DryadLINQ
A System for General-Purpose Distributed Data-Parallel Computing Using a High-Level Language

Arman Idani

14 Feb 2012
R202 – Data Centric Networking
Background

- Major Distributed Computing Frameworks
  - MapReduce
  - Dryad
  - Apache Hadoop (open source MapReduce)
Motivation

• Internet-scale Services
  • Computationally intensive
  • Huge I/O (terabyte-scale)

• Datacenters
  • Thousands of servers
  • Commodity off-the-shelf hardware
  • They fail
Solution?

- Faster servers
  - Performance not scaling with computational need
  - Memory and I/O limits

- GPUs
  - Tied to underlying hardware implementation
  - Memory and I/O limits

- Parallel databases
  - Designed only for relational algebra manipulations
MapReduce

- Map and Reduce... that’s it.
- No fault tolerance between Map and Reduce
- Reducers write to redundant storage
  - 2 network copies, 3 disk copies
- Architectural limits
  - No support for different types of I/O
- Ugly to program!
Dryad

- Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks (original paper)
- User defines dataflow of the program
Job = Directed Acyclic Graph

- Processing vertices
- Inputs
- Outputs
- Channels (file, pipe, shared memory)
Dryad Architecture
Dryad Properties

• Channel types
  • File transfer, Shared memory FIFO, TCP pipe

• Encapsulation
  • Convert a graph into a vertex for more complicated systems

• Fault tolerance for both vertices and inputs
  • Runs upstream vertices recursively if inputs are gone

• Map and Reduce classes
  • Easy to port MapReduce applications
LINQ

- Language INtegrated Query
  - A set of operators to manipulate datasets in .NET
  - All relational operators are supported
  - Integrated into C#, VB and F#
  - Declarative and Imperative programming
  - .NET development tools
LINQ Architecture

Local machine

Execution engines

Scalability

Multi-core

Single-core

.Net program (C#, VB, F#, etc)

LINQ provider interface

Query

Objects

PLINQ

LINQ-to-SQL

LINQ-to-Obj
Problem: How to easily write distributed data-parallel programs for a computer cluster?

Answer: Give the programmer the illusion of developing for a single computer
  • Let the system deal with parallelism and its complexities
  • Dryad: an execution engine for LINQ
Dryad as LINQ’s execution engine

Local machine

Execution engines

Scalability

Query provider interface

.Net program (C#, VB, F#, etc)

Objects

DryadLINQ

PLINQ

LINQ-to-SQL

LINQ-to-Obj

Cluster

Multi-core

Single-core
DryadLINQ

• Sequential, single machine programming abstraction

• Program runs on single-core, multi-core and a cluster

• Development in familiar programming languages

• Visual Studio development environment
DryadLINQ Overview

Client machine

1. ToDryadTable (.NET)
2. LINQ Expr
3. Compile
4. Vertex code
5. Exec plan
6. Dryad Execution
7. Input tables
8. JM
9. Output DryadTable

Data center

Results

Output Tables
DryadLINQ LINQ Integration

Query

DryadLINQ

Subquery

PLINQ
DryadLINQ SQL Integration

Query

DryadLINQ

Subquery

Subquery

Subquery

Subquery

Subquery

PLINQ

LINQ-to-SQL

LINQ-to-SQL
DryadLINQ Local Simulation

Diagram:
- **Local machine**
- **Query**
- **DryadLINQ**
- **LINQ-to-Object**
  - debug
- **Cluster**
  - production
Evaluation

• Configuration: 240 clusters (8x30)
  • Two dual-core AMD Opteron processors
  • 16GB of DDR2 RAM
  • Four stripped 750GB disks

• Benchmarks
  • TeraSort
  • SkyServer
  • PageRank
  • Machine Learning
TeraSort

- Performance scaling (1 < n < 240)
- Sorting records by string comparisons
- Each node stores 3.87GB

<table>
<thead>
<tr>
<th>Computers</th>
<th>1</th>
<th>2</th>
<th>10</th>
<th>20</th>
<th>40</th>
<th>80</th>
<th>240</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>119</td>
<td>241</td>
<td>242</td>
<td>245</td>
<td>271</td>
<td>294</td>
<td>319</td>
</tr>
<tr>
<td>Data Sorted (GB)</td>
<td>3.87</td>
<td>7.74</td>
<td>38.7</td>
<td>77.4</td>
<td>154.8</td>
<td>309.6</td>
<td>926.4</td>
</tr>
<tr>
<td>GB/s</td>
<td>0.03</td>
<td>0.03</td>
<td>0.16</td>
<td>0.32</td>
<td>0.57</td>
<td>1.16</td>
<td>2.90</td>
</tr>
<tr>
<td></td>
<td>Local</td>
<td>One switch</td>
<td>More than one switch</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SkyServer

- Comparing the location and colour of stars in an astronomical table in Dryad and DryadLINQ
  - Dryad: 1000 lines of code in C++
  - DryadLINQ: 100 lines of code in C#
  - $1 < n < 40$
SkyServer

![Graph showing the speed-up of Dryad Two-pass and DryadLINQ with increasing number of computers.](image-url)
PageRank

• Simple PageRank (iterative hyperlinks counting)
  • Naïve: Links are grouped by source (one Join operation per page)
    • 93 lines of code
    • Scales well
    • 10 iterations in 12,792 seconds
  • Optimized: one Join operation per link (80-90% more local updates)
    • Scales well
    • 10 iterations in 690 seconds
Machine Learning

• Clustering algorithm
  • Parse and re-partition data across the cluster
  • Count the records
  • 10 iterations of E-M algorithm
  • Execution time: 7:11 minutes (5 hours of CPU processing)

• Statistical Inference Algorithm
  • Discover network-wide relationships between hosts and services
  • 4:22 hours (10 days of CPU processing)
DryadLINQ (+)

- Combining LINQ + Dryad
- User defined dataflow
- Stage fault tolerance
- Programming with C#/VB/F#
- Illusions of sequential application development
- Microsoft Visual Studio
- Support for other local LINQ execution engines
- Support for multiple storage systems (NTFS, SQL, Windows Azure, Cosmos DFS)
- .NET libraries
DryadLINQ (-)

• Create the illusion of developing for a single machine

• Dataflow cannot change after initializing
  • Vertices not able to spawn new vertices

• No support for data streaming and pipelining
  • Not suitable for real-time applications

• No support for debugging on the cluster
  • Only local simulation

• Evaluation could be better
Future Work

- Approach the main goal as much as possible:
  - Create the illusion of developing for a single machine

- Developing extensions for DryadLINQ

- Debugging on the cluster and performance debugging

- Reusing previous computed results
  - DryadInc: Reusing work in large-scale computations (2009)