Flexible Cluster Computing: Dryad and DryadLINQ

Yuan Yu, Michael Isard, Dennis Fetterly, Mihai Budiu, Úlfar Erlingsson, Pradeep Kumar Gunda, Jon Currey, Andrew Birrell

presented by Michael George

Distributed Storage Systems Seminar
April 2, 2009

Computing on Clusters

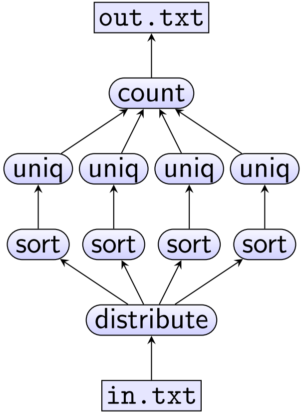
How to crunch lots of data?

- Explicit distribution
  - Write it by hand
  - Hard! failure, resource allocation, scheduling, ...
- Implicit distribution
  - MapReduce, DryadLINQ
  - Easy! As long as your computation is expressible...
- Virtualized distribution
  - Dryad
  - In between. Programmer specifies data flow, system handles details

Dryad Overview — Jobs

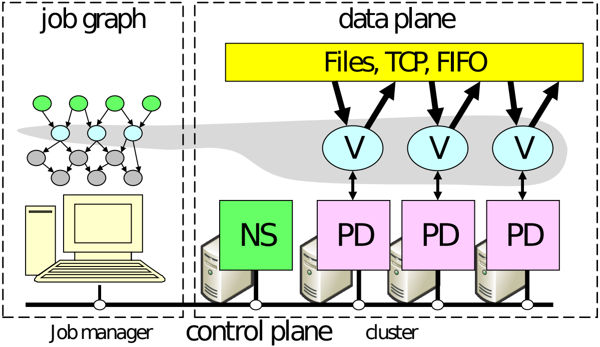
Dryad Job is a Directed Acyclic Graph

- Vertices are subcomputations
- Edges are data channels
- Graph is *virtual* — may be more or fewer vertices than cluster nodes.



Dryad Overview — Execution

Centralized *Job Manager* distributes virtual graph to actual cluster



Writing Vertex Programs

- A Vertex Program is a class that extends the base VP class
  - Base class provides typed I/O channels
  - Specialized abstract subclasses available
    - `Map`, `Reduce`, `Distribute`, ...
- Special support for legacy executables
  - `grep`, `perl`, `legacyApp`, ...
- Asynchronous I/O API available for vertices that require it
  - Runtime distinguishes asynch vertices, executes them efficiently on thread pool

Composing Vertex Programs

Vertex programs are joined into graphs

- edges are local files by default
- can also be TCP pipes or in-memory FIFOs

Composing Vertex Programs

Vertex programs are joined into graphs

- edges are local files by default
- can also be TCP pipes or in-memory FIFOs

Predefined operators for common composition patterns:

- Clone (`G^n`)

Composing Vertex Programs

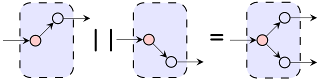
Vertex programs are joined into graphs

- edges are local files by default
- can also be TCP pipes or in-memory FIFOs

Predefined operators for common composition patterns:

- Clone (`G^n`)

- Merge (`G1||G2`)



Composing Vertex Programs

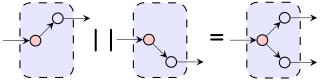
Vertex programs are joined into graphs

- edges are local files by default
- can also be TCP pipes or in-memory FIFOs

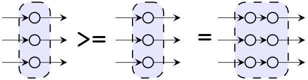
Predefined operators for common composition patterns:

- Clone (`G^n`)

- Merge (`G1||G2`)



- Pointwise composition (`G1>=G2`)



