Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks

Course: CS655

Rabiet Louis

Colorado State University

Thursday 19 September 2013

Plan

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 esle can we achieve that ?

Using graphs for describing the scheduling/ relation between jobs.

Solution and Results

How esle 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 esle can we achieve that ?

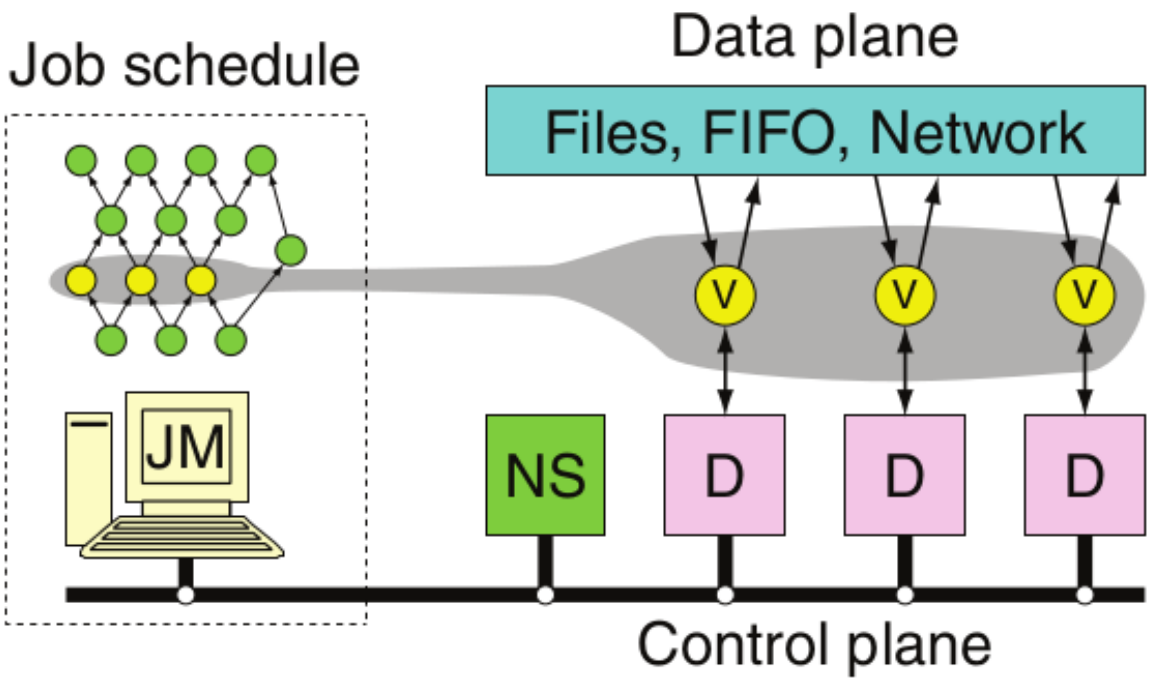
Using graphs for describing the scheduling/ relation between jobs. The system handles the rest i.e. managing execution of stages that compose the graph.

Results

Can be performance efficient and simpler.

Plan

System overview



Dryad program is a graph

Several operators over them can be specified

Dryad program is a graph

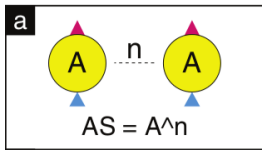
Several operators over them can be specified

