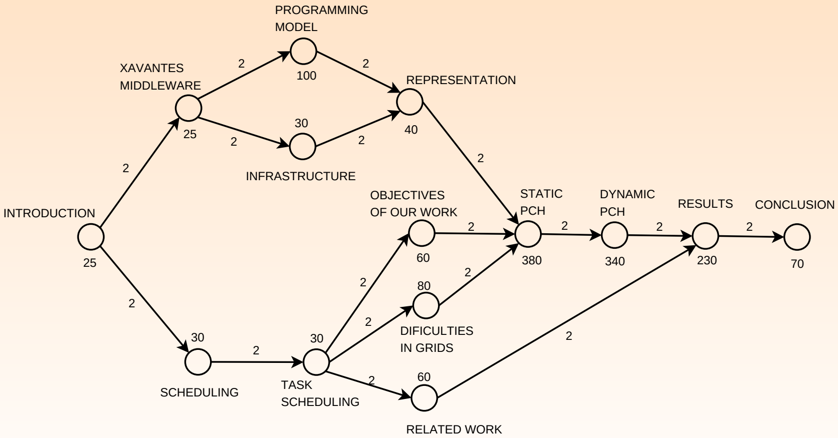
A Dynamic Approach for Scheduling Dependent Tasks on the Xavantes Grid Middleware

Luiz Fernando Bittencourt, Edmundo Roberto Mauro Madeira
{bit,edmundo}@ic.unicamp.br

Institute of Computing
State University of Campinas (UNICAMP)
Brazil

November, 27th, 2006

Summary and Tasks Representation



Summary of my presentation (Directed Acyclic Graph - DAG)

Difficulties

- Resources are heterogeneous.
  → scheduling algorithms must select resources by performance.
- Grid has no control over entries and exits of resources.
  → resources may not finish tasks' execution.
- Resources have varying performance.
  → foreseen execution times may not be real.
- Grids are potentially big.
  → scheduling algorithms must have low time complexity.

Objectives - Dynamic Path Clustering Heuristic (PCH)

- Schedule dependent tasks on Xavantes (NP-Hard).
- Create groups of dependent tasks (clusters of tasks) to avoid communication overhead generated by controllers.
- Schedule dependent tasks on nearby resources, allowing fast recovery and low communication times.
- Minimize the impact of a possible performance loss on resources.

Related Works

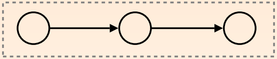
- Heterogeneous Earliest Finish Time - HEFT
- Critical Path on a Processor - CPOP
  → Static task schedulers that does not consider performance variations.
  → The spreading of tasks is not compatible with the use of controllers.
- Condor DAGMan
  → Meta-scheduler that does not consider the DAG dependencies when chosing resources.

Programming Model

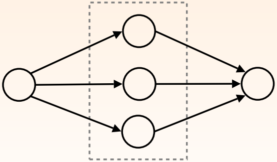
- Applications are specified as structured processes.
- Tasks are subordinated to controllers.
- Provides the necessary information about tasks and its dependencies.

Controllers

- Control structures that organize the execution of tasks.
- **All** communication between two tasks must be via its controllers.
- Sequential controllers: tasks are executed in sequence.



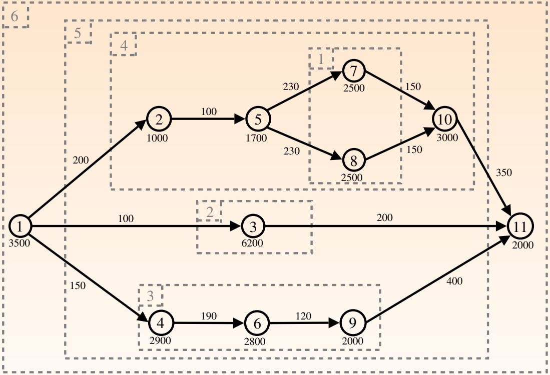
- Parallel controllers: tasks can be executed in parallel.



Controllers - Pros and Cons

- Pros:
  → Scalability: controllers distribute the processes' execution management.
  → Recovery: controllers know the execution state of the portion of the process subordinated to it.
  → They can provide communication between parallel tasks via shared variables.
- Cons:
  → They can generate communication overhead.

