Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks

Course: CS655

RABIET Louis

Colorado State University

Thursday 19 September 2013
1 Motivation and Goal
   - Why use Dryad?

2 Dryad
   - Graph
   - Example
   - Properties

3 Some others
   - Ants
   - Pegasus

4 An improvement: Ciel
   - Dynamic tasks

5 Comparison

6 Conclusion
Plan

1. Motivation and Goal
   - Why use Dryad?

2. Dryad
   - Graph
   - Example
   - Properties

3. Some others
   - Ants
   - Pegasus

4. An improvement: Ciel
   - Dynamic tasks

5. Comparison

6. Conclusion
### Why use Dryad?

**Parallel Programming**

**Goal**

Make parallel programs easier for developers.
Parallel Programming

Goal
Make parallel programs easier for developers.

Current Choices
- Google’s MapReduce is a popular framework for developing cloud-scale applications.
Parallel Programming

Goal
Make parallel programs easier for developers.

Current Choices

- Google’s MapReduce is a popular framework for developing cloud-scale applications.
- GPU shader languages.
Parallel Programming

Goal
Make parallel programs easier for developers.

Current Choices

- Google’s MapReduce is a popular framework for developing cloud-scale applications.
- GPU shader languages.
- Parallel databases.
Solution and Results

How else can we achieve that?
Using graphs for describing the scheduling/relation between jobs.
Solution and Results

How else can we achieve that?
Using graphs for describing the scheduling/relation between jobs. The system handles the rest i.e. managing execution of stages that compose the graph.
Solution and Results

How else can we achieve that?
Using graphs for describing the scheduling/ relation between jobs. The system handles the rest i.e. managing execution of stages that compose the graph.

Results
Can be performance efficient and simpler.
Plan

1. Motivation and Goal
   - Why use Dryad?

2. Dryad
   - Graph
   - Example
   - Properties

3. Some others
   - Ants
   - Pegasus

4. An improvement: Ciel
   - Dynamic tasks

5. Comparison

6. Conclusion
System overview
Dryad program is a graph

Several operators over them can be specified
Dryad program is a graph

Several operators over them can be specified

- Adding and cloning vertices
Dryad program is a graph

Several operators over them can be specified

- Adding and cloning vertices

\[ AS = A^n \]
**Dryad program is a graph**

Several operators over them can be specified

- Adding and cloning vertices

- Adding edges

![Graph diagram](image-url)
Dryad program is a graph

Several operators over them can be specified

- Adding and cloning vertices

- Adding edges
Dryad program is a graph

Several operators over them can be specified

- Adding and cloning vertices

- Adding edges

- Merging two graphs
Dryad program is a graph

Several operators over them can be specified

- Adding and cloning vertices
  ![Diagram](attachment:image.png)

- Adding edges
  ![Diagram](attachment:image.png)

- Merging two graphs
  ![Diagram](attachment:image.png)
<table>
<thead>
<tr>
<th>Motivation and Goal</th>
<th>Dryad</th>
<th>Some others</th>
<th>An improvement: Ciel</th>
<th>Comparison</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>•</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>••</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>•••</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Graph

- **Merge**
Motivation and Goal

Dryad

Some others

An improvement: Ciel

Comparison

Conclusion

Graph

- **Merge**
<table>
<thead>
<tr>
<th>Motivation and Goal</th>
<th>Dryad</th>
<th>Some others</th>
<th>An improvement: Ciel</th>
<th>Comparison</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Graph

- **Merge**

- **Merge with bypass**

![Graph](image_url)
- Merge
- Merge with bypass
- Merge
- Merge with bypass
- Asymmetric join
**Graph**

