# Online scheduling for moldable tasks in clusters

**Lamiel Toch**

Supervisors: **Laurent Philippe** and **Jean-Marc Nicod**
Laboratoire d'Informatique de Franche-Comté
Université de Franche-Comté
Besançon

Aussois - june 2010

INRIA
GRAAL

# References


- Pierre-François Dutot, Marco A.S. Netto, Alfredo Goldman, Fabio Kon, *Scheduling Moldable BSP Tasks*
- Pierre-François Dutot, *Hierarchical Scheduling for Moldable Tasks extended version*
- Mark Stillwell, Frédéric Vivien, Henri Casanova, *Dynamic Fractional Resource Scheduling for HPC Workloads*

# Sommaire

# Contents

# Introduction

## What is a moldable task ?

- a parallel task
- the number of processors for the task is set before its execution
- when the number **n** of processors allocated to it increases, its runtime does not increase
- when **n** increases, the surface **S** of the task does not decrease

## In which context ?

in cluster batch schedulers and using virtualization

# Introduction

## What is a moldable task ?

- a parallel task
- the number of processors for the task is set before its execution
- when the number **n** of processors allocated to it increases, its runtime does not increase
- when **n** increases, the surface **S** of the task does not decrease

## In which context ?

in cluster batch schedulers and using virtualization

# Introduction

## What is a moldable task ?

- a parallel task
- the number of processors for the task is set before its execution
- when the number **n** of processors allocated to it increases, its runtime does not increase
- when **n** increases, the surface **S** of the task does not decrease

| n | runtime | S |
|---|---|---|
| 1 | 60 | 60 |
| 2 | 30 | 60 |
| 3 | 20 | 60 |
| 4 | 20 | 80 |
| 5 | 20 | 100 |
| 6 | 10 | 60 |

## In which context ?

in cluster batch schedulers and using virtualization

# Introduction

## What is a moldable task ?

- a parallel task
- the number of processors for the task is set before its execution
- when the number **n** of processors allocated to it increases, its runtime does not increase
- when **n** increases, the surface **S** of the task does not decrease

| n | runtime | S |
|---|---|---|
| 1 | 60 | 60 |
| 2 | 30 | 60 |
| 3 | 20 | 60 |
| 4 | 20 | 80 |
| 5 | 20 | 100 |
| 6 | 10 | 60 |

## In which context ?

in cluster batch schedulers and using virtualization

# Existing batch Schedulers

## Implementations in cluster batch schedulers

- FIFO : PBS, Sun Grid Engine
- BackFilling : Maui, OAR

## Jobs constraints

Users submit their jobs with :

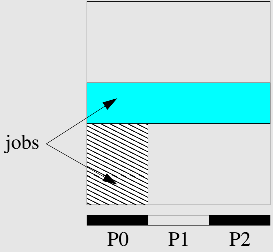
- requested processors : *req_procs*
- requested_time : *req_time*

→ batch scheduler must respect the jobs deadlines

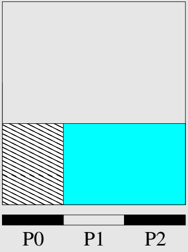
# Tasks folding

In the literature, Pierre-François Dutot et al. → tasks folding in an offline context.
Our approach :

- online.
- $alloc\_procs \leq req\_procs$
- respect of execution order (deadline)



FIFO



Task Folding

# Virtualization

In the literature, Mark Stillwell et al. → virtualization time-sharings techniques in batch scheduler.
Our approach → no time-sharing, no job migration.

## Why using virtualization with tasks folding ?

- using virtual processor cores when not enough processors allows task folding
- transparency for the batch schedulers : virtual nodes seen as physical nodes
- locking task in its cores

# Definitions

Let $J$ a job. We assume that

$$runtime_J(alloc\_procs_J) = runtime_J(req\_procs_J) \times \left\lceil \frac{req\_procs_J}{alloc\_procs_J} \right\rceil$$

## Tasks Folding Definitions

- Integer Folding : $alloc\_procs_J$ divides $req\_procs_J$
- Non-Integer Folding : $alloc\_procs_J$ does not divide $req\_procs_J$

# Contents

# Heuristic H1 : Integer task folding

For each job $J$ of the waiting queue do

choose $max(n)$ processors such as

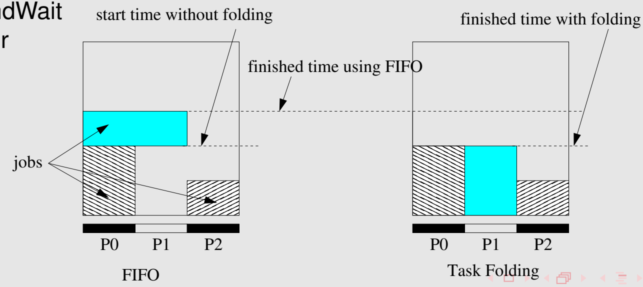
$\left(req\_procs_J \ mod \ n = 0 \text{ and } req\_time_J(n) + start\_time_J(n) \leq start\_time\_without\_folding_J\right)$