Process and controllers representation



Infrastructure

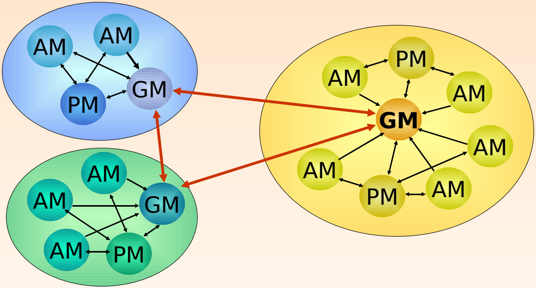


Figura: Resources organization in Xavantes.

Definitions - 1/3

- **Weight** (Computation Cost)

$$w_i = \frac{instructions_i}{power_r}$$

Represents the computation cost (execution time) of node i in resource r, where $prower_r$ is the processing power of resource r, in instructions per second.

- **Communication Cost**

$$c_{i,j} = \frac{data_{i,j}}{bandwidth_{r,t}}$$

Represents the communication cost (time for transmitting data) between nodes i and j, using the link between resources r and t. If $r = t$, $c_{i,j} = 0$.

Definitions - 1/3

- **Weight** (Computation Cost)

$$w_i = \frac{instructions_i}{power_r}$$

Represents the computation cost (execution time) of node i in resource r, where $prower_r$ is the processing power of resource r, in instructions per second.

- **Communication Cost**

$$c_{i,j} = \frac{data_{i,j}}{bandwidth_{r,t}}$$

Represents the communication cost (time for transmitting data) between nodes i and j, using the link between resources r and t. If $r = t$, $c_{i,j} = 0$.

Definitions - 2/3

- **Priority**

$$P_i = w_i + \max_{n_j \in suc(n_i)} (c_{i,j} + P_j)$$

Represents the priority level of node n_i. The priority of the exit node is $P_{\text{exit}} = w_{\text{exit}}$.

- **Earliest Start Time**

$$EST(n_i, r_k) = \max\{time(r_k), \max_{n_h \in pred(n_i)} (EST_h + w_h + c_{h,i})\}$$

Represents the lowest possible initial time for node n_i in resource r_k. In this definition, $time(r_k)$ represents the time when resource r_k will be ready to execute task n_i.

Definitions - 2/3

- **Priority**

$$P_i = w_i + \max_{n_j \in suc(n_i)} (c_{i,j} + P_j)$$

Represents the priority level of node n_i. The priority of the exit node is $P_{\text{exit}} = w_{\text{exit}}$.

- **Earliest Start Time**

$$EST(n_i, r_k) = \max\{time(r_k), \max_{n_h \in pred(n_i)} (EST_h + w_h + c_{h,i})\}$$

Represents the lowest possible initial time for node n_i in resource r_k. In this definition, $time(r_k)$ represents the time when resource r_k will be ready to execute task n_i.

Definitions - 3/3

- **Estimated Finish Time**

$$EFT(n_i, r_k) = EST(n_i, r_k) + \frac{instructions_{n_i}}{power_{r_k}}$$

Represents the estimated finish time for the execution of node n_i in resource r_k.

- We call **cluster** a group of tasks of a process. Tasks on the same cluster will be executed on the same resource.

- **Makespan** is the estimated execution time of a scheduled process (the "size" of the schedule).

Definitions - 3/3

- **Estimated Finish Time**

$$EFT(n_i, r_k) = EST(n_i, r_k) + \frac{instructions_{n_i}}{power_{r_k}}$$

Represents the estimated finish time for the execution of node n_i in resource r_k.

- We call **cluster** a group of tasks of a process. Tasks on the same cluster will be executed on the same resource.

- **Makespan** is the estimated execution time of a scheduled process (the "size" of the schedule).

PCH - Overview

```
while There are unscheduled nodes do
    cluster ⇐ get_next_cluster()
    resource ⇐ select_best_resource(cluster)
    Schedule cluster on resource
schedule_controllers()
```

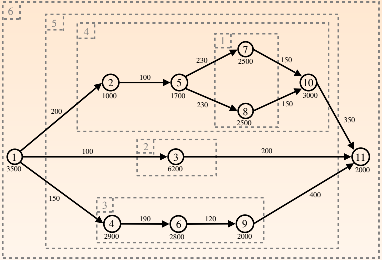
Task selection and clustering algorithm

$cluster \Leftarrow \text{get\_next\_cluster}()$

- Like a depth-first search looking on each level for the node i with the highest $P_i + EST_i$, and adding every i to the cluster.
- $P_i + EST_i$ represents the cost of the longest path from the entry node to the exit node, via node i.