- **Merge**
- **Merge with bypass**
- **Asymmetric join**
Code example

```sql
select distinct p.objID
from photoObjAll p
    join neighbors n -- call this join \X"
on p.objID = n.objID
    and n.objID < n.neighborObjID
    and p.mode = 1
join photoObjAll l -- call this join \Y"
on l.objid = n.neighborObjID
    and l.mode = 1
    and abs((p.u-p.g)-(l.u-l.g))<0.05
    and abs((p.g-p.r)-(l.g-l.r))<0.05
    and abs((p.r-p.i)-(l.r-l.i))<0.05
    and abs((p.i-p.z)-(l.i-l.z))<0.05
```
Corresponding graph
Example

**Corresponding vertex program**

```plaintext
GraphBuilder XSet = moduleX^N;
GraphBuilder DSet = moduleD^N;
GraphBuilder MSet = moduleM^(N*4);
GraphBuilder SSet = moduleS^(N*4);
GraphBuilder YSet = moduleY^N;
GraphBuilder HSet = moduleH^1;

GraphBuilder XInputs = (ugriz1 >= XSet) || (neighbor >= XSet);
GraphBuilder YInputs = ugriz2 >= YSet;
GraphBuilder XToY = XSet >= DSet >> MSet >= SSet;
for (i = 0; i < N*4; ++i)
{
    XToY = XToY || (SSet.GetVertex(i) >= YSet.GetVertex(i/4));
}
GraphBuilder YToH = YSet >= HSet;
GraphBuilder HOutputs = HSet >= output;
GraphBuilder final = XInputs || YInputs || XToY || YToH || HOutputs;
```
Some nice properties...

**Channels**

Communication can be chosen between

- Files
- TCP Pipe
- Shared-Memory FIFO
Some nice properties ...

Channels
Communication can be chosen between
- Files
- TCP Pipe
- Shared-Memory FIFO

Support for legacy applications
Node/vertices can execute old code not specifically written for Dryad.
Some nice properties

Runtime aggregation
Some nice properties

Runtime aggregation

Fault tolerance
Restart failed vertex.
Scalability ...
Scalability
# Properties

<table>
<thead>
<tr>
<th>SQL</th>
<th>Dryad In-Memory</th>
<th>Dryad Two-Pass</th>
<th>SQLServer 2005</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed-up</td>
<td>0.0</td>
<td>2.0</td>
<td>4.0</td>
</tr>
<tr>
<td>Number of Computers</td>
<td>0</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

**Motivation and Goal**

- Dryad

**Some others**

**An improvement: Ciel**

**Comparison**

**Conclusion**
Plan

1. Motivation and Goal
   - Why use Dryad?

2. Dryad
   - Graph
   - Example
   - Properties

3. Some others
   - Ants
   - Pegasus

4. An improvement: Ciel
   - Dynamic tasks

5. Comparison

6. Conclusion
Some limitations of Dryad
Some limitations of Dryad

✗ Fault tolerance: what happens if a stage manager fails.
Some limitations of Dryad

✗ Fault tolerance: what happens if a stage manager fails.
✗ No way of defining dynamic graphs.
## Linking ACO and Dryad

- Finding a path between producers and consumers
- Is finding a path sufficient?
- What happens if a node is already close to overutilization?
- **Need a way of evaluating the quality of the path.**
- **Avoid interference between two tasks.**
Motivation and Goal

Dryad

Some others

An improvement: Ciel

Comparison

Conclusion

Ants

Linking ACO and Dryad
Linking ACO and Dryad

- Finding a path between producers and consumers
Linking ACO and Dryad

- Finding a path between producers and consumers

Is finding a path sufficient?
Linking ACO and Dryad

- Finding a path between producers and consumers

Is finding a path sufficient?
What happens if a node is already close to overutilization?
Linking ACO and Dryad

- Finding a path between producers and consumers

Is finding a path sufficient?
What happens if a node is already close to overutilization?

1. Need a way of evaluating the quality of the path.
Linking ACO and Dryad

- Finding a path between producers and consumers

Is finding a path sufficient?

What happens if a node is already close to overutilization?