Wait until $n$ processors are available

execute $J$

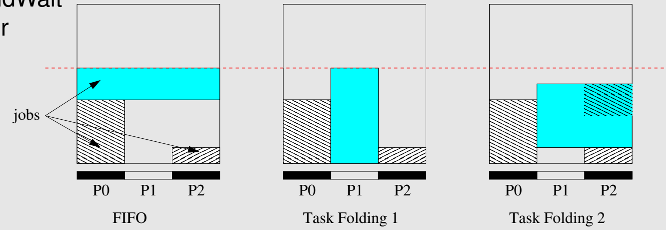
EndWait

EndFor



# Heuristic H2 : Non-Integer task folding

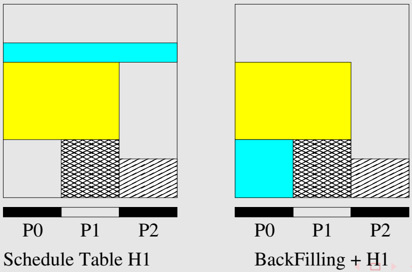
For each job $J$ of the waiting queue do
choose $n$ processors such as
$\left( req\_time_J(n) + start\_time_J(n) \text{ is minimum} \right)$
Wait until $n$ processors is available
execute $J$
EndWait
EndFor



# Heuristic H3 : Integer task folding H1 + backfilling

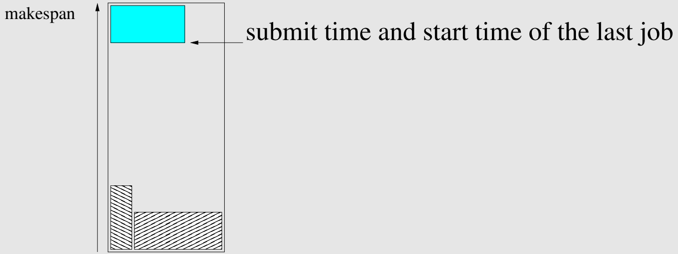
For each job *J* of the waiting queue do
    Schedule J in a schedule table with heuristic H1
EndFor

In parallel when a resource is released execute backfilling with the jobs of the schedule table to choose which jobs to be executed



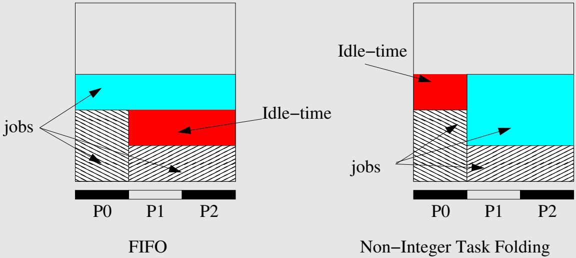
# Contents

# The Makespan




In logs of clusters → no utilization and then arrival of one job.
The makespan is determined by this job : unfillable hole.
→ Experiment with real workload not successful.

# The idle-time




## The idle-time maybe a good metric ?

If an no-integer folding is applied on a task, its surface is increasing, so the idle-time decreases, but practically nothing has be gained.

# Contents

# Experimental settings

## An experiment with a NASA workload

`http://www.cs.huji.ac.il/labs/parallel/workload/l_nasa_ipsc/index.html`

Cluster : 128 processors and 18239 Jobs

## An experiment with a synthetic workload of 10000 jobs

- random number of job's requested processors between 30 and 90
- random requested time before 200 s and 20000 s
- two consecutive submission times differ randomly from 1s to 100s
- the runtime differs from the requested time from 0% to 30%

# Metrics used

## Metrics

- makespan with synthetic workload because no unfillable holes
- number of jobs finished before using FIFO scheduler and gained time
- comparisons between algorithms : FIFO used as reference

# The Makespan on the real workload

## An experiment with a NASA workload

Whatever the algorithm , the makespan is unchanged on this workload : $543,236,801$ s $\approx 17$ years
Explanation : Users don't submit any jobs during a long time.

# The Makespans with the synthetic workload

## Makespans

| Algo | makespan in (s) | improvement (%) |
|---|---|---|
| FIFO | $6.8373.10^8$ | - |
| Integer Task Folding H1 | $5.98137.10^8$ | 12.5 |
| Non-Integer Task Folding H2 | $6.44008.10^8$ | 5.8 |
| Backfilling | $5.58397.10^8$ | 18 |
| H1 + Backfilling | $5.32475.10^8$ | 22 |

# Number of tasks finished before using FIFO

## Real NASA Workload

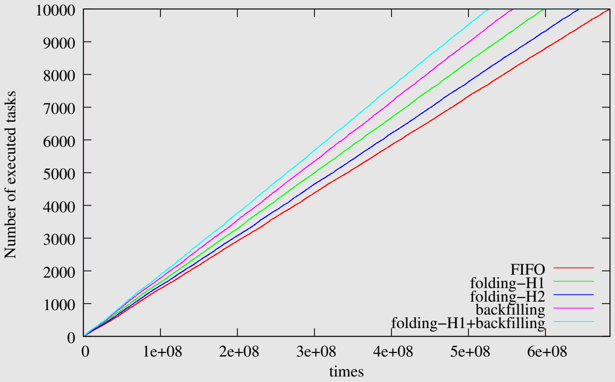
| Algo | number of tasks | gain in (s) |
|---|---|---|
| Integer Task Folding H1 | 7 | 121380 |
| Non-Integer Task Folding H2 | 7 | 132138 |
| Backfilling | 4 | 70161 |
| H1 + Backfilling | 8 | 135145 |

# Number of tasks finished before using FIFO

## Synthetic Workload

| Algo | number of tasks | gain in (s) |
|---|---|---|
| Integer Task Folding H1 | 9998 | 425,461,824,083 |
| Non-Integer Task Folding H2 | 9998 | 195,822,977,276 |
| Backfilling | 9963 | 626,273,927,451 |
| H1 + Backfilling | 9974 | 785,306,481,841 |

# Number of executed tasks in time on a synthetic workload



# Conclusion and future works

Heuristic H1+backfilling gives good results in :

- makespans with synthetic workload
- gained times with synthetic and real workloads

Future Works

- pursue experiments with gaussian synthetic workloads
- pursue experiments on other real workloads
- find other metrics
- find other task folding algorithms based on strip-packing and binary search

# Thank you !