Task selection and clustering algorithm

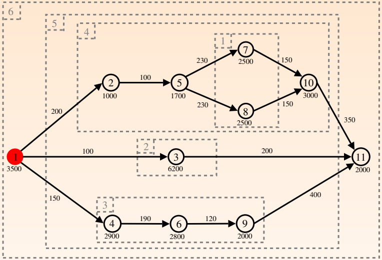
```
n ⇐ not scheduled node with the highest Priority.
cluster ⇐ cluster ∪ n
while (n has not scheduled successors) do
    n_suc ⇐ sucessor_i of n with the highest P_i + EST_i
    cluster ⇐ cluster ∪ n_suc
    n ⇐ n_suc
return cluster
```



$n \Leftarrow n_1$

Task selection and clustering algorithm

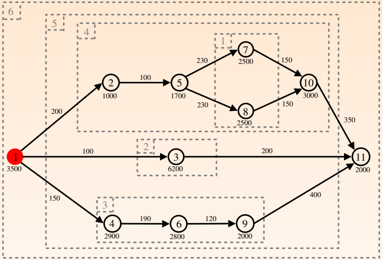
$n \Leftarrow$ not scheduled node with the highest Priority.
$cluster \Leftarrow cluster \cup n$
while (n has not scheduled successors) **do**
    $n_{suc} \Leftarrow sucessor_i$ of n with the highest $P_i + EST_i$
    $cluster \Leftarrow cluster \cup n_{suc}$
    $n \Leftarrow n_{suc}$
return $cluster$



$cluster \cup n_1 = \{n_1\}$

Task selection and clustering algorithm

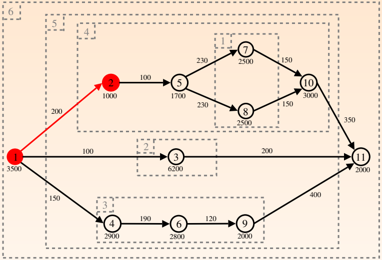
```
n ⇐ not scheduled node with the highest Priority.
cluster ⇐ cluster ∪ n
while (n has not scheduled successors) do
    n_suc ⇐ sucessor_i of n with the highest P_i + EST_i
    cluster ⇐ cluster ∪ n_suc
    n ⇐ n_suc
return cluster
```



$n_{suc} \Leftarrow n_2$

Task selection and clustering algorithm

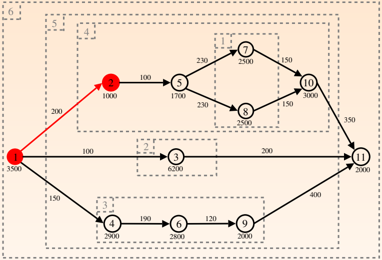
$n \Leftarrow$ not scheduled node with the highest Priority.
$cluster \Leftarrow cluster \cup n$
while (n has not scheduled successors) **do**
  $n_{suc} \Leftarrow sucessor_i$ of n with the highest $P_i + EST_i$
  $cluster \Leftarrow cluster \cup n_{suc}$
  $n \Leftarrow n_{suc}$
return $cluster$



$cluster \cup n_2 = \{n_1, n_2\}$

Task selection and clustering algorithm

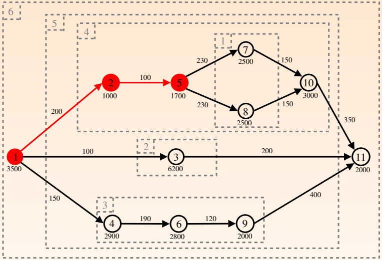
$n \Leftarrow$ not scheduled node with the highest Priority.
$cluster \Leftarrow cluster \cup n$
while (n has not scheduled successors) **do**
    $n_{suc} \Leftarrow sucessor_i$ of n with the highest $P_i + EST_i$
    $cluster \Leftarrow cluster \cup n_{suc}$
    $n \Leftarrow n_{suc}$
return $cluster$



$n_{suc} \Leftarrow n_5$

Task selection and clustering algorithm

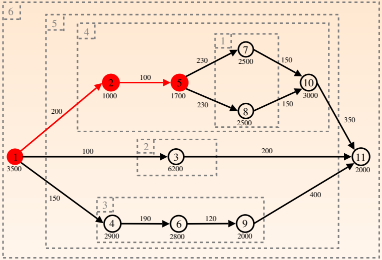
$n \Leftarrow$ not scheduled node with the highest Priority.
$cluster \Leftarrow cluster \ \cup \ n$
while (n has not scheduled successors) **do**
  $n_{suc} \Leftarrow sucessor_i$ of n with the highest $P_i + EST_i$
  $cluster \Leftarrow cluster \ \cup \ n_{suc}$
  $n \Leftarrow n_{suc}$
return $cluster$



