DryadLINQ 11/24/09 -- Lecture 20

What is DryadLINQ?

(Programming language and distributed execution framework for manipulating very large datasets spread across many computers.)

LINQ -- programming language

Dryad -- execution framework

What are the goals of Dryad?

Provide parallel database-style scalability in a more general purpose language.

Users write a single program, don't worry about how it is partitioned across many machines.

Why not SQL?

Want to run arbitrary programs over data. Databases do allow users to write user defined functions and create data types, but this can be awkward at times.

Want better integration with development framework and main (host) language.

Why not MapReduce?

It has a very restricted communication pattern that often requires multiple map/reduce phases to write complex tasks (e.g., joins.) No cross-phase optimization.

What types of programs are they trying to run?
They are not trying to provide transactions, and are focused more on chewing on a bunch of big data files than applying a few small updates, etc.

What is the LINQ programming model?

SQL-like, except that it is embedded in C#, and can make use of C# functions. It is compiled and type checked using C#.

Example:

Ex 1:

```
using System;
using System.Linq;
using System.Collections.Generic;

class app {
  static void Main() {
    string[] names = { "Burke", "Connor", "Frank",
                       "Everett", "Albert", "George",
                       "Harris", "David" };

    IEnumerable<string> query = from s in names
                                where s.Length == 5
                                orderby s
                                select s.ToUpper();

    foreach (string item in query)
      Console.WriteLine(item);
  }
}
```

Ex 2:

```
string[] names = { "Albert", "Burke", "Connor", "David",
                   "Everett", "Frank", "George", "Harris"};

// group by length
var groups = names.GroupBy(s => s.Length, s => s[0]);
foreach (IGrouping<int, char> group in groups) {
    Console.WriteLine("Strings of length {0}", group.Key);

    foreach (char value in group)
        Console.WriteLine("  {0}", value);
}
```

This variation prints the following:

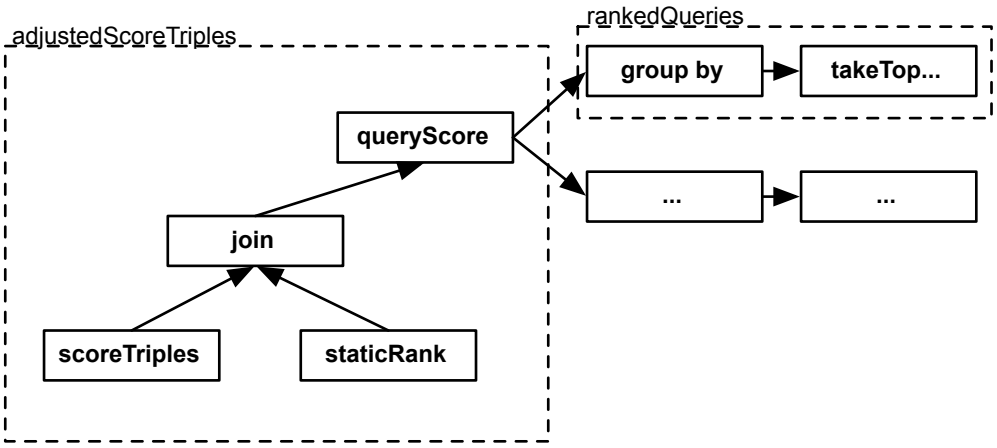
```
Strings of length 6
  A
  C
  G
  H
Strings of length 5
  B
  D
  F
Strings of length 7
  E
```

Ex 3: (In general, you can combine a sequence of operations together)

```
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);
```

Can think of this as a *graph* of operators, just like in SQL (except that SQL plans are mostly trees, whereas here they are DAGs)



Operators:

select
project
order by
join
group by

Why is that good? What are the advantages?

Avoids "Impedence Mismatch" where you run a SQL query, retrieve and typecheck the results, then pack the results back into a string, which you send to the SQL engine, which typechecks the inputs, etc....

Any disadvantages?

Binds you to a specific language; complicates host language; may not support all of SQL.

