Periodic I/O Scheduling for Supercomputers

Guillaume Aupy[1], Ana Gainaru[2], Valentin Le Fèvre[3]

1 – Inria & U. of Bordeaux
2 – Vanderbilt University
3 – ENS Lyon & Inria

PMBS Workshop, November 2017
Slides available at `https://project.inria.fr/dash/`

IO congestion in HPC systems

Some numbers for motivation:

- Computational power keeps increasing (Intrepid: 0.56 PFlop/s, Mira: 10 PFlop/s, Aurora: 450 PFlop/s (?)).
- IO Bandwidth increases at slowlier rate (Intrepid: 88 GB/s, Mira: 240 GB/s, Aurora: 1 TB/s (?)).

IO congestion in HPC systems

Some numbers for motivation:

- Computational power keeps increasing (Intrepid: 0.56 PFlop/s, Mira: 10 PFlop/s, Aurora: 450 PFlop/s (?)).
- IO Bandwidth increases at slowlier rate (Intrepid: 88 GB/s, Mira: 240 GB/s, Aurora: 1 TB/s (?)).

In other terms:

Intrepid can process 160 GB for every PFlop
Mira can process 24 GB for every PFlop
Aurora will (?) process 2.2 GB for every PFlop

Congestion is coming.

Burst buffers: the solution?

Simplistically:

- If IO bandwidth available: use it
- Else, fill the burst buffers
- When IO bandwidth is available: empty the burst-buffers.

If the Burst Buffers are big enough it should work

Burst buffers: the solution?

Simplistically:

- If IO bandwidth available: use it
- Else, fill the burst buffers
- When IO bandwidth is available: empty the burst-buffers.

If the Burst Buffers are big enough it should work right?

Burst buffers: the solution?

Simplistically:

- If IO bandwidth available: use it
- Else, fill the burst buffers
- When IO bandwidth is available: empty the burst-buffers.

If the Burst Buffers are big enough it should work right?

Average I/O occupation: sum for all applications of the volume of I/O transfered, divided by the time they execute, normalized by the peak I/O bandwidth.

Burst buffers: the solution?

Simplistically:

- If IO bandwidth available: use it
- Else, fill the burst buffers
- When IO bandwidth is available: empty the burst-buffers.

If the Burst Buffers are big enough it should work right?

Average I/O occupation: sum for all applications of the volume of I/O transfered, divided by the time they execute, normalized by the peak I/O bandwidth.

Given a set of data-intensive applications running conjointly:

- on Intrepid have a max average I/O occupation of **25%**

Burst buffers: the solution?

Simplistically:

- If IO bandwidth available: use it
- Else, fill the burst buffers
- When IO bandwidth is available: empty the burst-buffers.

If the Burst Buffers are big enough it should work right?

Average I/O occupation: sum for all applications of the volume of I/O transfered, divided by the time they execute, normalized by the peak I/O bandwidth.

Given a set of data-intensive applications running conjointly:

- on Intrepid have a max average I/O occupation of **25%**
- on Mira have an average I/O occupation of **120 to 300%**!

Previously in IO cong.

"Online" scheduling (Gainaru et al. IPDPS'15):

- When an application is ready to do I/O, it sends a message to an I/O scheduler;
- Based on the other applications running and a priority function, the I/O scheduler will give a **GO** or **NOGO** to the application.
- If the application receives a **NOGO**, it pauses until a **GO** instruction.
- Else, it performs I/O.

Previously in IO cong.

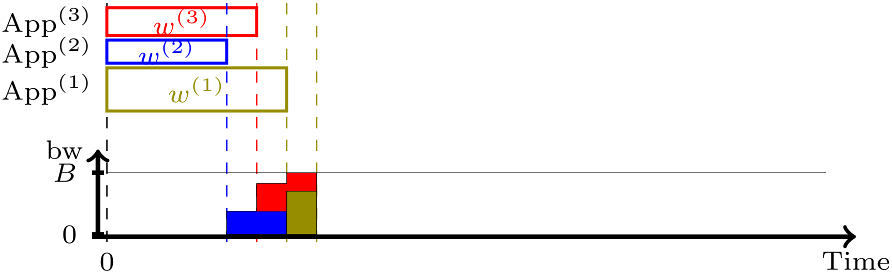
Previously in IO cong.