- Adding and cloning vertices

Dryad program is a graph

Several operators over them can be specified

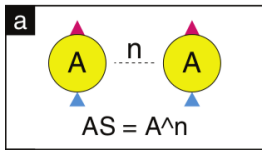
- Adding and cloning vertices



Dryad program is a graph

Several operators over them can be specified

- Adding and cloning vertices

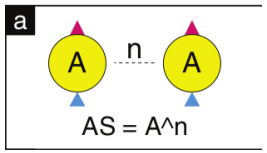


- Adding edges

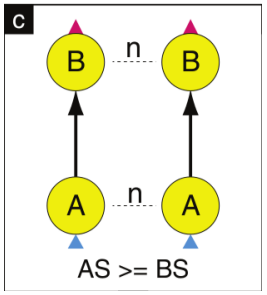
Dryad program is a graph

Several operators over them can be specified

- Adding and cloning vertices



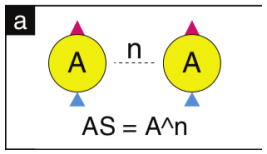
- Adding edges



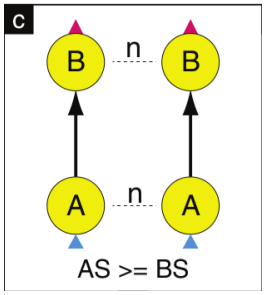
Dryad program is a graph

Several operators over them can be specified

- Adding and cloning vertices



- Adding edges

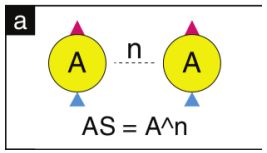


- Merging two graphs

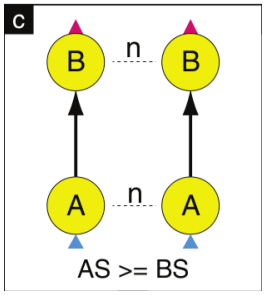
Dryad program is a graph

Several operators over them can be specified

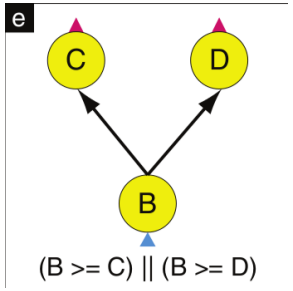
- Adding and cloning vertices



- Adding edges



- Merging two graphs

