Batch Scheduling for Identical Multi-Tasks Jobs on Heterogeneous Platforms

Jean-Marc Nicod (Jean-Marc.Nicod@lifc.univ-fcomte.fr)
Sékou Diakité, Laurent Philippe

- 16/05/2008

Laboratoire d'Informatique de l'Université de Franche-Comté - Besançon

2nd *"Scheduling in Aussois"* workshop

Presentation Outline

Outline

- Execution Platform:
  - non oriented graph, $G = (P, L)$:
    - P: processors $p_i, i \in [1, m]$
    - L: communication links $l_j, j \in [1, c]$
- Jobs:
  - DAGs without fork (*intrees*), $J = (T, D)$:
    - T: tasks $t_k, k \in [1, n]$
    - D: tasks dependencies $d_l, l \in [1, d]$
  - N instances of the same job.

Scheduling Problem

- Characteristics:
  - Each task of a job must be performed by a specific function,
  - F is the set of the functions needed to process a job,
  - Each execution resource provides a subset of F,
  - The execution resources are unrelated.
- Problem:
  - Schedule a batch of N jobs J
  - Objective function: C_{max}
  - $R_m \mid intrees, batch\ of\ identical\ jobs \mid C_{max}$

Scheduling Problem

Problem illustration

N instances of the same job:

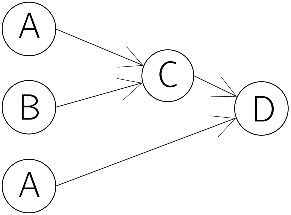


Figure: Job

1 platform for execution:

		p_1	p_2	p_3	p_4
Type	A	20	∞	∞	15
	B	10	10	∞	∞
	C	∞	10	10	∞
	D	∞	∞	10	10

Table: Execution times

- Grid:
  - Image processing: filters, 2D or 3D reconstructions,
  - Servers provides an application set.
- Micro-Factories:
  - Composed of cells: assembly, treatments, ...
  - Less geographical constraints,
  - Products are micro-metric $\longrightarrow$ easily buffered

Outline

Possible solutions

- Classical (Off-line):
  - schedule a DAG on an heterogeneous platform,
  - C_{max} optimization $\rightarrow$ NP-Hard,
  - batch of jobs: no use of identical jobs.
- Steady state technics (Off-line):
  - flow optimization: maximize the throughput
  - optimal solution on heterogeneous platforms,
  - use the identical job characteristic,
  - does not take starting/ending into account.
- On-line:
  - batch $\rightarrow$ waiting queue,
  - schedule ready tasks,
  - no use of identical job characteristic,

Used algorithms

- *On-line* scheduling:
  - simple, assign tasks on the fly,
  - respect dependencies,
  - used as reference.
- Genetic Meta-heuristic *GATS*[Daoud05]:
  - improves list scheduling,
  - good results on DAG schedule,
  - needs to be adapted to batches: period,
  - performances of a standard heuristic on batches of jobs?
- Steady State[Beaumont04]:
  - optimal for batches of infinite size,
  - performances on finite size batches?

- Definition and resolution of a linear program:
  - define the constraints of the problem,
  - flow: solutions are time ratios per resources.
- Compute a cyclic schedule:
  - construct allocations with respect of the ratios,
  - 1 port model: communication intervals.
- Execution:
  - starting,
  - cyclic schedule,
  - ending.

Steady State

linear system

$$MAXIMIZE\ \rho = \sum_{i=1}^{p} cons(p_i, t_f),$$

$$UNDER\ \ THE\ \ CONSTRAINTS$$

$$\begin{cases} (1) p_i, t_k \in T, \alpha(p_i, t_k) = cons(p_i, t_k) \times c_{i,k} \\ (2) p_i, t_k \in T, 0 \leq \alpha(p_i, t_k) \leq 1 \\ (3) p_i, \sum_{t_k \in T} \alpha(p_i, t_k) \leq 1 \end{cases}$$

Steady State

Solution

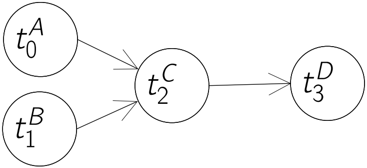


Figure: Job

		p_1	p_2	p_3	p_4
Type	A	20	∞	∞	20
	B	10	10	∞	∞
	C	∞	10	10	∞
	D	∞	∞	10	10

Table: Cost: c

	p_1	p_2	p_3	p_4
t_0^A	$7/200$	-	-	$9/200$
t_1^B	$3/100$	$1/20$	-	-
t_2^C	-	$1/20$	$3/100$	-
t_3^D	-	-	$7/100$	$1/100$