Previously in IO cong.

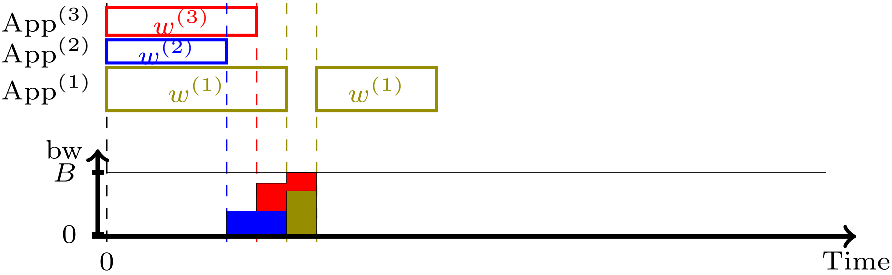
Previously in IO cong.

Previously in IO cong.

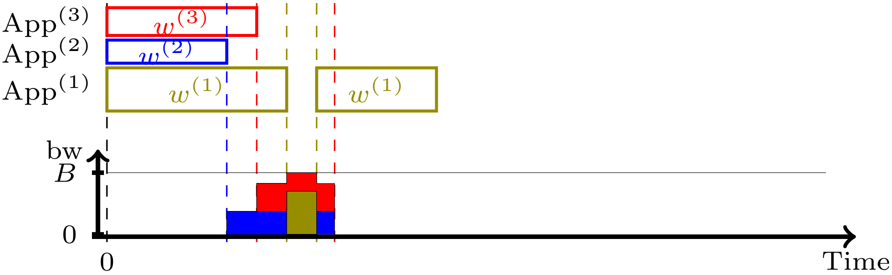
Previously in IO cong.



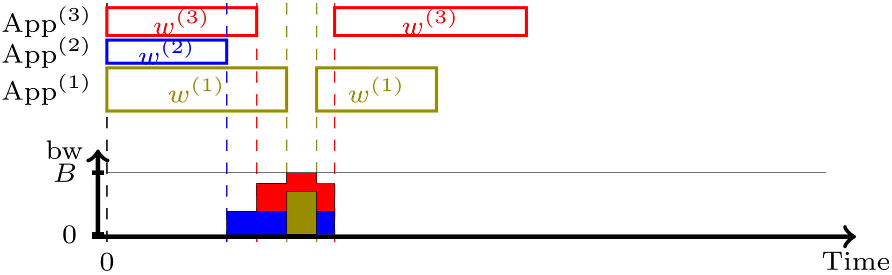
Previously in IO cong.



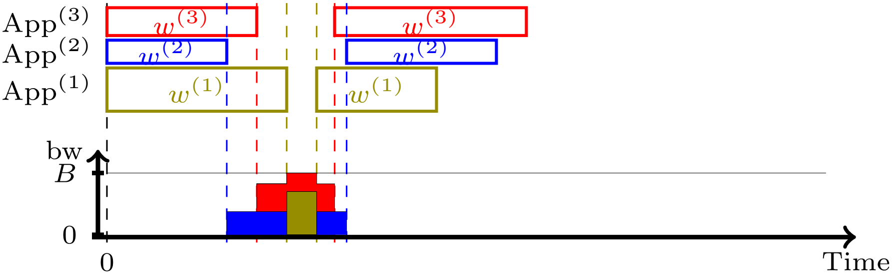
Previously in IO cong.



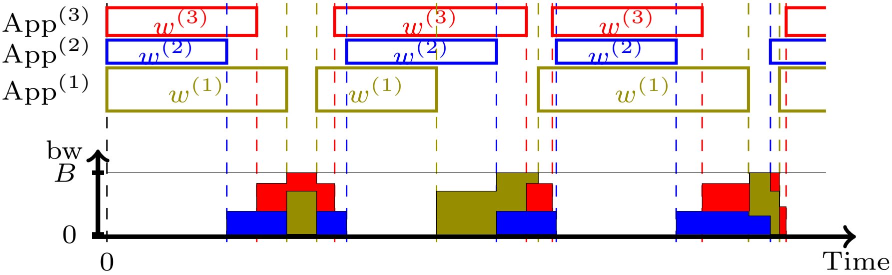
Previously in IO cong.