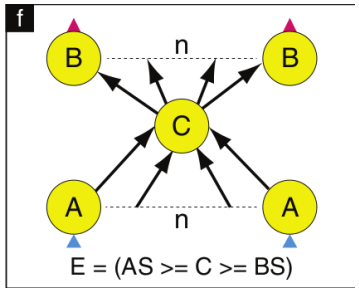


Graph

- Merge

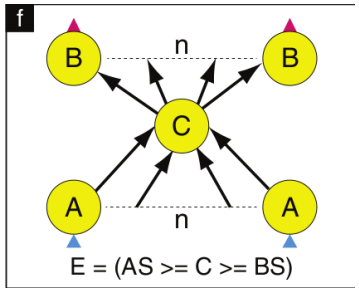
Graph

- Merge



Graph

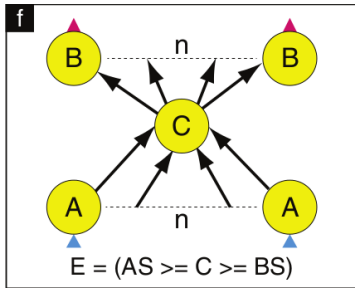
- Merge



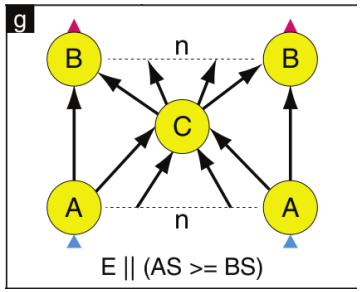
- Merge with bypass

Graph

- Merge

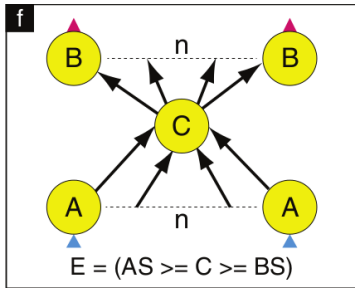


- Merge with bypass

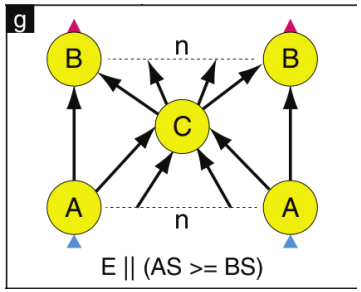


Graph

- Merge



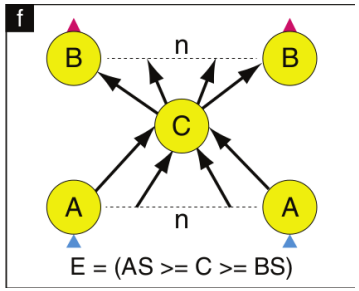
- Merge with bypass



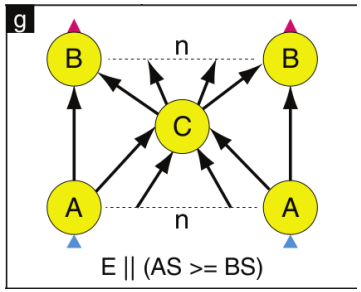
- Asymmetric join

Graph

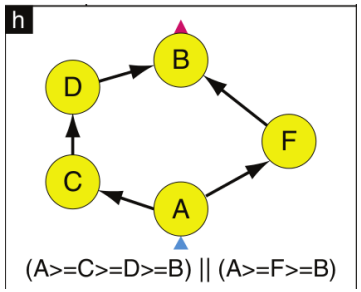
- Merge



- Merge with bypass



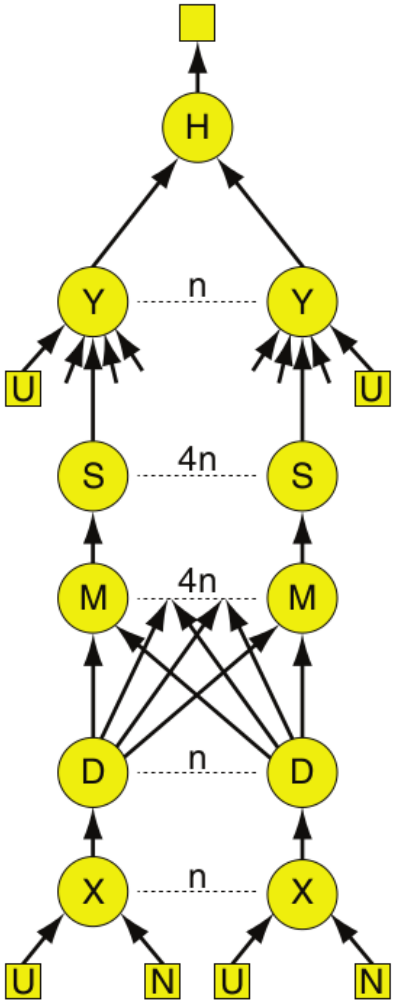
- Asymmetric join



Code example

```
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



Corresponding vertex program

```
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

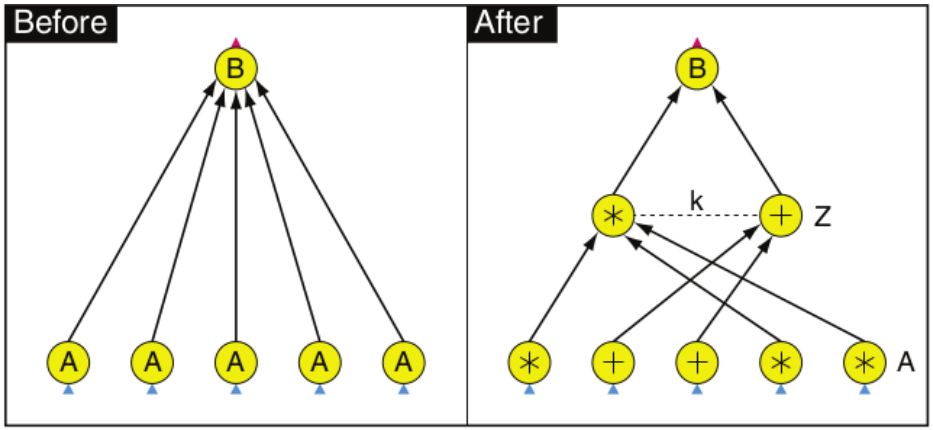
- Files
- TCP Pipe
- Shared-Memory FIFO

Support for legacy applications

Node/vertices can execute old code not specifically written for Dryad.