$cluster \ \cup \ n_5 = \{n_1, n_2, n_5\}$

Task selection and clustering algorithm

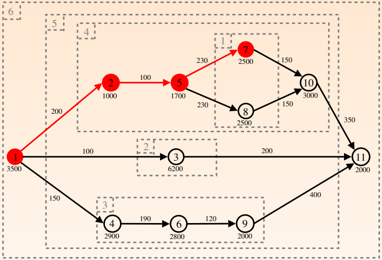
$n \Leftarrow$ not scheduled node with the highest Priority.
$cluster \Leftarrow cluster \cup n$
while (n has not scheduled successors) **do**
    $n_{suc} \Leftarrow sucessor_i$ of n with the highest $P_i + EST_i$
    $cluster \Leftarrow cluster \cup n_{suc}$
    $n \Leftarrow n_{suc}$
return $cluster$



$n_{suc} \Leftarrow n_7$

Task selection and clustering algorithm

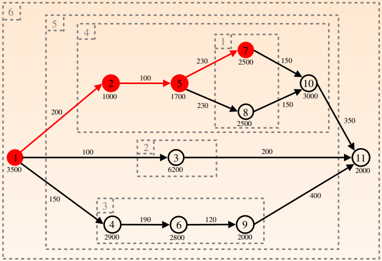
$n \Leftarrow$ not scheduled node with the highest Priority.
cluster $\Leftarrow$ *cluster* $\cup$ n
while (n has not scheduled successors) **do**
    $n_{suc} \Leftarrow sucessor_i$ of n with the highest $P_i + EST_i$
    $cluster \Leftarrow cluster \cup n_{suc}$
    $n \Leftarrow n_{suc}$
return *cluster*



$cluster \cup n_7 = \{n_1, n_2, n_5, n_7\}$

Task selection and clustering algorithm

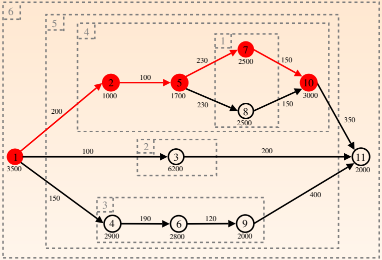
$n \Leftarrow$ not scheduled node with the highest Priority.
$cluster \Leftarrow cluster \ \cup \ n$
while (n has not scheduled successors) **do**
    $n_{suc} \Leftarrow sucessor_i$ of n with the highest $P_i + EST_i$
    $cluster \Leftarrow cluster \ \cup \ n_{suc}$
    $n \Leftarrow n_{suc}$
return $cluster$



$n_{suc} \Leftarrow n_{10}$

Task selection and clustering algorithm

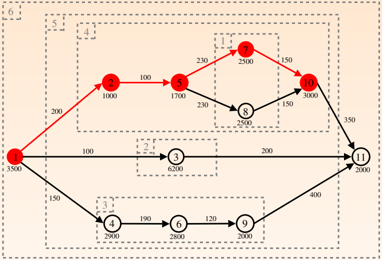
$n \Leftarrow$ not scheduled node with the highest Priority.
$cluster \Leftarrow cluster \cup n$
while (n has not scheduled successors) **do**
  $n_{suc} \Leftarrow sucessor_i$ of n with the highest $P_i + EST_i$
  $cluster \Leftarrow cluster \cup n_{suc}$
  $n \Leftarrow n_{suc}$
return $cluster$



$cluster \cup n_{10} = \{n_1, n_2, n_5, n_7, n_{10}\}$

Task selection and clustering algorithm

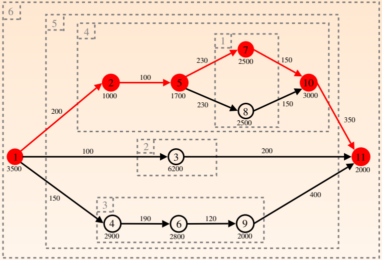
$n \Leftarrow$ not scheduled node with the highest Priority.
$cluster \Leftarrow cluster \ \cup \ n$
while (n has not scheduled successors) **do**
    $n_{suc} \Leftarrow sucessor_i$ of n with the highest $P_i + EST_i$
    $cluster \Leftarrow cluster \ \cup \ n_{suc}$
    $n \Leftarrow n_{suc}$
return $cluster$