Previously in IO cong.



Previously in IO cong.



Approx 10% improvement in application performance with 5% gain in system performance on Intrepid.

This work

Assumption: Applications follow I/O patterns[1] that we can obtain (based on historical data for intance).

- We use this information to compute an I/O *time* schedule;
- Each application then knows its **GO/NOGO** information and uses it to perform I/O.

Spoiler: it works very well (at least it seems)

[1] *periodic pattern*, to be defined

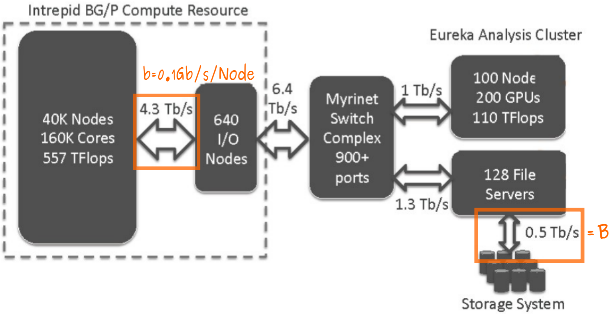
I/O characterization of HPC applis

Hu et al. 2016

1. Periodicity: computation and I/O phases (write operations such as checkpoints).
2. Synchronization: parallel identical jobs lead to synchronized I/O operations.
3. Repeatability: jobs run several times with different inputs.
4. Burstiness: short burst of write operations.

Idea: use the periodic behavior to compute periodic schedules.

- N unit-speed processors, equipped with an I/O card of bandwidth b
- Centralized I/O system with total bandwidth B

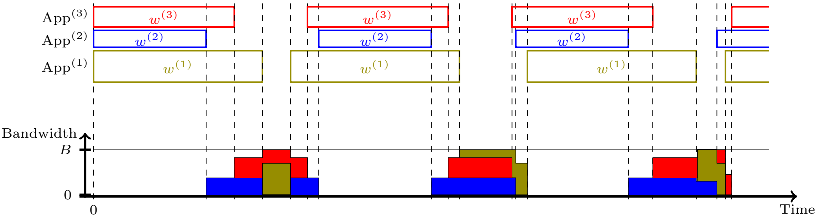


Model instantiation for the Intrepid platform.

K periodic applications **already scheduled in the system**: $\text{App}^{(k)}(\beta^{(k)}, w^{(k)}, \text{vol}_{\text{io}}^{(k)})$.

- $\beta^{(k)}$ is the number of processors onto which $\text{App}^{(k)}$ is assigned
- $w^{(k)}$ is the computation time of a period
- $\text{vol}_{\text{io}}^{(k)}$ is the volume of I/O to transfor after the $w^{(k)}$ units of time

$$\text{time}_{\text{io}}^{(k)} = \frac{\text{vol}_{\text{io}}^{(k)}}{\min(\beta^{(k)} \cdot b, B)}$$



If App$^{(k)}$ runs during a total time T_k and performs $n^{(k)}$ instances, we define:

$$\rho^{(k)} = \frac{w^{(k)}}{w^{(k)} + \text{time}_{\text{io}}^{(k)}}, \qquad \tilde{\rho}^{(k)} = \frac{n^{(k)} w^{(k)}}{T_k}$$

If App$^{(k)}$ runs during a total time T_k and performs $n^{(k)}$ instances, we define:

$$\rho^{(k)} = \frac{w^{(k)}}{w^{(k)} + \text{time}_{\text{io}}^{(k)}}, \qquad \tilde{\rho}^{(k)} = \frac{n^{(k)} w^{(k)}}{T_k}$$

SysEfficiency

maximize peak performance
(average number of Flops):

maximize $\frac{1}{N} \sum_{k=1}^{K} \beta^{(k)} \tilde{\rho}^{(k)}$.