Table: Comsumption: *cons*, objective $\rho = 2/25$

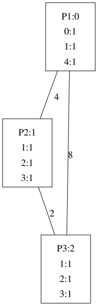


Figure: Platform

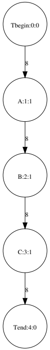
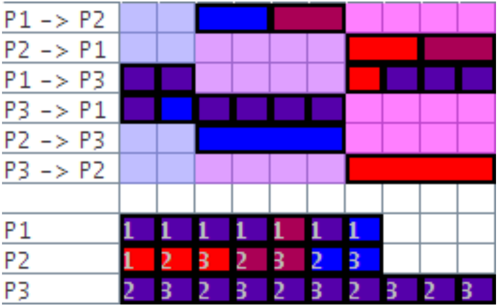


Figure: Job



Outline

$$efficiency = makespan_o / makespan_r$$

$$efficiency = N / (\rho \times makespan_r)$$

- $makespan_o$: lower bound
  - $makespan_o$: N/ρ,
  - reference time
- $makespan_r$: *makespan* resulting from experience,
- Simulation results (SimGrid),
- Communications neglected.

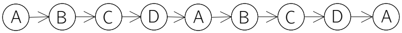


Figure: Job j_0

		p_1	p_2	p_3	p_4	p_5	p_6
Type	A	10	∞	∞	∞	100	1000
	B	∞	10	∞	∞	10	∞
	C	∞	∞	10	∞	10	10
	D	∞	∞	∞	10	∞	∞

Table: Grid G_0

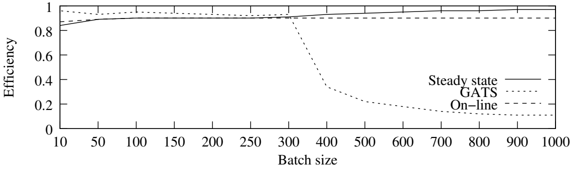


Figure: Execution of batches j_0 on G_0

Global results:

- Steady state tends toward optimal,
- GATS good for small batches, then collapses,
- *on-line* is constant.

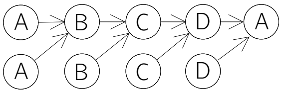


Figure: Job j_1

		p_1	p_2	p_3	p_4	p_5	p_6
Type	A	10	∞	∞	∞	100	1000
	B	∞	10	∞	∞	10	∞
	C	∞	∞	10	∞	10	10
	D	∞	∞	∞	10	∞	∞

Table: Grid G_0

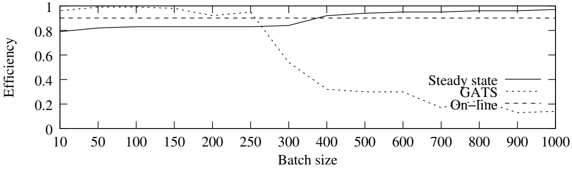


Figure: Execution of batches j_1 on G_0

- Sames tasks as j_0,
- GATS collapses earlier (250 instances vs. 300).

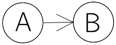
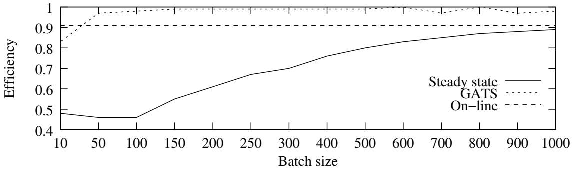


Figure: Job j_2

		p_1	p_2	p_3	p_4	p_5	p_6
Type	A	10	∞	∞	∞	100	1000
	B	∞	10	∞	∞	10	∞
	C	∞	∞	10	∞	10	10
	D	∞	∞	∞	10	∞	∞

Table: Grid G_0



- Steady state uses p_6 (A: 1000),

Figure: Execution of batches j_2 on G_0

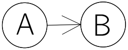


Figure: Job j_2

		p_1	p_2	p_3
Type	A	10	50	40
	B	100	∞	∞

Table: Grid G_1

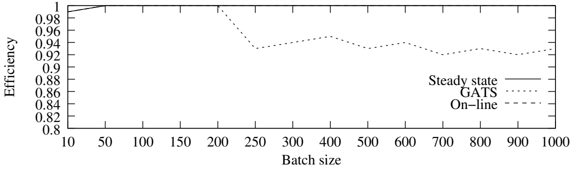


Figure: Execution of batches j_2 on G_1

- The efficiency of GATS decreases starting at 200 instances.