$n_{suc} \Leftarrow n_{11}$

Task selection and clustering algorithm

```
n ⇐ not scheduled node with the highest Priority.
cluster ⇐ cluster ∪ n
while (n has not scheduled successors) do
    n_suc ⇐ sucessor_i of n with the highest P_i + EST_i
    cluster ⇐ cluster ∪ n_suc
    n ⇐ n_suc
return cluster
```



$cluster \cup n_{11} = \{n_1, n_2, n_5, n_7, n_{10}, n_{11}\}$

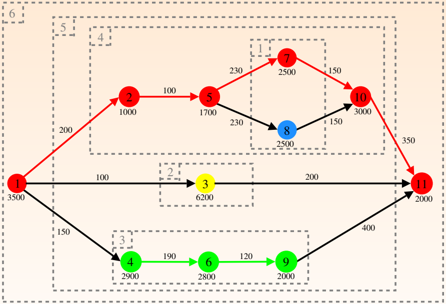
Resource Selection Algorithm

$resource \Leftarrow \text{select\_best\_resource}(cluster)$

- The selected resource for a cluster c is the one that gives the smallest EST for the node that succeeds c in the graph.

Resource Selection Algorithm

for all $r \in$ *resources* **do**
  schedule $\Leftarrow$ Insert *cluster* in $schedule_r$
  compute_EFT(*schedule*)
  $time_r \Leftarrow$ compute_EST(successor(*cluster*))
return resource r with the lowest $time_r$



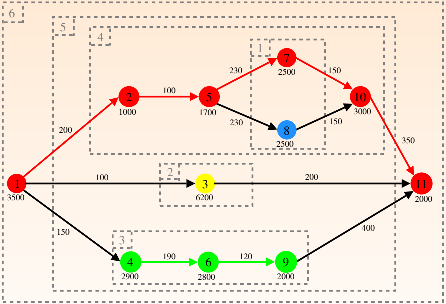
Resources

Rec.	Power	Bandwidth			
0	133	∞	10	5	5
1	130	10	∞	5	5
2	118	5	5	∞	10
3	90	5	5	10	∞

Resource 0: $n_1, n_2, n_5, n_7, n_{10}, n_{11} \longrightarrow 103.0$
Resource 1: $n_1, n_2, n_5, n_7, n_{10}, n_{11} \longrightarrow 105.4$
Resource 2: $n_1, n_2, n_5, n_7, n_{10}, n_{11} \longrightarrow 116.1$
Resource 3: $n_1, n_2, n_5, n_7, n_{10}, n_{11} \longrightarrow 152.2$

Resource Selection Algorithm

for all $r \in resources$ **do**
  $schedule \Leftarrow$ Insert $cluster$ in $schedule_r$
  compute_EFT($schedule$)
  $time_r \Leftarrow$ compute_EST(successor($cluster$))
return resource r with the lowest $time_r$



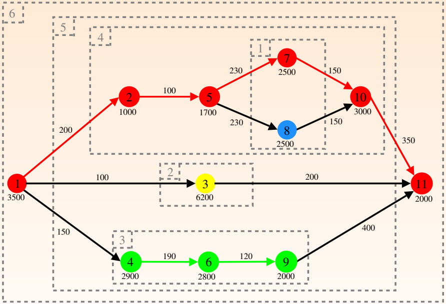
Resources

Rec.	Power	Bandwidth			
0	133	∞	10	5	5
1	130	10	∞	5	5
2	118	5	5	∞	10
3	90	5	5	10	∞

Resource 0: $n_1, n_2, n_4, n_5, n_7, n_6, n_{10}, n_9, n_{11} \longrightarrow 145.9$
Resource 1: $n_4, n_6, n_9 \longrightarrow 100.5$
Resource 2: $n_4, n_6, n_9 \longrightarrow 121.6$
Resource 3: $n_4, n_6, n_9 \longrightarrow 141.9$

Resource Selection Algorithm

for all $r \in resources$ **do**
  $schedule \Leftarrow$ Insert *cluster* in $schedule_r$
  compute_EFT(*schedule*)
  $time_r \Leftarrow$ compute_EST(successor(*cluster*))
return resource r with the lowest $time_r$



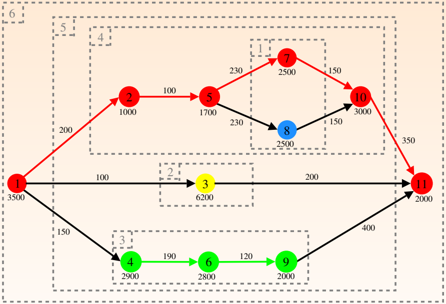
Resources

Rec.	Power	Bandwidth			
0	133	∞	10	5	5
1	130	10	∞	5	5
2	118	5	5	∞	10
3	90	5	5	10	∞