Objectives

If App$^{(k)}$ runs during a total time T_k and performs $n^{(k)}$ instances, we define:

$$\rho^{(k)} = \frac{w^{(k)}}{w^{(k)} + \text{time}_{\text{io}}^{(k)}}, \qquad \tilde{\rho}^{(k)} = \frac{n^{(k)} w^{(k)}}{T_k}$$

SysEfficiency

maximize peak performance
(average number of Flops):

maximize $\frac{1}{N}\sum_{k=1}^{K} \beta^{(k)}\tilde{\rho}^{(k)}$.

Dilation

minimize largest slowdown
(fairness between users):

minimize $\max_{k=1..K} \frac{\rho^{(k)}}{\tilde{\rho}^{(k)}}$.

High-level constraints

- Applications are already scheduled on the machines: not (yet) our job to do it;

High-level constraints

- Applications are already scheduled on the machines:
  not (yet) our job to do it;
- We want the schedule information distributed over the applis:
  the goal is not to add a new congestion point;

High-level constraints

- Applications are already scheduled on the machines:
  not (yet) our job to do it;
- We want the schedule information distributed over the applis:
  the goal is not to add a new congestion point;
- Computing a full I/O schedule over all iterations of all applications would be too expensive (i) in time, (ii) in space.

High-level constraints

- Applications are already scheduled on the machines:
  not (yet) our job to do it;
- We want the schedule information distributed over the applis:
  the goal is not to add a new congestion point;
- Computing a full I/O schedule over all iterations of all applications would be too expensive (i) in time, (ii) in space.
- We want a minimum overhead for Applis users:
  otherwise, our guess is, users might not like it that much ☺.

High-level constraints

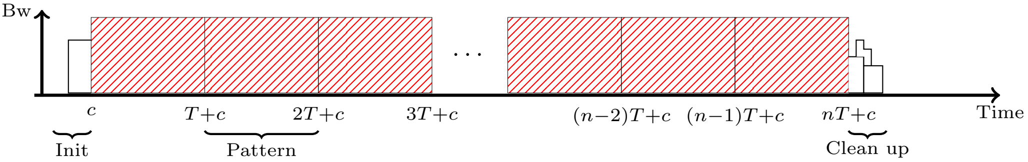
- Applications are already scheduled on the machines:
  not (yet) our job to do it;
- We want the schedule information distributed over the applis:
  the goal is not to add a new congestion point;
- Computing a full I/O schedule over all iterations of all applications would be too expensive (i) in time, (ii) in space.
- We want a minimum overhead for Applis users:
  otherwise, our guess is, users might not like it that much ☺.

High-level constraints

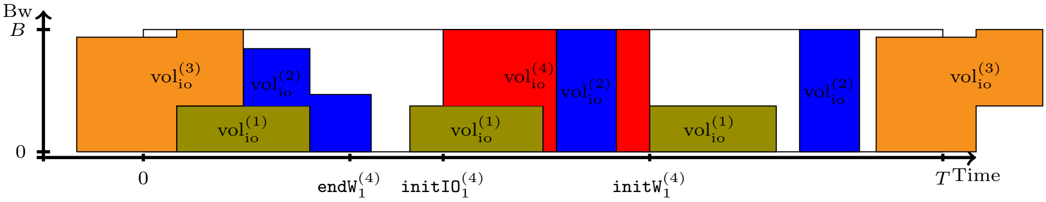
- Applications are already scheduled on the machines:
  not (yet) our job to do it;
- We want the schedule information distributed over the applis:
  the goal is not to add a new congestion point;
- Computing a full I/O schedule over all iterations of all applications would be too expensive (i) in time, (ii) in space.
- We want a minimum overhead for Applis users:
  otherwise, our guess is, users might not like it that much ☺.

We introduce Periodic Scheduling.