1. Need a way of evaluating the quality of the path.
2. Avoid interference between two tasks.
## Abstraction of the problem

Each cell is a server in a cell C, we have N(C), the neighbours ΦCD, n: probability if the ant has the destination D that the ant is going to the cell n when currently on C. Routing table: mean and variance of the time to go from C to D.
Abstraction of the problem

- Each cell is a server
Abstraction of the problem

■ Each cell is a server
■ In a cell C, we have
  ■ $N(C)$, the neighbours
  ■ $\Phi^C_{D,n}$: probability if the ant has the destination D that the ant is going to the cell $n$ when currently on $C$.
■ Routing table: mean and variance of the time to go from $C$ to $D$. 
Abstraction of the problem

- Each cell is a server
- In a cell C, we have
  - $N(C)$, the neighbours
  - $\Phi_{C,D,n}$: probability if the ant has the destination D that the ant is going to the cell $n$ when currently on $C$.
  - Routing table: mean and variance of the time to go from $C$ to $D$. 
First type of ant: routing ants

An ant records its path, knows the final destination $D$ and the source $C_s$. At each step, when an ant is on a cell $C$ (different from $D$).
First type of ant: routing ants

An ant records its path, knows the final destination $D$ and the source $C_s$. At each step, when an ant is on a cell $C$ (different from $D$).

1. Record queue waiting time at $C$. 
First type of ant: routing ants

An ant records its path, knows the final destination $D$ and the source $C_s$. At each step, when an ant is on a cell $C$ (different from $D$).

1. Record queue waiting time at $C$.
2. Choose next hop (unknown neighbour first) using $\Phi_{D,n}^C$.
   [Detection of cycles]
First type of ant: routing ants

An ant records its path, knows the final destination $D$ and the source $C_s$. At each step, when an ant is on a cell $C$ (different from $D$).

1. Record queue waiting time at $C$.
2. Choose next hop (unknown neighbour first) using $\Phi_{D,n}^C$.
   [Detection of cycles]
3. En-queue on the link to go to $n$. 
First type of ant: routing ants

An ant records its path, knows the final destination \( D \) and the source \( C_s \). At each step, when an ant is on a cell \( C \) (different from \( D \)).

1. Record queue waiting time at \( C \).
2. Choose next hop (unknown neighbour first) using \( \Phi^C_{D,n} \).
   [Detection of cycles]
3. En-queue on the link to go to \( n \).
4. Record waiting time and transmission time.
First type of ant: routing ants

An ant records its path, knows the final destination \( D \) and the source \( C_s \). At each step, when an ant is on a cell \( C \) (different from \( D \)).

1. Record queue waiting time at \( C \).
2. Choose next hop (unknown neighbour first) using \( \Phi_{D,n}^C \).
   [Detection of cycles]
3. En-queue on the link to go to \( n \).
4. Record waiting time and transmission time.
5. If the ant has not arrived at \( D \), insert itself in the queue, repeat 1-5.
First type of ant: routing ants

An ant records its path, knows the final destination $D$ and the source $C_s$. At each step, when an ant is on a cell $C$ (different from $D$).

1. Record queue waiting time at $C$.
2. Choose next hop (unknown neighbour first) using $\Phi^C_{D,n}$.
   [Detection of cycles]
3. En-queue on the link to go to $n$.
4. Record waiting time and transmission time.
5. If the ant has not arrived at $D$, insert itself in the queue, repeat 1-5.
6. If the ant arrived at $D$, start to go back to $C_s$.
Second type: scouting ants

Try to find alternative path to go from $C_s$ to $D$. 
Second type: scouting ants

Try to find alternative path to go from $C_s$ to $D$.

- If the computing time of the new tasks $L_{new}$ and the computing time of the old tasks $L_{prev}$ does not surpass a threshold $L_T$ (user defined).
Second type: scouting ants

Try to find alternative path to go from $C_s$ to $D$.

- If the computing time of the new tasks $L_{\text{new}}$ and the computing time of the old tasks $L_{\text{prev}}$ does not surpass a threshold $L_T$ (user defined).