How does this get distributed in DryadLINQ?
(Show architecture diagram)
Label pieces -- note similarity to BigTable/MapReduce
Note that the output of each operator goes to disk

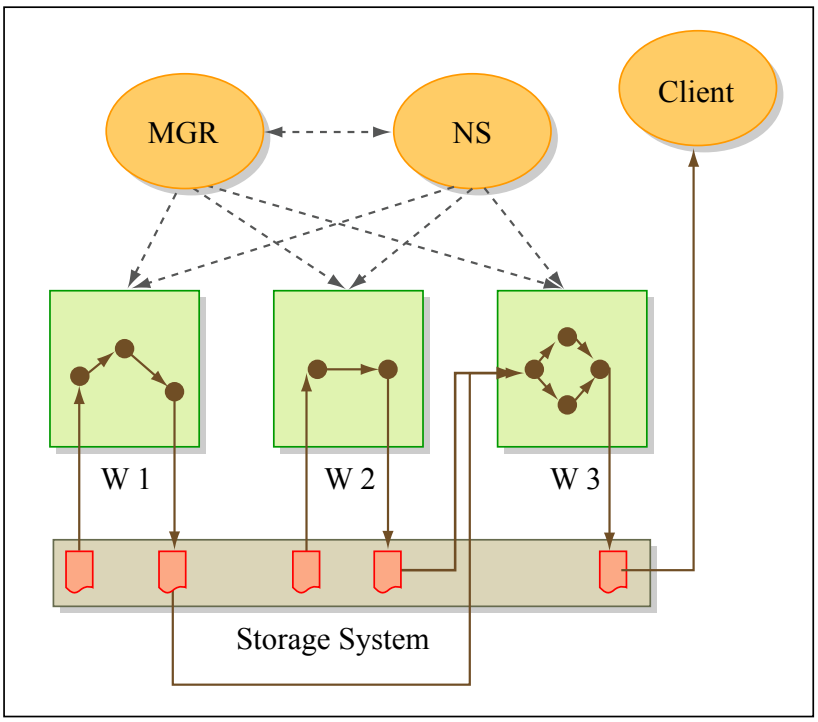
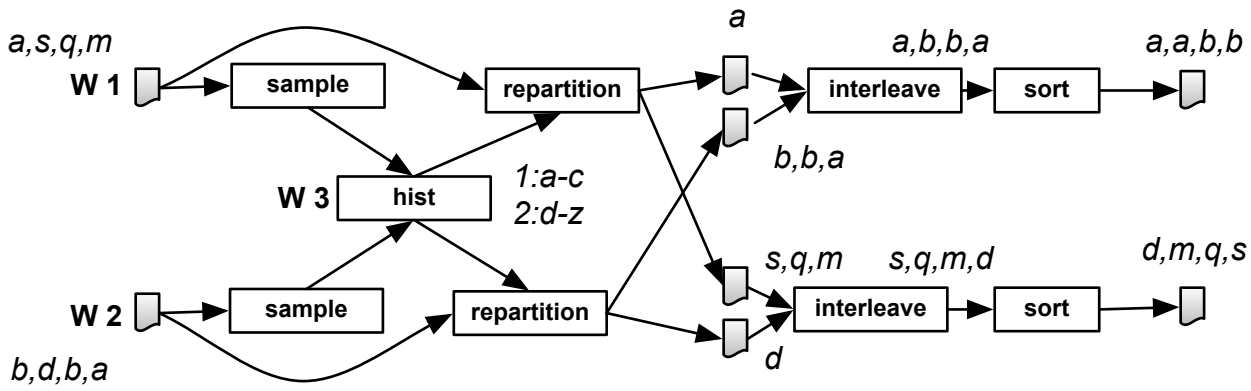


Image by MIT OpenCourseWare.

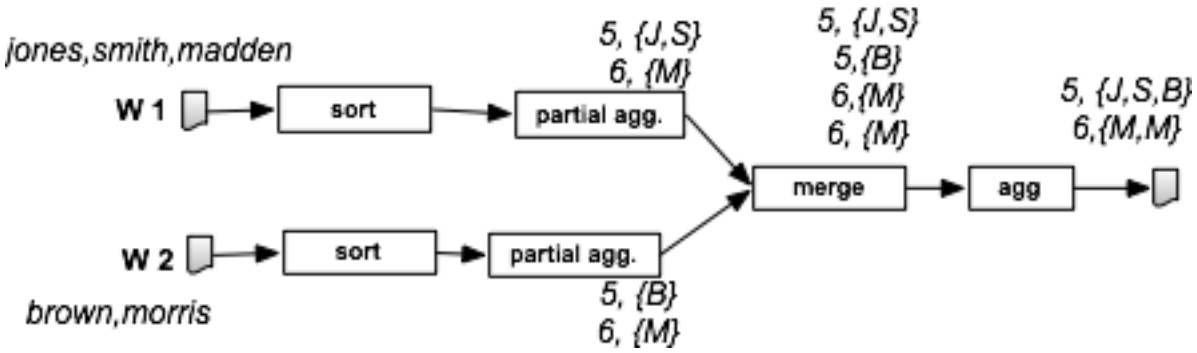
Simple example: Sort



Input data partitioned across nodes, just like in RDBMS

Example 2 -- Aggregate, e.g. Q2

```
var names = GetTable<LineRecord>("file://names.txt")
// group by length
var groups = names.GroupBy(s => s.Length, s => s[0]);
var output = ToDryadTable(groups, "file://out.txt")
```



Very similar to how a DDBMS would do this.

