

DryadLINQ

Distributed Computation

Smruti R. Sarangi

Department of Computer Science
Indian Institute of Technology
New Delhi, India

Outline

- 1 Motivation
 - Basic Idea
 - Related Work
- 2 DryadLINQ Design
 - System Architecture
 - LINQ
 - Execution Plan Graph
 - Miscellaneous
- 3 Evaluation
 - Terasort
 - SkyServer

Outline

- 1 Motivation
 - Basic Idea
 - Related Work
- 2 DryadLINQ Design
 - System Architecture
 - LINQ
 - Execution Plan Graph
 - Miscellaneous
- 3 Evaluation
 - Terasort
 - SkyServer

Dryad-LINQ

- Current programming models for large scale distributed programming
 - Map-Reduce
 - MPI
 - Microsoft Dryad
- A DryadLINQ program contains LINQ expressions that:
 - Use LINQ expressions to specify side effect free transformations to data
 - The Dryad system parallelizes portions of the program, and runs the program on thousands of machines
- A sort of a terabyte level data set takes 319 seconds, on a 240 node system.

Basic Idea

LINQ Constructs

- **LINQ** (Language INtegrated Query) is a set of .NET constructs
- It provides support for imperative and declarative programming
- Programming languages supported: C#, F#, VB
- Imperative programming: variables, loops, iterators, conditionals
- Declarative programming: functors, type inferencing

Outline

- 1 Motivation
 - Basic Idea
 - **Related Work**
- 2 DryadLINQ Design
 - System Architecture
 - LINQ
 - Execution Plan Graph
 - Miscellaneous
- 3 Evaluation
 - Terasort
 - SkyServer

Related Work

Parallel Databases

- Implement only declarative variants of SQL Queries
- The query oriented nature of SQL makes it hard to specify typical programming constructs

Map Reduce

- Not very flexible.
- Hard to perform operations such as sorting or database joins
- Lack of type support

Related Work - II

- Domain specific languages on top of MapReduce – Sawzall, Pig (Yahoo), Hive (Facebook)
- They are a combination of declarative constructs, and iterative constructs
- However, they are not very flexible since their pattern is inherently based on SQL
- How is DryadLINQ **different** ?
 - The computation is not dependent on the nature of underlying resources.
 - Uses **virtual execution plans**
 - Underlying computational resources can change dynamically (faults, outages, ...)

Overview of DryadLINQ

Structure of a Dryad job

- It is a directed acyclic graph (DAG)
- Each vertex is a program
- Each edge is a data channel that transmits a finite sequence of records at runtime

Outline

- 1 Motivation
 - Basic Idea
 - Related Work
- 2 **DryadLINQ Design**
 - **System Architecture**
 - LINQ
 - Execution Plan Graph
 - Miscellaneous
- 3 Evaluation
 - Terasort
 - SkyServer

Dryad System Architecture

Dryad System Architecture

It contains a centralized job manager whose role is:

- Instantiating a job's dataflow graph.
- Scheduling processes
- Fault tolerance
- Job monitoring and management
- Transforming the job graph at runtime according to the user's instructions

DryadLINQ Execution Overview

- 1 The user runs a .NET application. It creates a DryadLINQ expression object that has deferred evaluation.
- 2 The application calls the method *ToDryadTable*. This method hands over the expression object to DryadLINQ.
- 3 DryadLINQ compiles the expression, and makes an **execution plan**
 - 1 Decomposition into sub-expressions
 - 2 Generation of code and data for Dryad nodes
 - 3 Generation of serialization and synchronization code.
- 4 Dryad invokes a custom **job manager** .

DryadLINQ Execution Overview - II

- 1 The **job manager** creates a job graph. It schedules and spawns the jobs.
- 2 Each node in the graph executes the program assigned to it.
- 3 When the program is done, it writes the data to the output table.
- 4 After the **job manager** terminates, DryadLINQ collates all the output and creates the DryadTable object.
- 5 Control returns to the user application.
 - 1 Dryad passes an iterator object to the table object.
 - 2 This can be passed to subsequent statements.