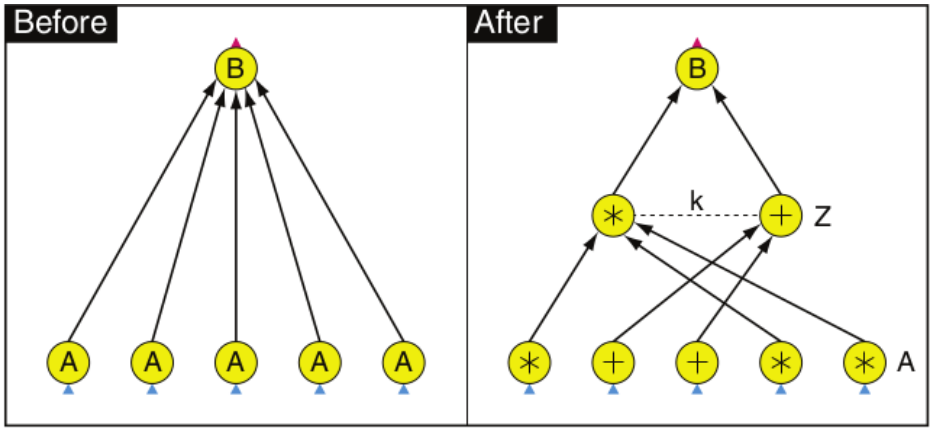
Some nice properties

Runtime aggregation



Some nice properties

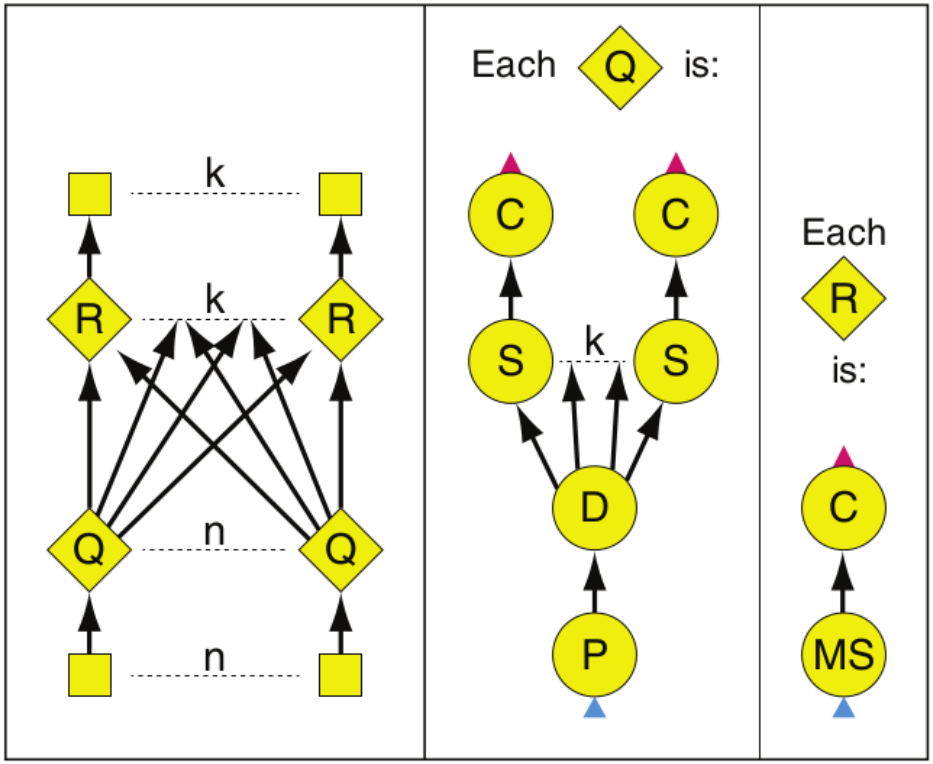
Runtime aggregation



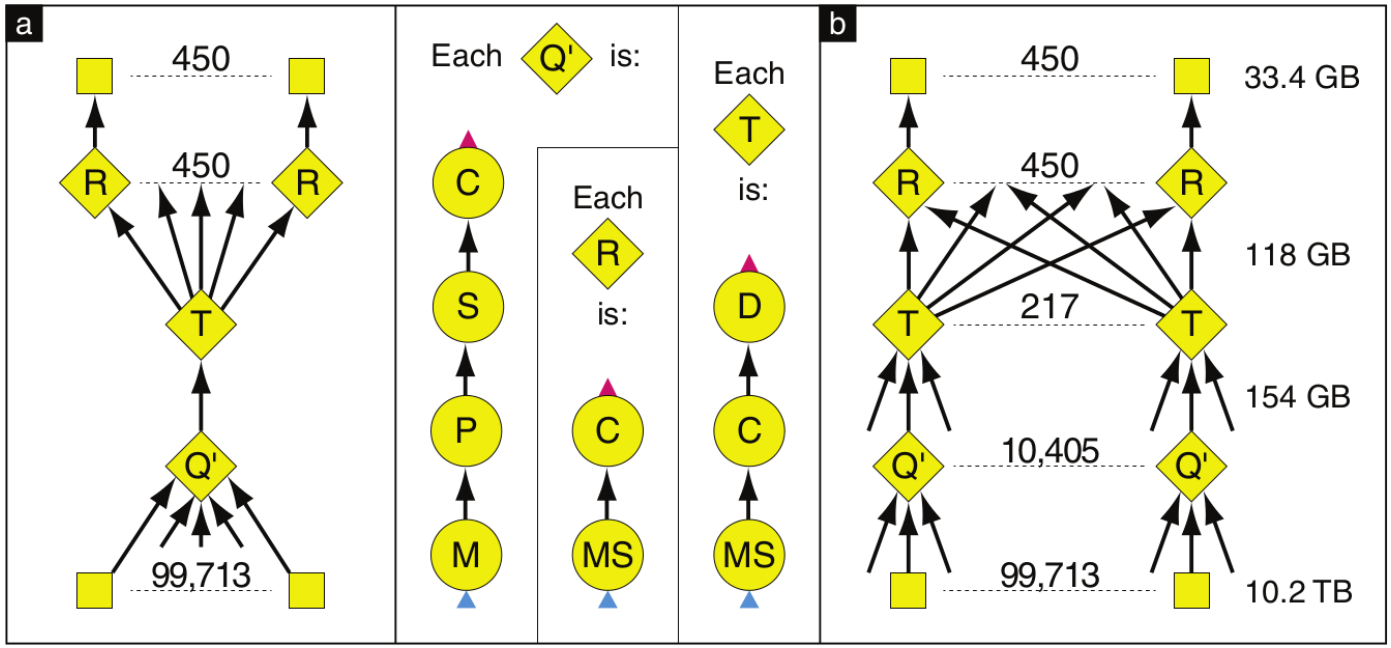
Fault tolerance

Restart failed vertex.