Outline

Small batches: 50			Medium batches: 100			Large batches: 500		
On-line	Steady	GATS	*On-line*	Steady	GATS	*On-line*	Steady	GATS
0.93	0.93	<u>0.99</u>	0.94	0.94	<u>0.99</u>	0.94	<u>0.97</u>	0.61

Table: Mean efficiency

- Synthesis:
  - GATS: up to 200,
  - Steady state: from 500.
- Time consumption for 1000 instances:
  - steady state: 0.08s,
  - *on-line*: 35.04s,
  - GATS: 1799.68s,

Outline

Steady state for small batches

Means of action

- Aim: improve (decrease) the global *makespan* for small batches,
- The steady state phase is optimal.
- Schedule starting/ending phases on the heterogeneous platform:
  - NP-Hard $\rightarrow$ find a good schedule,
  - reduce the work in initialization/ending phases.
- What can be done on starting/ending phases?
  - Keeping steady state optimal:
    - re-organise affectations to reduce the period size,
    - resolve dependencies inside the period.
  - Deterioration of steady state:
    - reduce the number of instances per periods

Steady state schedule

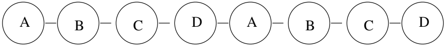
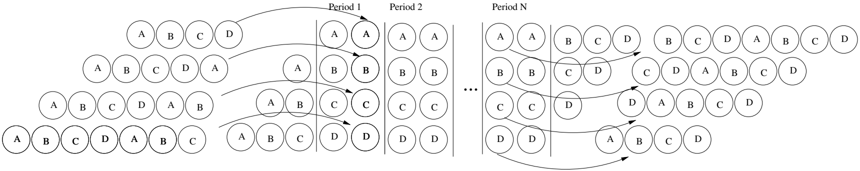


Figure: Job j_3

		p_1	p_2	p_3	p_4
Type	A	10	∞	∞	10
	B	10	10	∞	∞
	C	∞	10	10	∞
	D	∞	∞	10	10

Table: Grid G_2



Outline

Steady state for small batches

Example

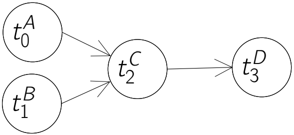


Figure: Job

		p_1	p_2	p_3	p_4
Type	A	20	∞	∞	20
	B	10	10	∞	∞
	C	∞	10	10	∞
	D	∞	∞	10	10

Table: Cost

	p_1	p_2	p_3	p_4
t_0^A	7/200	-	-	9/200
t_1^B	3/100	1/20	-	-
t_2^C	-	1/20	3/100	-
t_3^D	-	-	7/100	1/100

Table: Consumption

Period = 200.

Steady state for small batches

Algorithm

	p_1	p_2	p_3	p_4
t_0^A	$7/200$	-	-	$9/200$
t_1^B	$3/100$	$1/20$	-	-
t_2^C	-	$1/20$	$3/100$	-
t_3^D	-	-	$7/100$	$1/100$

Table: Consumptions: *cons*

	p_1	p_2	p_3	p_4
t_0^A	7	-	-	9
t_1^B	6	10	-	-
t_2^C	-	10	6	-
t_3^D	-	-	14	2

Table: Integer consuptions: *consInt*

Steady state for small batches

- Initial steady state schedule S
  - P: period, P : LCM of the matrix denominators,
  - ρ: throughput, $\rho = a/b$, reduced fraction.
- Let P_{min} be the minimum possible period
  - $P_0 = b$,
  - $P_{min} = \alpha \times P_0, \alpha \in [1, P/b]$
  - Find P_{min}, that respect the constraints:
    - For lines L_j: $\sum_{i \in L_j} cons(i,j) = \rho$,
    - For columns C_i: $\sum_{j \in C_j} cons(i,j) \times w_{ij} < 1$