Periodic schedules

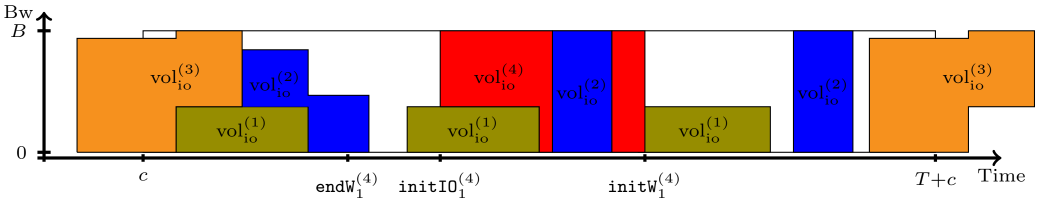


(a) Periodic schedule (phases)



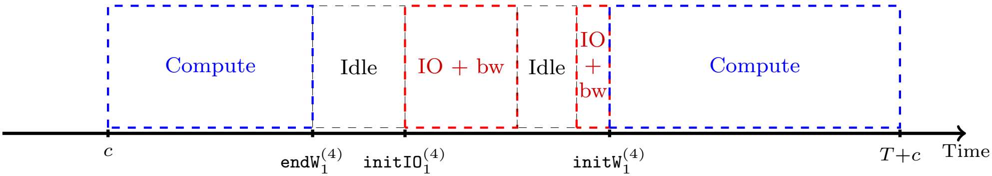
(b) Detail of I/O in a period/pattern

Time Schedule vs what Application 4 sees



- Distributed information
- Low complexity
- Minimum overhead

Time Schedule vs what Application 4 sees



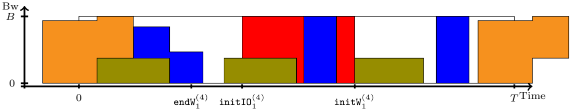
- Distributed information ✓
- Low complexity
- Minimum overhead ✓

Finding a schedule

Obj: algorithm with good SysEfficiency and Dilation perf.

Questions:

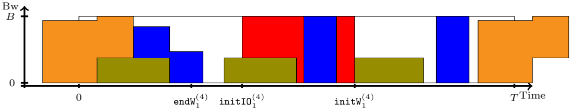
1. Pattern length T?
2. How many instances of each application?
3. How to schedule them efficiently?



Obj: algorithm with good SysEfficiency and Dilation perf.

Questions:

1. Pattern length T?
2. How many instances of each application?
3. How to schedule them efficiently?



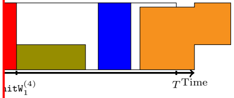
Answers:

1. Iterative search, exponential growth ($T_{\min}$ to $T_{\max}$).
2. Bound on the number of instances of each application $O\left(\frac{\max_k(w^{(k)}+\text{time}_{\text{io}}^{(k)})}{\min_k(w^{(k)}+\text{time}_{\text{io}}^{(k)})}\right)$.
3. Greedy insertion of instances, priority to applis with small Dilation.

Obj: algorithm with good SysEfficiency and Dilation perf.

Questions:

1. Pattern length T
2. How many instar
3. How to schedule



- Distributed information ✓
- Low complexity ✓
- Minimum overhead ✓

Answers:

1. Iterative search, exponential growth (T_{min} to T_{max}).
2. Bound on the number of instances of each application $O\left(\frac{\max_k(w^{(k)}+\text{time}_{\text{io}}^{(k)})}{\min_k(w^{(k)}+\text{time}_{\text{io}}^{(k)})}\right)$.
3. Greedy insertion of instances, priority to applis with small Dilation.

Announcement:

It's hard to find appli data $(w^{(k)}, \mathrm{vol}_{\mathrm{io}}^{(k)}, \beta^{(k)})$.
If you have some, let's talk ☺.

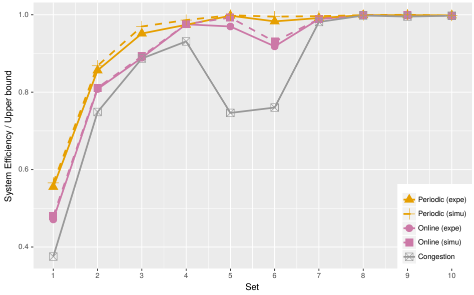
Model validation (I)

- Four periodic behaviors from the literature to instantiate the applications.
- Comparison between simulations and a real machine (Jupiter @Mellanox: 640 cores, $b = 0.01GB/s$, $B = 3GB/s$).
- We use IOR benchmark to instantiate the applications on the cluster (ideal world, no other communication than I/O transfers).