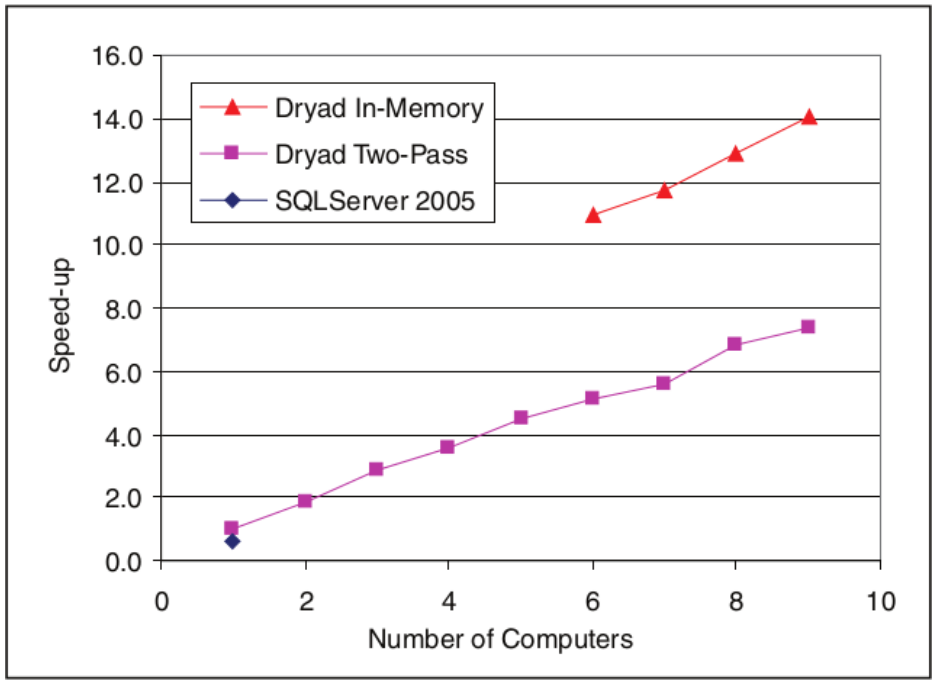
Scalability ...



Scalability



SQL


16.0
14.0
12.0
10.0
8.0
6.0
4.0
2.0
0.0
Speed-up
0
2
4
6
8
10
Number of Computers
Dryad In-Memory
Dryad Two-Pass
SQLServer 2005

Plan

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

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

Abstraction of the problem

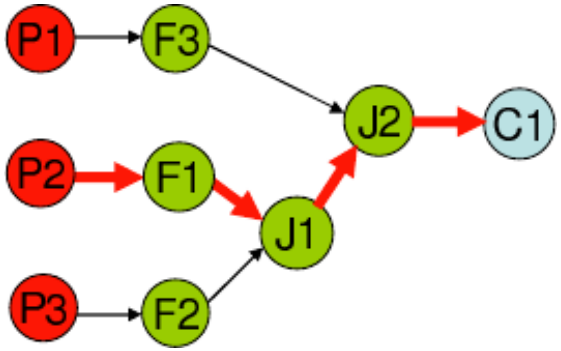
- Each cell is a server

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 $\Phi^{C}_{D,n}$. [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 $\Phi^{C}_{D,n}$. [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 $\Phi^{C}_{D,n}$. [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.

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_{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 α (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 α (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 α 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 $\Phi^{C}_{D,n}$ (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