Resource 0: $n_1, n_2, n_4, n_5, n_7, n_6, n_{10}, n_9, n_{11} \longrightarrow 145.9$
Resource 1: $n_4, n_6, n_9 \longrightarrow 140.5$
Resource 2: $n_4, n_6, n_9 \longrightarrow 201.6$
Resource 3: $n_4, n_6, n_9 \longrightarrow 221.9$

Resource Selection Algorithm

for all $r \in$ *resources* **do**
    schedule $\Leftarrow$ Insert *cluster* in $schedule_r$
    compute_EFT(*schedule*)
    $time_r \Leftarrow$ compute_EST(successor(*cluster*))
return resource r with the lowest $time_r$



Resources

Rec.	Power	Bandwidth			
0	133	∞	10	5	5
1	130	10	∞	5	5
2	118	5	5	∞	10
3	90	5	5	10	∞

Resource 0: $n_1, n_2, n_5, n_7, n_8, n_{10}, n_{11} \longrightarrow 155.5$
Resource 1: $n_4, n_6, n_9 \longrightarrow 100.5$
Resource 2: $n_3 \longrightarrow 98.8$
Resource 3:

Path Clustering Heuristic

```
while there are unscheduled nodes do
    cluster ⇐ get_next_cluster()
    resource ⇐ select_best_resource(cluster)
    Schedule cluster on resource
    Recompute Weights, ESTs e EFTs
schedule_controllers()
```

Resulting Schedule

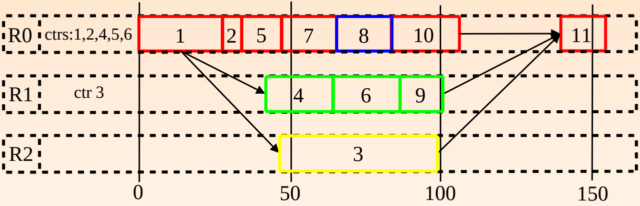


Figura: Resulting static schedule.

Dynamic Approach

- But what if the owner of resource zero starts executing other jobs independent of the grid jobs?
  → If all tasks are sent to execution with this schedule, a performance loss could delay the execution.
- A dynamic approach could be used to send tasks to execution as other tasks finishes.
  → At each group of tasks finished, the scheduler has a new view of the resources, with knowledge about the current performances.
  → It avoids reallocation of tasks, what consumes bandwidth and time.

Dynamic Approach Overview

Schedule DAG G using the static PCH Algorithm
while not(all nodes of G have finished) **do**
    Select tasks to execute according to a policy.
    Send tasks of this round to execution.
    Evaluate the resources performance.
    Reschedule tasks if necessary.

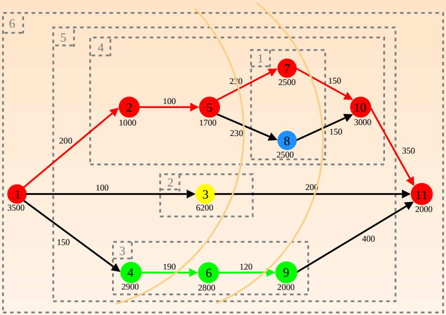
Rounds and Dynamic Reschedule

- With all nodes scheduled, the algorithm decides which nodes will be sent to execution.
- As the tasks are being executed, the algorithm can compare the real execution times with that estimated by the calculated attributes (EST, EFT, Weight).
  → If the performance of a resource is below a threshold, the tasks scheduled on that resource are rescheduled using the static PCH.

Rounds and Dynamic Reschedule

- A node n is sent to execution in the round k (of a total of R rounds) if n has not started its execution and if $EFT_n \leq \frac{makespan}{R/k}$, with $1 \leq k \leq R$, or if there is no task on the resource where n is scheduled.
- It is like cutting the graph into small pieces, then sending these pieces to execution one after another.

Dynamic Approach Example



Node	EFT
1	26.3
2	33.8
3	98.8
4	63.6
5	46.6
6	85.1
7	65.4
8	84.2
9	100.5
10	106.8
11	155.5

Resources

Resource 0: n_1, n_2, n_5, n_7, n_8, n_{10}, n_{11} $\longrightarrow$ 155.5
Resource 1: n_4, n_6, n_9 $\longrightarrow$ 100.5
Resource 2: n_3 $\longrightarrow$ 98.8
Resource 3:

Performance Metrics - 1/2

- Schedule Length Ratio

$$SLR = \frac{makespan}{\sum_{n_i \in CP} \frac{instructions_{n_i}}{power_{best}}}$$

where, CP is the set of nodes that compose the critical path of the initial graph, and $power_{best}$ is the processing power of the best resource available.
$\longrightarrow$ SLR tells how many times the given makespan is bigger than the execution of the critical path on the best resource.