Steady state for small batches

Example

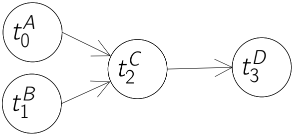


Figure: Job

		p_1	p_2	p_3	p_4
Type	A	20	∞	∞	20
	B	10	10	∞	∞
	C	∞	10	10	∞
	D	∞	∞	10	10

Table: Cost

	p_1	p_2	p_3	p_4
t_0^A	7/200	-	-	9/200
t_1^B	3/100	1/20	-	-
t_2^C	-	1/20	3/100	-
t_3^D	-	-	7/100	1/100

Table: Consumption

	p_1	p_2	p_3	p_4
t_0^A	1/25	-	-	1/25
t_1^B	1/50	3/50	-	-
t_2^C	-	1/25	1/25	-
t_3^D	-	-	3/50	1/50

Table: Modified consumption

Steady state for small batches

Algorithm

- Aim: maximize CD of the *ConsInt* matrix that respect the constraints,
- Optimisation problem with integers.
  - → Constraints programming with finite domains (swi-prolog)
  - Exponential complexity but the problem is small

Algorithm 1: `reducePeriod`(cons: Matrix, cost: Matrix) : Matrix

```
cd ← periodLength/throughputDenominator;
while cd > 1 do
    newCons : Matrix;
    if newCons ← reorganize(cons, cost, cd) then
        return newCons;
    else
        cd ← cd − 1;
    end
end
return cons;
```

Outline

Steady state for small batches

Metric

$$efficiency = makespan_o / makespan_r$$

$$efficiency = Batch\ size / (rate \times makespan_r)$$

- Steady state rate is optimal
  - $\longrightarrow$ time reference: N/ρ,
  - $\Leftrightarrow$ lower bound for optimal *makespan* ($makespan_o$),
- $makespan_r$: *makespan* of the algorithm.

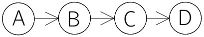
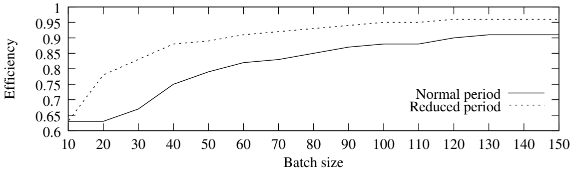


Figure: Job j_3

		p_1	p_2	p_3	p_4
Type	A	20	∞	∞	20
	B	10	10	∞	∞
	C	∞	10	10	∞
	D	∞	∞	10	10

Table: Grid G_3



- Period: 16/200 $\Rightarrow$ 4/50 (/4),

Figure: Execution of batches j_3 on G_2

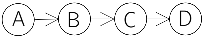
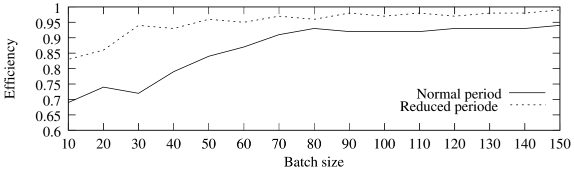


Figure: Job j_3

Type		p_1	p_2	p_3	p_4
	A	20	∞	20	20
	B	10	10	∞	10
	C	10	10	10	∞
	D	∞	10	10	10

Table: Grid G_4



- $96/1200 \Rightarrow 4/50$ (/24),

Figure: Execution of batches j_3 on G_3

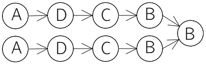


Figure: Job j_4

		p_1	p_2	p_3	p_4
Type	A	20	∞	∞	20
	B	10	10	∞	∞
	C	∞	10	10	∞
	D	∞	∞	10	10

Table: Grid G_3

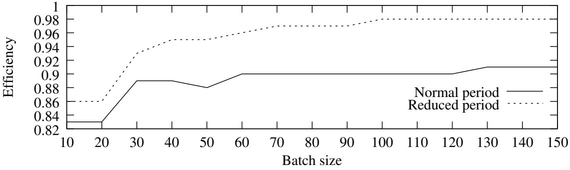


Figure: Execution of batches j_4 on G_2

- $48/1320 \Rightarrow 4/110$ (/12),
- starting: 194 instances,
- never in steady state.