Outline

- 1 Motivation
 - Basic Idea
 - Related Work
- 2 DryadLINQ Design
 - System Architecture
 - LINQ
 - Execution Plan Graph
 - Miscellaneous
- 3 Evaluation
 - Terasort
 - SkyServer

LINQ

- The base type is an interface `IEnumerable<T>` – An iterator for a set of objects with type T
 - The programmer is not aware of the data type associated with an instance of `IEnumerable`
- `IQueryable<T>` is a subtype of `IEnumerable<T>`
 - This is an unevaluated expression
 - It undergoes deferred evaluation
 - DryadLINQ creates a concrete class to implement the `IQueryable` expression at runtime

LINQ SQL Syntax Example

```
// Join two tables: scoreTriples and staticRank
var adjustedScoreTriples =
    from d in scoreTriples
    join r in staticRank on d.docID equals r.key
    select new QueryScoreDocIDTriple(d, r);
var rankedQueries =
    from s in adjustedScoreTriples
    group s by s.query into g
    select TakeTopQueryResults(g);
```


LINK OOP Syntax Example

```
var adjustedScoreTriples =  
    scoreTriples.Join(staticRank,  
        d => d.docID, r => r.key,  
        (d, r) => new QueryScoreDocIDTriple(d, r));  
var groupedQueries =  
    adjustedScoreTriples.GroupBy(s => s.query);  
var rankedQueries = groupedQueries.Select(  
    g => TakeTopQueryResults(g));
```

DryadLINQ Constructs

- A DryadLINQ collection (defined by IEnumerable) is a distributed dataset. Partitioning strategies.
 - Hash Partitioning
 - Range Partitioning
 - Round-robin Partitioning
- The results of a DryadLINQ computation are represented by the object – `DryadTable<T>`
 - Subtypes determine the actual storage interface.
 - Can include additional details such as metadata and schemas.

DryadLINQ Methods

- All the methods need to be **side effect free**
- Shared objects can be distributed in any way
- The functions to access a DryadTable are serializable
 - `GetTable<T>`
 - `ToDryadTable<T>`
- Custom partitioning operators
 - `HashPartition<T,K>`
 - `RangePartition<T,K>`
- Functional Operators
 - `apply(f,dataset)` Applies function `f` to all the elements in a dataset
 - `fork(f,dataset)` Similar to `Apply`, but can produce multiple output datasets.
- Dryad annotations – parallelization, storage policies

Outline

- 1 Motivation
 - Basic Idea
 - Related Work
- 2 **DryadLINQ Design**
 - System Architecture
 - LINQ
 - **Execution Plan Graph**
 - Miscellaneous
- 3 Evaluation
 - Terasort
 - SkyServer

System Implementation

Execution Plan Graph (EPG)

- DryadLINQ converts the raw link expressions to the nodes of the EPG
- The EPG is a DAG
- A part of the EPG can also be generated at runtime based on the values of iterative and conditional expressions
- DryadLINQ also needs to respect the metadata (node requirements, and parallelization directives) while generating the EPG
- Needs to support the deferred evaluation of functions

Static Optimizations

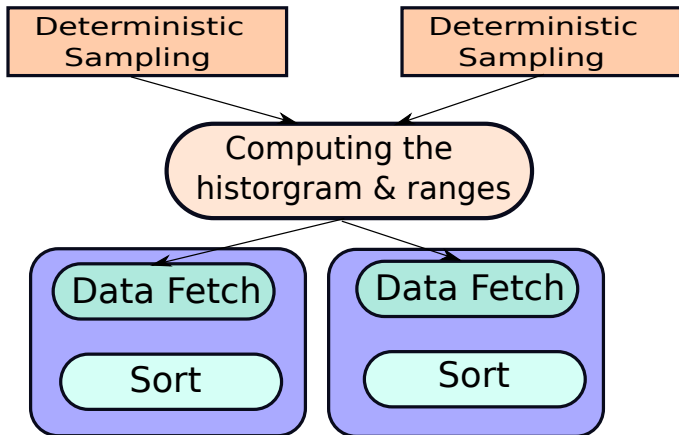
- **Pipelining** : One process executes multiple operations in a pipelined fashion
- **Redundancy Removal** : Remove dead code, and unnecessary partitioning
- **Eager Aggregation** : Intelligently reduce data movement by optimizing aggregation and repartitioning
- **I/O Reduction** : Use TCP pipes, and in-memory channels to reduce persistence to files