Performance Metrics - 2/2

- Speedup

$$Speedup = \frac{\sum_{n_i \in V} \frac{instructions_{n_i}}{power_{best}}}{makespan}$$

→ Speedup tells how many times faster the execution with the current schedule is when compared to the execution of all tasks in sequence on the best resource.

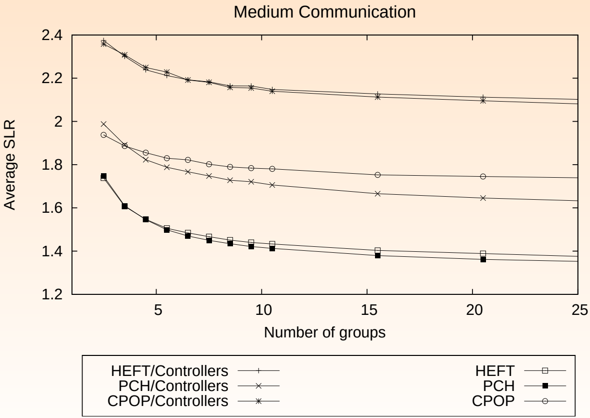
- The number of times an algorithm gives the best schedule (lowest makespan among all compared) is also a comparison metric.

Experiments

- 15 DAGs randomly taken, with random topology.
- Each DAG was scheduled 1000 times, with random computation and communication costs.
- Medium and high Communication scenarios.
  Medium: → communication and computation costs randomly taken on the same interval.
  High: → each communication more costly than each computation.
- 2 to 25 groups in Xavantes topology, each group with a random number of resources between 1 and 5.

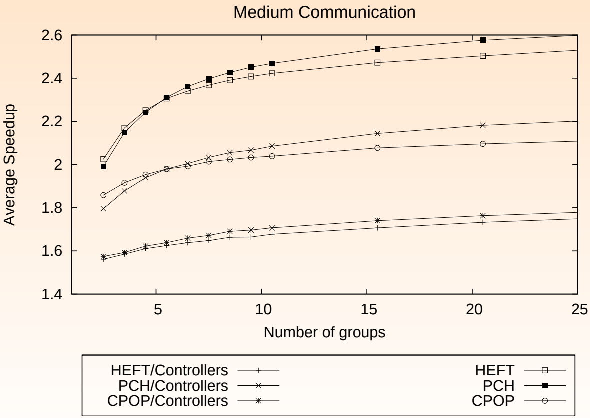
Static PCH

SLR with Medium Communication



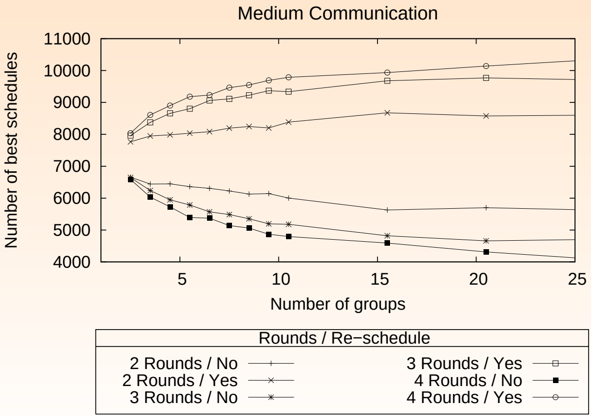
Static PCH

Speedup with Medium Communication



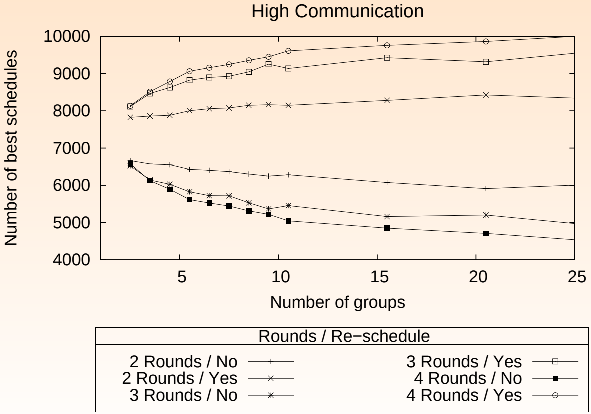
Dynamic PCH vs. Static PCH

Number of best schedules with medium communication.