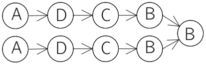


Figure: Job j_4

		p_1	p_2	p_3	p_4
Type	A	20	∞	20	20
	B	10	10	∞	10
	C	10	10	10	∞
	D	∞	10	10	10

Table: Grid G_4

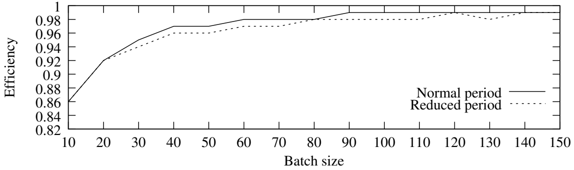


Figure: Execution of batches j_4 on G_3

- $12/330 \Rightarrow 4/110$ (/3),
- starting: 38 instances,
- 2 full periods for a batch of 150 jobs,
- Max efficiency difference: 1%.

10

Outline

Conclusion and futur works

- Conclusion:
  - Comparison of algorithms performances,
  - Minimal period while keeping steady state,
  - Large gain for small batches,
  - Not always possible.
- Futur works:
  - Keeping an optimal steady state:
    - $\longrightarrow$ Comparison of dependencies resolutions.
  - Deterioration of steady state:
    - $\longrightarrow$ Reduce the number of instances per periods.

The end

Thanks for your attention.

[diakite,nicod,philippe]@lifc.univ-fcomte.fr

Steady state for small batches

Reminder

- Initialization prepares all the dependencies needed before entering in steady state:
  - For each task or communication in the period:

    → execute all the preceding tasks/communications in the graph
- Ending: finish all the remaining tasks/communications,

→ Common work with Loris Marchal (LIP)

→ Reduce the number of tasks computed in the starting and ending phases.

Dependencies resolution

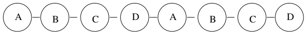


Figure: Job j_3

		p_1	p_2	p_3	p_4
Type	A	10	∞	∞	10
	B	10	10	∞	∞
	C	∞	10	10	∞
	D	∞	∞	10	10

Table: Grid G_5

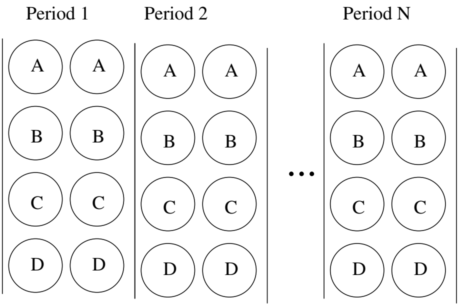
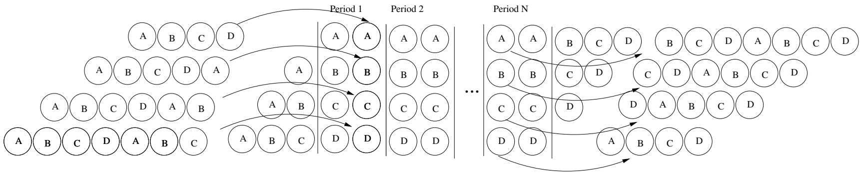


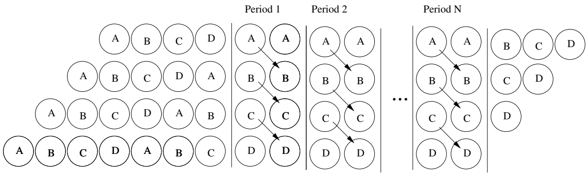
Figure: Execution of batches j_3 on G_5

Dependencies resolution

Initial Schedule:



Schedule with less dependencies:



Dependencies resolution

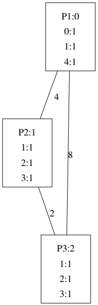


Figure: Platform

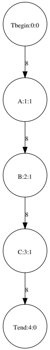
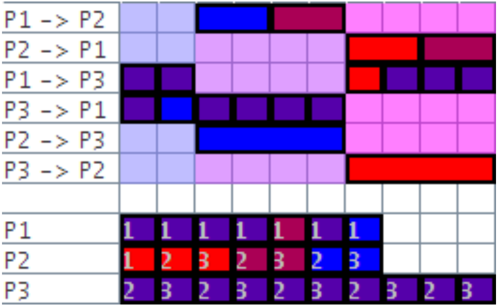


Figure: Job



Dependencies resolution

- 1 suppressed dependency = 1 subgraph less,
- Balance starting and ending,
- How to optimize dependencies resolution?
  - How to mesure the gain ?
  - The more ... the less
  - Are there better dependencies ?
- Max number of dependencies: two-partition
- Reoganize the periodic schedule: heuristics
- Find a good scheduling algorithm for starting and ending phases.
- Link number of jobs <=> period size

8