How many partitions to create?

Just say it "depends on the size of the input data"

Are there restrictions on the programs that can be run?

Says that they must be "side effect free"

What does that mean?

No modifications to any shared state.

How restrictive is that?

Can make it tough to do some operations -- for example, you can't write a group by in the language w/out special group by support.
Ex:

```
curgroup = 0
saved = NULL
void aggGbySorted(t, groupf, aggf)
  if (groupf(t) != curgroup && saved != NULL)
        emit aggf(saved)
        saved = new list().append(t)
        curgroup  = groupf(t)
 else
        saved.append(t)
```

(can only auto-parallelie user defined funcs that are stateless)

Can I read shared state? Yes

How is that handled?

Shipped to each processing node

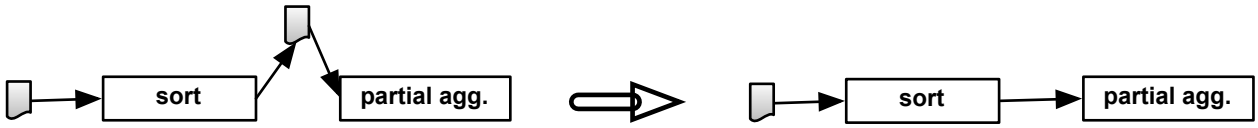
What if I can't figure out how to write my program in the above constraints or constructs?

Apply (f, list) --> Can perform arbitrary computation on the list, but will be run on a single node

What compiler optimizations are supported?

Static (e.g., at compile time)

- Pipeline operators on the same node
Ex: (somewhat unclear how they choose what to combine)



- Eager aggregation -- see aggregate above; perform partial aggregation before repartitioning

- I/O reduction

don't write intermediate data to files --
instead, store in a buffer or send over a TCP pipe when possible

Dynamic (e.g., at run time):

- Dynamic aggregation
  - perform eager aggregation at edges between data centers

- Dynamic partitioning
  - choose number of partitions to create based on the amount of data
    unclear how they do this

- Sort optimizations
    - specific plan for order by -- sample data to determine ranges, compute histogram, split into ranges with equal #s of tuples, redistribute and sort on each node.

Optimizations are very specialized to sepcific programs -- e.g., they seem to have a specific optimization to make orderby, groupby, etc run fast, rather than true general purpose opts.

Lots of talk about how they can 'automatically apply' many different optimizations and infer properties like commutativity, etc., but a bit hard to figure out how these actually work. If you go look at their other papers they have more detail.

In general, this means that programs with lots of custom ops may do many repartitionings -- e.g., after any selection that modifies tuples, repartitioning is needed.

Fault tolerance

Don't really mention it -- though they claim the ability to restart slow nodes as a contribution (kind of an oversight.)

Presumably they do something like what MapReduce does -- if they notice that a particular part of a job fails, they can simply restart/rerun it. Less clear that this works as well as in map reduce where there is a well defined notion of a "job" assigned to one or more nodes.

Debugging, etc.

LINQ programs can compile to other execution frameworks, including a single node thing that makes debugging easy

Performance -- Terasort (3.87 GB / node) (Sec 5.1/Table 1)
1 node takes 119 secs

4 striped disks should read/write sequentially at about ~200 MB/sec

3.87 / .2 = ~20 sec
~40 sec to read and write data; so sorting 3.87 GB is about 80 secs?

> 1 node -- Each node ships n-1/n of data

Data compresses by a factor of 6

Compressed 645 MB
for n=2, sends 320 MB

1 gbit/sec switch ==> 2.5 sec to send

for n = 10, sends 580 MB ==> 4.6 sec to send
(so slight increase from 2-->10 seems to make sense)

So why is sorting so much slower when going from 1-->2?

Sampling? Dunno. Frustrating performance analysis.

6.830 / 6.814 Database Systems

Fall 2010