- If $L_T$ is not defined, the ant just ensures that the load on the server is inferior to $\alpha$ (random variable). $g$ is an other random variable to determine the number of tasks the ant will place on the server.
Second type: scouting ants

Try to find alternative path to go from $C_s$ to $D$.

- If the computing time of the new tasks $L_{new}$ and the computing time of the old tasks $L_{prev}$ does not surpass a threshold $L_T$ (user defined).
- If $L_T$ is not defined, the ant just ensures that the load on the server is inferior to $\alpha$ (random variable). $g$ is an other random variable to determine the number of tasks the ant will place on the server.
- The system generate scout ants with various $\alpha$ and $g$. If some ants are not able to place all the tasks and have reached $D$ then they will crawl back and modify the $\Phi_{D,n}^C$ (Negative Reinforcement).
Third type: enforcement ants
Third type: enforcement ants

- When enough scout ants are reporting success, some enforcement ants are released.
Third type: enforcement ants

- When enough scout ants are reporting success, some enforcement ants are released.
- Except if extraordinary results are shown by scouts reports, the number of successful scout ants need to be higher than a threshold.
Third type: enforcement ants

- When enough scout ants are reporting success, some enforcement ants are released.
- Except if extraordinary results are shown by scouts reports, the number of successful scout ants need to be higher than a threshold.
- The enforcement ants recompute the cost (and compare it with the scout results); if everything is fine the placement is done.
Some drawbacks
### Some drawbacks

× Evaluation is done via simulation
Some drawbacks

✗ Evaluation is done via simulation
✗ No theoretical proof of correctness for placement algorithm
Some drawbacks

- Evaluation is done via simulation
- No theoretical proof of correctness for placement algorithm
- Very few optimizations → what about overhead costs?
<table>
<thead>
<tr>
<th>Motivation and Goal</th>
<th>Dryad</th>
<th>Some others</th>
<th>An improvement: Ciel</th>
<th>Comparison</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
</tbody>
</table>

Pegasus

Diagram:

- **Head node**
- **jobmanager**
- **GridFTP Server**
- **Storage System**
- **Resource**
- **LRC**
- **MDS local**
- **MDS Index**
- **RLI**

Information flow:
- From **MDS Index** to **jobmanager**
- From **jobmanager** to **GridFTP Server**
- From **GridFTP Server** to **Resource**
Scheduling horizon
Plan

1. Motivation and Goal
   - Why use Dryad?

2. Dryad
   - Graph
   - Example
   - Properties

3. Some others
   - Ants
   - Pegasus

4. An improvement: Ciel
   - Dynamic tasks

5. Comparison

6. Conclusion
**Control Flow**

![Control Flow Diagram]

<table>
<thead>
<tr>
<th>Task ID</th>
<th>Dependencies</th>
<th>Expected outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>{ u }</td>
<td>_/</td>
</tr>
<tr>
<td>B</td>
<td>{ v }</td>
<td>x</td>
</tr>
<tr>
<td>C</td>
<td>{ w }</td>
<td>y</td>
</tr>
<tr>
<td>D</td>
<td>{ x, y }</td>
<td>z</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Object ID</th>
<th>Produced by</th>
<th>Locations</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td>-</td>
<td>host19, host85</td>
</tr>
<tr>
<td>v</td>
<td>-</td>
<td>host21, host23</td>
</tr>
<tr>
<td>w</td>
<td>-</td>
<td>host22, host57</td>
</tr>
<tr>
<td>x</td>
<td>B</td>
<td>_</td>
</tr>
<tr>
<td>y</td>
<td>C</td>
<td>_</td>
</tr>
<tr>
<td>z</td>
<td>A, D</td>
<td>_</td>
</tr>
</tbody>
</table>
Publishing: how to handle dynamic graphs
Lazy evaluation

Lazy evaluation is used to schedule task.
Lazy evaluation

Lazy evaluation is used to schedule task.

1. Start by the (expected) output of the root task
Lazy evaluation

Lazy evaluation is used to schedule task.

1. Start by the (expected) output of the root task
2. If T has only concrete dependencies, execute it immediately
Lazy evaluation

Lazy evaluation is used to schedule task.