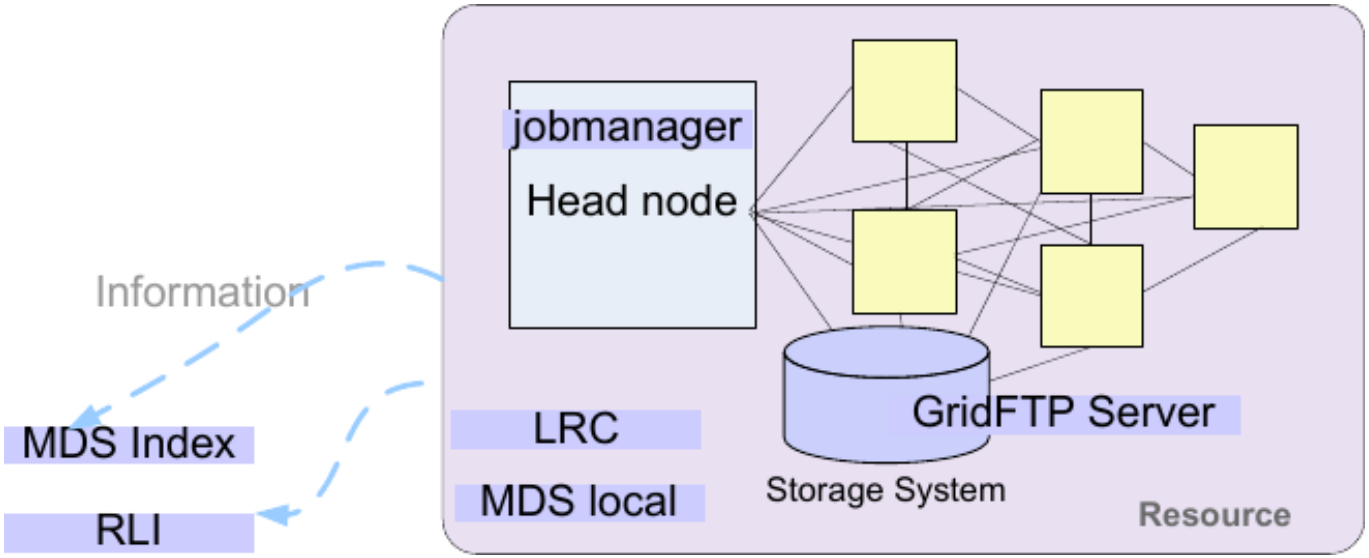
Some drawbacks

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

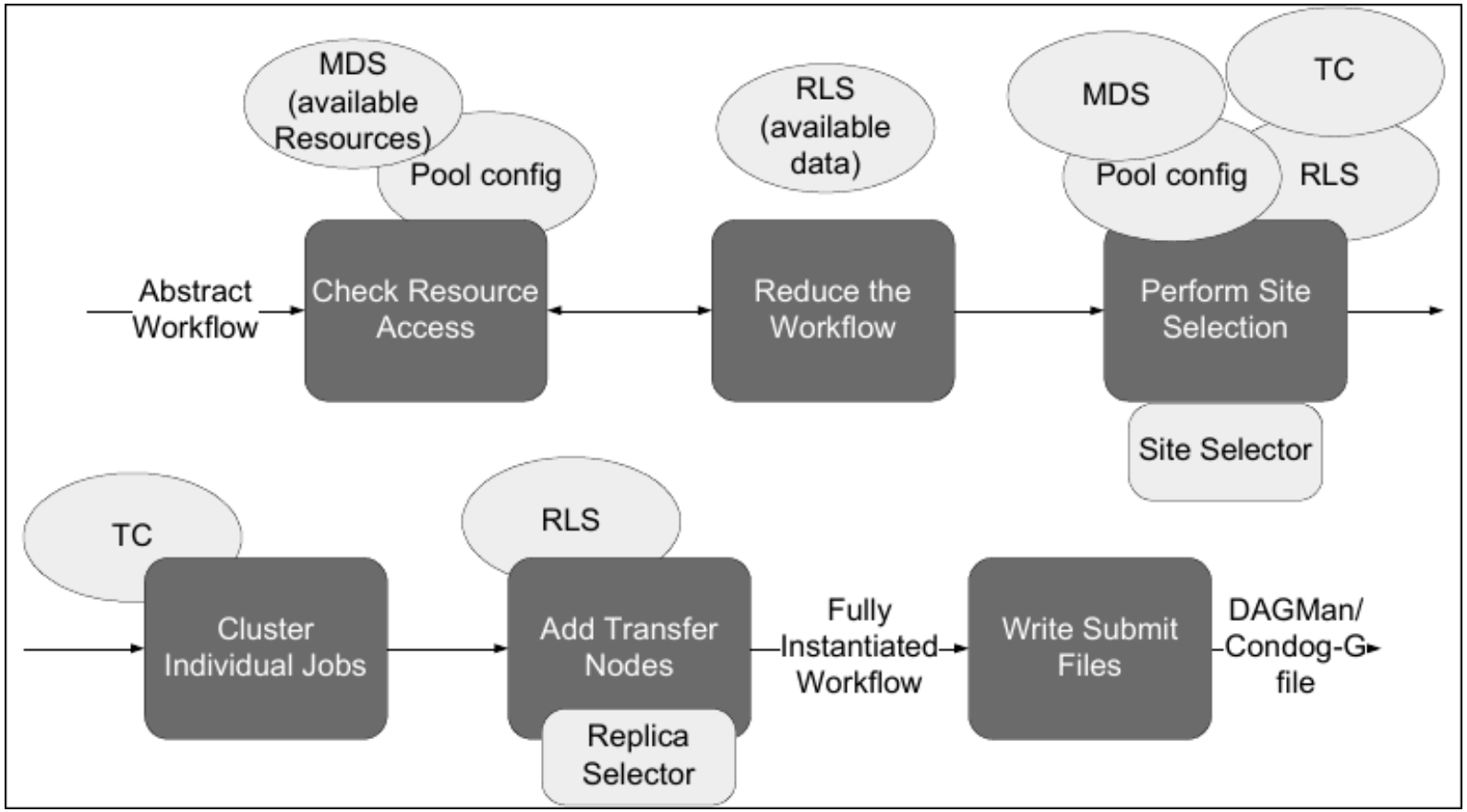
Some drawbacks

- ✗ Evaluation is done via simulation
- ✗ No theoretical proof of correctness for placement algorithm
- ✗ Very few optimizations → what about overhead costs ?