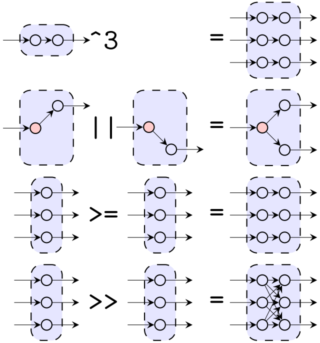
Composing Vertex Programs

Vertex programs are joined into graphs

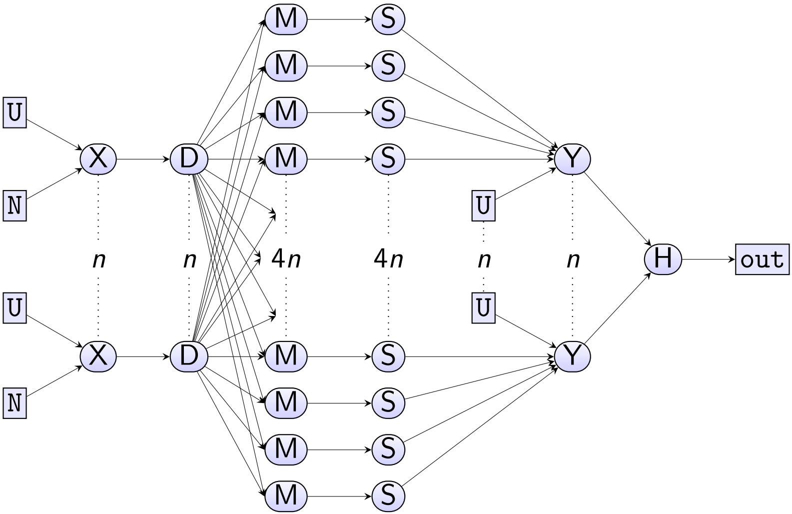
- edges are local files by default
- can also be TCP pipes or in-memory FIFOs

Predefined operators for common composition patterns:

- Clone (`G^n`)
- Merge (`G1||G2`)
- Pointwise composition (`G1>=G2`)
- Bipartite composition (`G1>>G2`)



Example Job




```
((((U >= X) || (N >= X)) >= D)^n >> (((M >= S)^4 >> Y) || U >= Y)^n) >> H >= out
```

Running a Job

Vertices are instantiated on nodes

- May be multiple *execution records* due to failure
- Node placement handled by Job Manager
  - Applications can specify locality "hints" or "requirements"
  - Edge requirements may force vertices to co-locate

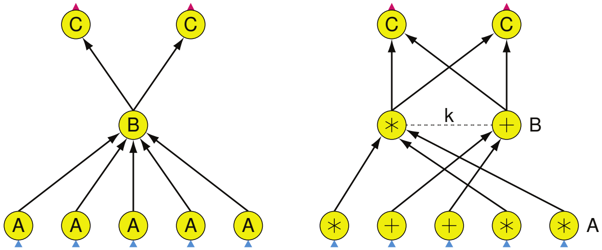
Job manager is notified of node transitions

- May rerun failed vertices
- Can rewrite the graph
- May run duplicate process to route around slow nodes

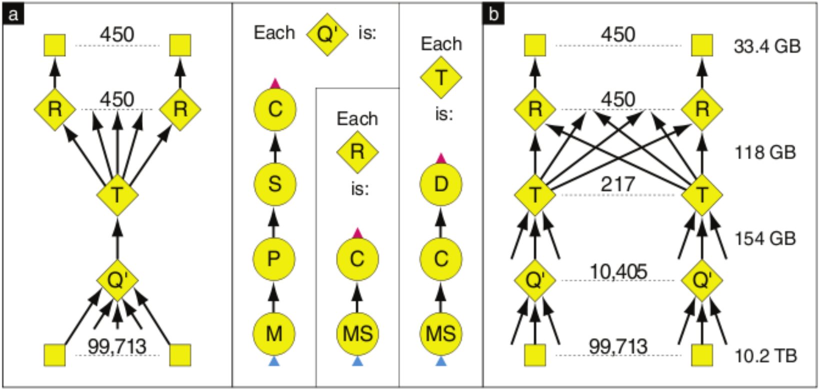
Callback Example: Dynamic Optimization

Client may not know size or distribution of data at load time

- Can dynamically rewrite the graph



MapReduce in Dryad



So Far. . .

