.s Aggregation Tree

- **Naïve aggregation**
  - Second per Iteration vs Number of AM files
  - Number of Output Tuples vs Second Per Iteration

- **Aggregation Tree**
  - Second per Iteration vs Number of AM files
  - Number of Output Tuples vs Second Per Iteration

- **CPU Usage (%)**
  - Network Usage (Bytes/second)
Conclusion

• We investigated the usability and performance of using DryadLINQ to develop three applications: SW-G, Matrix Multiply, PageRank. And we abstracted three design patterns: pleasingly parallel programming pattern, hybrid parallel programming pattern, distributed aggregation pattern.

• Usability
  – It is easy to tune task granularity with DryadLINQ data model and interfaces.
  – The unified data model and interface enable developers to easily utilize the parallelism of single-core, multi-core and multi-node environments.
  – DryadLINQ provide optimized distributed aggregation approaches, and the choice of approaches have materialized effect on the performance.

• Performance
  – The parallel MM algorithm scale up for large matrices sizes.
  – Distributed aggregation optimization can speed up the program by 30% than naïve approach.
Question?

• Thank you!
Outline

• Introduction
  – Big Data Challenge
  – MapReduce
  – DryadLINQ CTP

• Programming Model
  Using DryadLINQ
  – Pleasingly
  – Hybrid
  – Distributed Grouped Aggregation

• Implementation

• Discussion and Conclusion
Workflow of Dryad job

Window HPC Server 2008 R2 Cluster

Head node
- DSC
- DSC Service
- HPC Job Scheduler Service

Compute node
- Dryad graph manager
- Vertex 1
- Vertex 2
- Vertex n

Data

Workstation computer
- DryadLINQ Provider
- DSC Client Service
- HPC Client Utilities

HPC Client Utilities
- DSC Client Service
- DryadLINQ Provider

Workstation computer