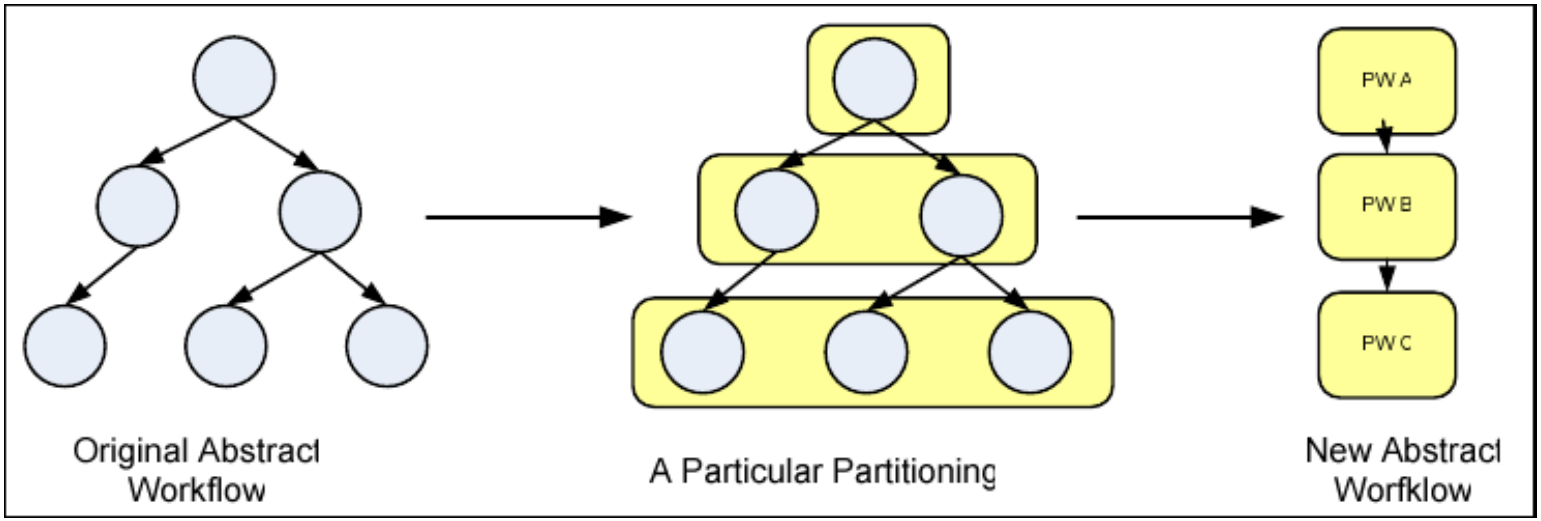
Pegasus


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

Pegasus

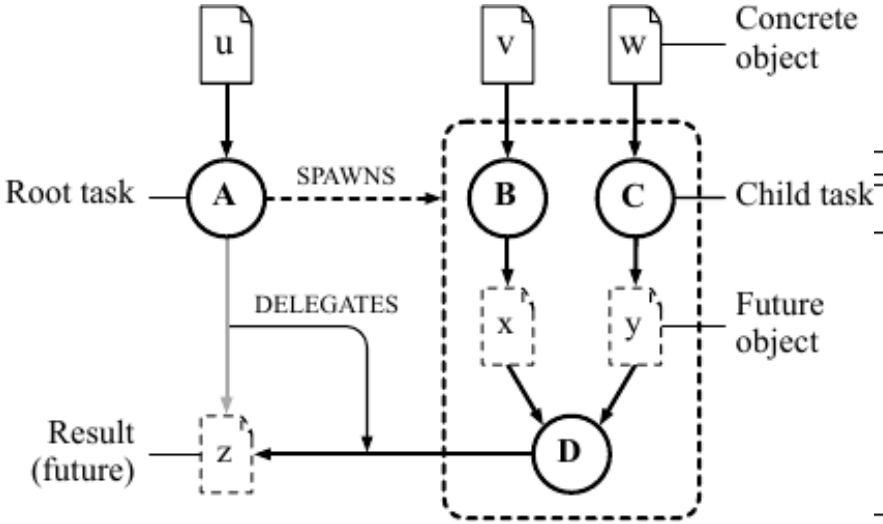

MDS (available Resources)
Pool config
RLS (available data)
MDS
TC
Pool config
RLS
Abstract Workflow
Check Resource Access
Reduce the Workflow
Perform Site Selection
Site Selector
TC
RLS
Cluster Individual Jobs
Add Transfer Nodes
Replica Selector
Fully Instantiated Workflow
Write Submit Files
DAGMan/ Condog-G file