1. Start by the (expected) output of the root task
2. If $T$ has only concrete dependencies, execute it immediately
3. Else block $T$ until the input becomes concrete.
An optimization

\[
\text{result} = \text{spawn_exec(executor, args, n)};
\]

**Unique Naming**

for the \(i^{th}\) output: \(\text{name} = \text{executor:SHA1(arg}\|\text{n}):i\)

**Memoisation**

If a new task’s outputs have already been produced by a previous task, the new task need not be executed at all.
### Master

Ciel support master failure
Persistent logging

- Write when a new job is launched
Persistent logging

- Write when a new job is launched
- The descriptors (graphs) are periodically written to the disk.
Persistent logging

- Write when a new job is launched
- The descriptors (graphs) are periodically written to the disk.
- If the master reboots, it looks for unmatched jobs. They are then restarted (the graph has been stored).
Secondary masters
Secondary masters

- Everything is sent to every master (primary and secondaries).
Secondary masters

- Everything is sent to every master (primary and secondaries).
- Every secondary records workers address.
Object table reconstruction

In a case of a master rebooting (failure then recovering):
Object table reconstruction

In a case of a master rebooting (failure then recovering):

1. Workers detect master failure (heartbeat).
Object table reconstruction

In a case of a master rebooting (failure then recovering):

1. Workers detect master failure (heartbeat).
2. If a failure is detected, workers send registration to the master.
Object table reconstruction

In a case of a master rebooting (failure then recovering):

1. Workers detect master failure (heartbeat).
2. If a failure is detected, workers send registration to the master.
3. Master resends objects list.
Speed comparison
Plan

1 Motivation and Goal
   ■ Why use Dryad?

2 Dryad
   ■ Graph
   ■ Example
   ■ Properties

3 Some others
   ■ Ants
   ■ Pegasus

4 An improvement: Ciel
   ■ Dynamic tasks

5 Comparison

6 Conclusion
Map Reduce
Map Reduce

✓ Simple
✓ Efficient for a specific kind of programs: web indexing
Map Reduce

✓ Simple
✓ Efficient for a specific kind of programs: web indexing
✗ Reliability very simple
Map Reduce

- Simple
- Efficient for a specific kind of programs: web indexing
- Reliability very simple
- Workflow hardcoded
Map Reduce

✓ Simple
✓ Efficient for a specific kind of programs: web indexing
✗ Reliability very simple
✗ Workflow hardcoded
✗ New?
Map Reduce

✓ Simple
✓ Efficient for a specific kind of programs: web indexing
✗ Reliability very simple
✗ Workflow hardcoded
✗ New? PL-SQL, Teradata.
Map Reduce

✔ Simple
✔ Efficient for a specific kind of programs: web indexing
✘ Reliability very simple
✘ Workflow hardcoded
✘ New ? PL-SQL, Teradata.

Improvement

- Instead of having a Map phase then a Reduce phase: do a combination of Map-Reduce.
Map Reduce

✓ Simple
✓ Efficient for a specific kind of programs: web indexing
✗ Reliability very simple
✗ Workflow hardcoded
✗ New ? PL-SQL, Teradata.

Improvement

- Instead of having a Map phase then a Reduce phase: do a combination of Map-Reduce.
- Computation Reuse!
Dryad

Some optimizations: aggregation (data locality).

Much more general than Map Reduce.

Not so simple (look at the code).

No cycles.

Coming from SQL.

Improvement:

Ciel

Comparison:

Conclusion:
### Dryad

- Some optimizations: aggregation (data locality).

---
Dryad

✓ Some optimizations: aggregation (data locality).
✓ Much more general than Map Reduce
Dryad

- Some optimizations: aggregation (data locality).
- Much more general than Map Reduce
- Not so simple (look at the code)
Dryad

- Some optimizations: aggregation (data locality).
- Much more general than Map Reduce
- Not so simple (look at the code)
- No cycles
Dryad

- Some optimizations: aggregation (data locality).
- Much more general than Map Reduce
- Not so simple (look at the code)
- No cycles
- Coming from SQL.
Dryad

- Some optimizations: aggregation (data locality).
- Much more general than Map Reduce
- Not so simple (look at the code)
- No cycles
- Coming from SQL.