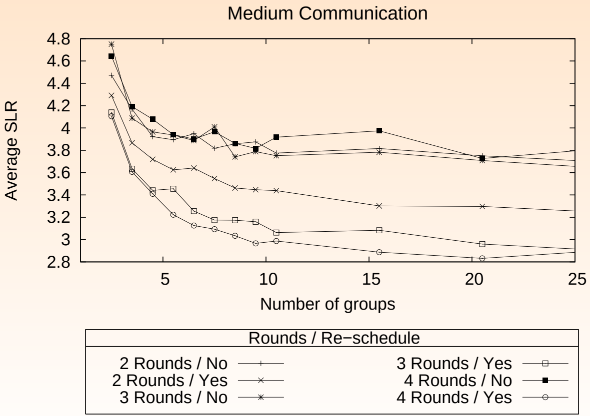
Dynamic PCH vs. Static PCH

Number of best schedules with high communication.



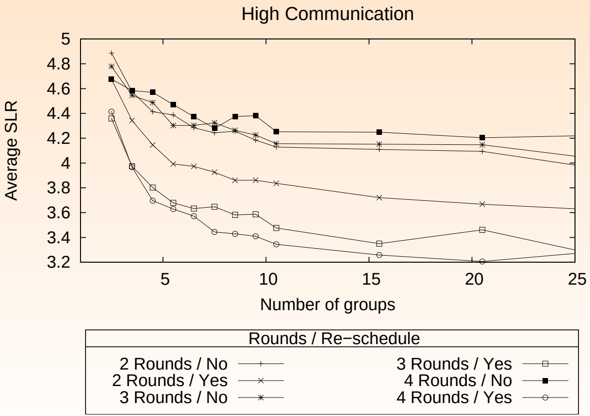
Dynamic PCH vs. Static PCH

SLR with medium communication.



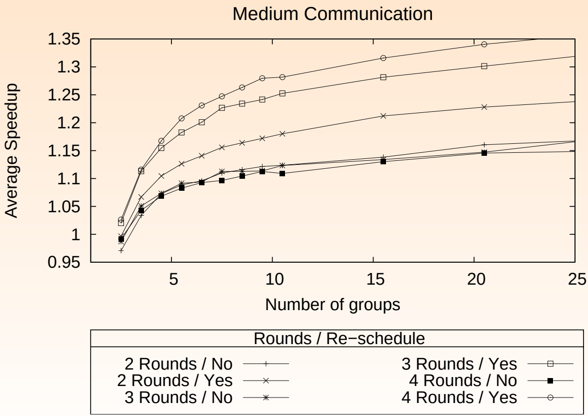
Dynamic PCH vs. Static PCH

SLR with high communication.



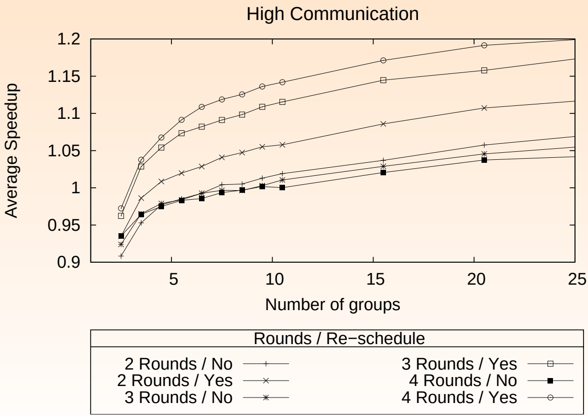
Dynamic PCH vs. Static PCH

Speedup with medium communication



Dynamic PCH vs. Static PCH

Speedup with high communication.



Conclusion

- Dynamic task scheduling is very important for dynamic systems like grids.
- The proposed dynamic approach can deal with performance losses in resources.
- As the number of rounds increases, better are the results.
  $\rightarrow$ but too much rounds = too much computation.
- The rounds concept can be applied to other DAG scheduling systems.

Future Work

- Improve the round task selection criteria (how/where to cut the DAG on each round).
- Development of an adaptive number of rounds for each graph.
- History-based performance prediction.
- Reallocation policy for big tasks in poor performance resources.
- Implementation and evaluation of the round concept with other DAG scheduling heuristics.

Thanks

Thank you. Questions?

Acknowledgements:

FAPESP

CAPES
www.capes.gov.br

INSTITUTE OF COMPUTING