Scheduling horizon



Plan

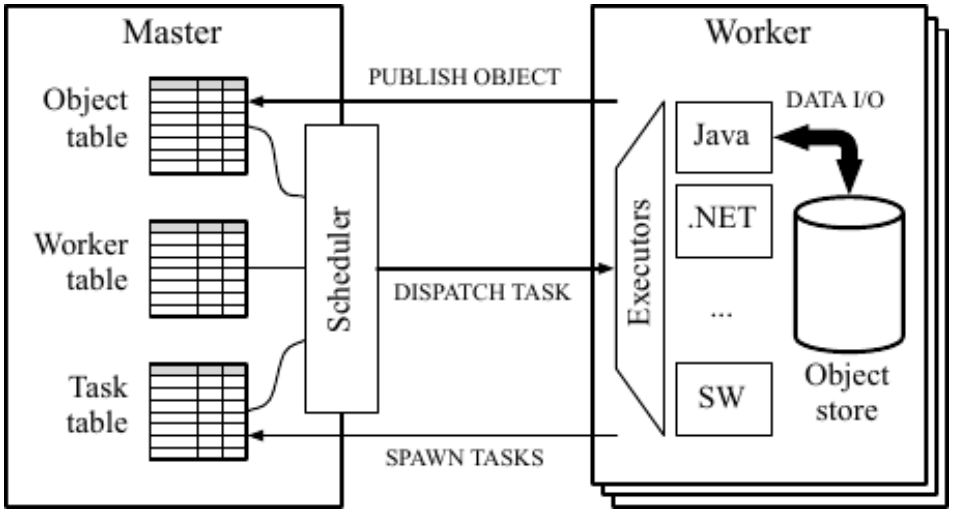
Control Flow



Task ID	Dependencies	Expected outputs
A	{ u }	~~z~~
B	{ v }	x
C	{ w }	y
D	{ x, y }	z

Object ID	Produced by	Locations
u	–	{ host19, host85 }
v	–	{ host21, host23 }
w	–	{ host22, host57 }
x	**B**	∅
y	**C**	∅
z	~~**A**~~ **D**	∅

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

```
result = spawn_exec(executor, args, n);
```

Unique Naming

for the ith output: name = executor:SHA1(arg||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):

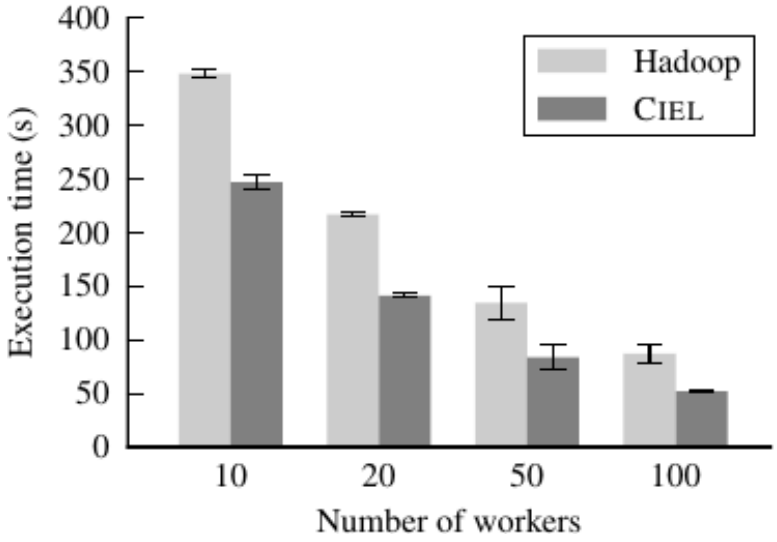
1. Workers detect master failure (heartbeat).
2. If a failure is detected, workers send registration to the master.

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

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

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

Pegasus

- ✓ Performance issue
- ✗ Not general enough

Pegasus

- ✓ Performance issue
- ✗ Not general enough

Improvement

Reliability (only partition level)

Ciel

Ciel

✓ Much more general.

Ciel

- ✓ Much more general.
- ✓ Dynamic task dependencies

Ciel

- ✓ Much more general.
- ✓ Dynamic task dependencies

Ciel

- ✓ Much more general.
- ✓ Dynamic task dependencies

Improvement

- Random funtions

Ciel

- ✓ Much more general.
- ✓ Dynamic task dependencies

Improvement

- Random funtions
- How far are you from machine performance ?

Perpectives

Perpectives

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

Perpectives

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

Perpectives

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

Plan

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