- Dryad provides a mid-level execution platform
- Good performance possible
- Flexibility
- But
  - Data parallelization done by hand
  - Optimization done by hand
  - Unfamiliar programming style

DryadLINQ

LINQ (Language INtegrated Query)

- Standard .NET extenstion
- Embeds SQL-like operators into programming languages
- Developers can mix declarative, functional, and imperative statements

DryadLINQ

- Compiles LINQ statements to run on Dryad
- Provides simple, flexible, efficient access to cluster computing

LINQ Example

```
var adjustedScoreTriples =
  from d in scoreTriples
  join r in staticRank on d.docID equals r.key
  select new QueryScoreDecIDTriple(d,r);
var rankedQueries =
  from  s in adjustedScoreTriples
  group s by s.query into g
  select TakeTopQueryResults(g);
```

LINQ Example

```
var adjustedScoreTriples =
  from d in scoreTriples
  join r in staticRank on d.docID equals r.key
  select new QueryScoreDecIDTriple(d,r);
var rankedQueries =
  from  s in adjustedScoreTriples
  group s by s.query into g
  select TakeTopQueryResults(g);
```

```
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

Types:

IEnumerable<T>

IQueryable<T>

DryadTable<T>

NTFS GFS SQL Table

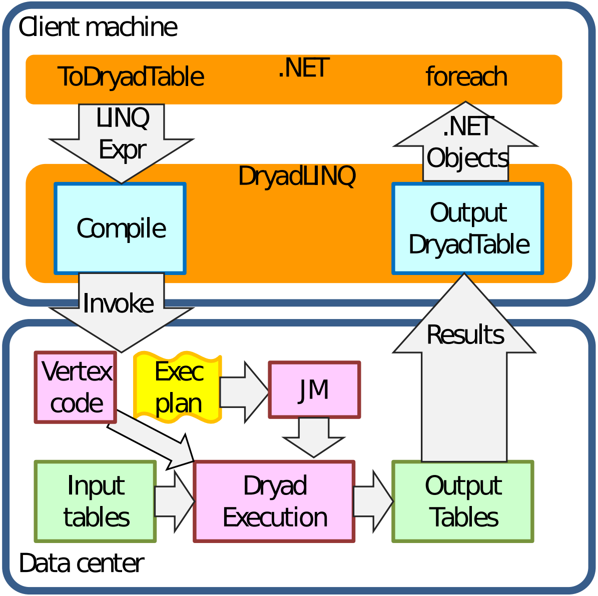
Data Partitioning Operators:

- HashPartition<T,K>
- RangePartition<T,K>

Escape Hatches:

- Apply (f)
- Fork (f)

Execution Overview



Execution Details

1. LINQ expression is compiled to an Execution Plan Graph
   - EPG is a skeleton of a job

Execution Details

1. LINQ expression is compiled to an Execution Plan Graph
   - EPG is a skeleton of a job
2. EPG is optimized using term rewriting