Dynamic Optimizations

Optimally Implementing **OrderBy**

- Deterministically sample the values.
- Plot a histogram, and compute the appropriate keys for range partitioning
- A set of vertices now perform the range partitioning.
- A node now fetches the inputs, and then sorts them. These two actions can be pipelined.

Execution Plan of OrderBy



Code Generation

- The EPG is a **virtual execution plan**
- DryadLINQ dynamically generates code for each EPG node
 - DryadLINQ generates a **.NET assembly snippet** that corresponds to each LINQ subexpression.
 - It contains the **serialization and I/O code** for ferrying data.
 - The EPG node code is generated at the computer of the client (**job submitter**), because it may depend on the local context. Values in the local context are embedded in the function/expression. The expression undergoes **partial evaluation** later.
 - Uses **.NET reflection** to find the transitive closure of all .NET libraries. The EPG code, and all the associated libraries are **shipped** to the cluster computer for remote execution.

Outline

- 1 Motivation
 - Basic Idea
 - Related Work
- 2 DryadLINQ Design
 - System Architecture
 - LINQ
 - Execution Plan Graph
 - **Miscellaneous**
- 3 Evaluation
 - Terasort
 - SkyServer

Interacting with other Frameworks

PLINQ

- Runs a subexpression in a cluster node in parallel using multicore processors.
- Uses user supplied annotations (mostly transparent to the user).
- Uses parallel iterators (similar to OpenMP)

SQL

- DryadLINQ nodes can directly access SQL databases.
- They can save internal datasets in SQL tables.
- Can ship some subexpressions to run directly as SQL procedures.

Debugging

Debugging massively parallel applications is very difficult

Debugging

Debugging massively parallel applications is very difficult

Debugging

- Visual Studio .NET interface to debug the DryadLINQ program on a single computer
- DryadLINQ has a deterministic replay model
 - It is possible to replay the entire execution – event by event
 - Secondly, it is possible to replay any subexpression on a local machine and view the outputs
 - Performance Debugging
 - Collect detailed profiling information.

Setup

- 240 computer cluster
- Each node contains two AMD Opteron nodes
- 16GB of main memory
- Experiments

Terasort

- Sort a terabyte size dataset.
- 3.87 GB saved per node.

Outline

- 1 Motivation
 - Basic Idea
 - Related Work
- 2 DryadLINQ Design
 - System Architecture
 - LINQ
 - Execution Plan Graph
 - Miscellaneous
- 3 Evaluation
 - Terasort
 - SkyServer

Results: Terasort - I

- The number of computers was varied from 1 to 250
- The execution time was 120s for 1 machine, and quickly jumped to 250 s.
- Then it grew very **slowly** (sub-linearly) to 320s.

Source [1]

Number of nodes vs Time

Computers	DryadLINQ
1	2666
5	580
10	328
20	176
40	113

Outline

- 1 Motivation
 - Basic Idea
 - Related Work
- 2 DryadLINQ Design
 - System Architecture
 - LINQ
 - Execution Plan Graph
 - Miscellaneous
- 3 Evaluation
 - Terasort
 - SkyServer

SkyServer benchmark

- The number of machines was varied from 1 to 40
- The speedup increased from 1 to 19 **sub-linearly** for DryadLINQ
- The speedup increased from 1 to 24 **sub-linearly** for Dryad Two-pass

Source [1]



DryadLINQ: A System for General-Purpose Distributed Data-Parallel Computing Using a High-Level Language by Yuan Yu, Michael Isard, Dennis Fetterly, Mihai Budiu, Ulfar Erlingsson, Pradeep Kumar Gunda, and Jon Currey, OSDI 2008