Improvement
Ciel
Ants
Ants

✓ Analogy with nature.
Ants

- Analogy with nature.
- Distributed
**Ants**

- ✔ Analogy with nature.
- ✔ Distributed
- ✔ General
Ants

✓ Analogy with nature.
✓ Distributed
✓ General
✓ Account for machine load
Ants

- Analogy with nature.
- Distributed
- General
- Account for machine load
- Difficult to predict behaviour
Ants

- ✓ Analogy with nature.
- ✓ Distributed
- ✓ General
- ✓ Account for machine load
- × Difficult to predict behaviour
- × Overhead !!
Ants

- Analogy with nature.
- Distributed
- General
- Account for machine load
- Difficult to predict behaviour
- Overhead !!
Ants

✓ Analogy with nature.
✓ Distributed
✓ General
✓ Account for machine load
✗ Difficult to predict behaviour
✗ Overhead !!

Improvement
Ants

- Analogy with nature.
- Distributed
- General
- Account for machine load
- Difficult to predict behaviour
- Overhead !!

Improvement

- What happens if an ant is faulty?
Ants

✓ Analogy with nature.
✓ Distributed
✓ General
✓ Account for machine load
✗ Difficult to predict behaviour
✗ Overhead !!

Improvement

- What happens if an ant is faulty?
- I would replace it with a simpler search of optimal paths.
Pegasus
Pegasus

✓ Performance issue
Pegasus

- ✔ Performance issue
- ✗ Not general enough
<table>
<thead>
<tr>
<th>Motivation and Goal</th>
<th>Dryad</th>
<th>Some others</th>
<th>An improvement: Ciel</th>
<th>Comparison</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Pegasus**

✅ Performance issue

✗ Not general enough
Pegasus

- Performance issue
- Not general enough

**Improvement**

Reliability (only partition level)
<table>
<thead>
<tr>
<th>Motivation and Goal</th>
<th>Dryad</th>
<th>Some others</th>
<th>An improvement: Ciel</th>
<th>Comparison</th>
<th>Conclusion</th>
</tr>
</thead>
</table>

**Ciel**

Much more general.

Dynamic task dependencies

Improvement

Random functions

How far are you from machine performance?
Ciel

✓ Much more general.
Motivation and Goal

Dryad

Some others

An improvement: Ciel

Comparison

Conclusion

Ciel

✓ Much more general.

✓ Dynamic task dependencies
Ciel

- Much more general.
- Dynamic task dependencies
## Ciel

- ✓ Much more general.
- ✓ Dynamic task dependencies

### Improvement

- Random functions
Ciel

✓ Much more general.
✓ Dynamic task dependencies

Improvement

■ Random funtions
■ How far are you from machine performance?
Perpectives
Perspectives

- Adding user’s transformations for graphs: tiling, merge, etc...
Perspectives

- Adding user’s transformations for graphs: tiling, merge, etc...
- BGP (distance-vector routing protocol)
Perspectives

- Adding user’s transformations for graphs: tiling, merge, etc...
- BGP (distance-vector routing protocol)
- Look at this problem using maximum flow problem.
## Plan

1. **Motivation and Goal**
   - Why use Dryad?

2. **Dryad**
   - Graph
   - Example
   - Properties

3. **Some others**
   - Ants
   - Pegasus

4. **An improvement: Ciel**
   - Dynamic tasks

5. **Comparison**

6. **Conclusion**
Conclusion

Microsoft decided to stop developing Dryad for commercial reasons.
Conclusion

Summary

Several restrictions that make the result possible:

- Node results are deterministic
- Job manager doesn’t fail
- Acyclic graph
- Doesn’t add enough to MapReduce