   - Pipelining added
   - Redundancy redundancy removed
   - Aggregation made eager
   - TCP/FIFO annotations added where possible

Execution Details

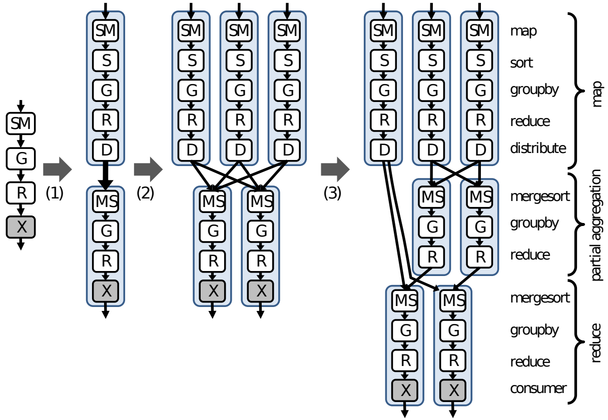
1. LINQ expression is compiled to an Execution Plan Graph
   - EPG is a skeleton of a job
2. EPG is optimized using term rewriting
   - Pipelining added
   - Redundancy redundancy removed
   - Aggregation made eager
   - TCP/FIFO annotations added where possible
3. Code Generated
   - Partially evaluated LINQ subexpressions
   - Serialization code

Execution Details

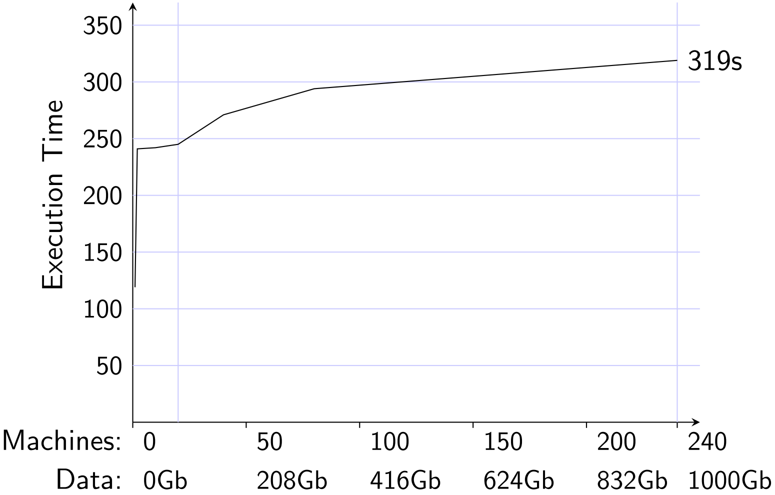
1. LINQ expression is compiled to an Execution Plan Graph
   - EPG is a skeleton of a job
2. EPG is optimized using term rewriting
   - Pipelining added
   - Redundancy redundancy removed
   - Aggregation made eager
   - TCP/FIFO annotations added where possible
3. Code Generated
   - Partially evaluated LINQ subexpressions
   - Serialization code
4. Dynamically, EPG is executed by DryadLINQ job manager
   - vertices replicated to match data
   - dynamic optimizations automated
   - vertices use local LINQ execution engines (e.g. PLINQ)

MapReduce on DryadLINQ

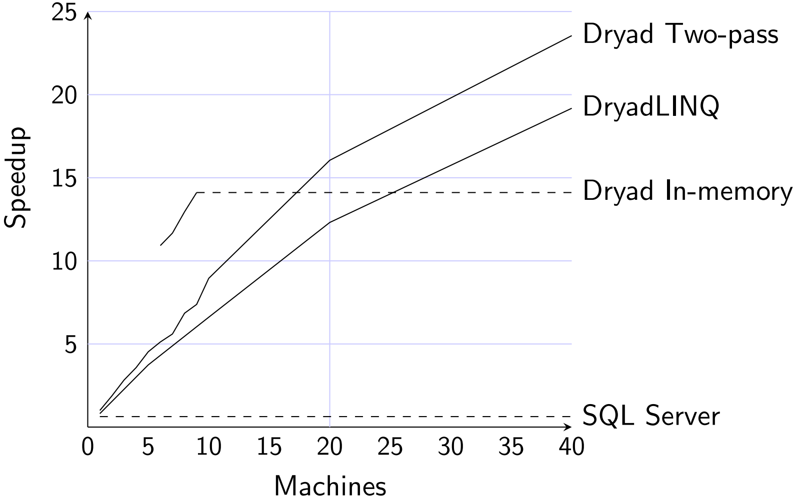
```
public static MapReduce(source, mapper, keySelector, reducer) {
  var mapped = source.selectMany(mapper);
  var groups = mapped.groupBy(keySelector);
  return groups.selectMany(reducer);
}
```



Scalability Evaluation — TeraSort



Overhead Evaluation — SkyServer



Conclusions and Discussion

Dryad meets its goals:

- efficient
- flexible
- programmable

DryadLINQ builds on Dryad:

- almost as efficient
- concise
- familiar