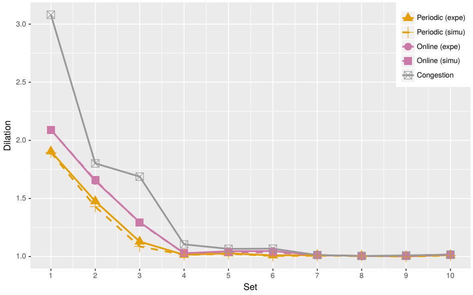
$\text{App}^{(k)}$		$w^{(k)}$ (s)	$\text{vol}_{\text{io}}^{(k)}$ (GB)	$\beta^{(k)}$
Turbulence1	(T1)	70	128.2	32,768
Turbulence2	(T2)	1.2	235.8	4,096
AstroPhysics	(AP)	240	423.4	8,192
PlasmaPhysics	(PP)	7554	34304	32,768

Set #	T1	T2	AP	PP
1	0	**10**	0	0
2	0	**8**	**1**	0
3	0	**6**	**2**	0
4	0	**4**	**3**	0
5	0	**2**	0	**1**
6	0	**2**	**4**	0
7	**1**	**2**	0	0
8	0	0	**1**	**1**
9	0	0	**5**	0
10	**1**	0	**1**	0

Model validation (II)



(c) SysEfficiency/Upper bound



(d) Dilation

The performance estimated by our model is accurate within 3.8% for periodic schedules and 2.3% for online schedules.

Results

- Periodic: our periodic algorithm;
- Online: the best performance (Dilation or SysEff resp.) of any online algorithm in Gainaru et al. Ipdps'15;
- Congestion: Current performance on the machine.

Set	Dilation	SysEff
1	-9.33%	+17.94%
2	-13.81%	+7.01%
3	-15.81%	+8.60%
4	-1.46%	+1.09%
5	-0.49%	+0.62%
6	-2.90%	+6.96%
7	-0.49%	+0.73%
8	-0.00%	+0.00%
9	-0.40%	+0.10%
10	-0.59%	+0.10%

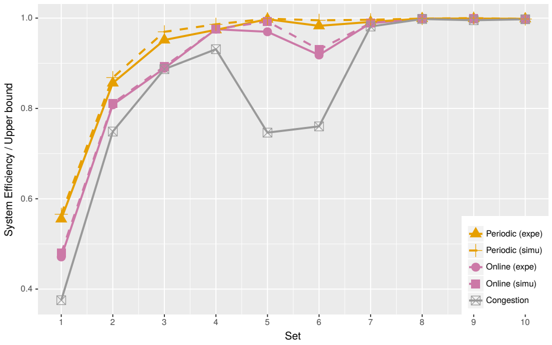


Figure: SysEfficiency/Upper bound

RESULTS

- Periodic: our periodic algorithm;
- Online: the best performance (DILATION or SYSEFF resp.) of any online algorithm in Gainaru et al. IPDPS'15;
- Congestion: Current performance on the machine.

Set	DILATION	SYSEFF
1	-9.33%	+17.94%
2	-13.81%	+7.01%
3	-15.81%	+8.60%
4	-1.46%	+1.09%
5	-0.49%	+0.62%
6	-2.90%	+6.96%
7	-0.49%	+0.73%
8	-0.00%	+0.00%
9	-0.40%	+0.10%
10	-0.59%	+0.10%

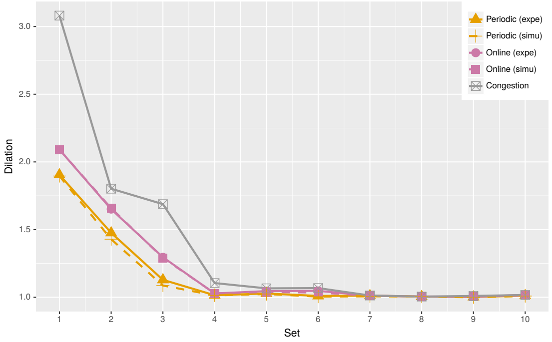
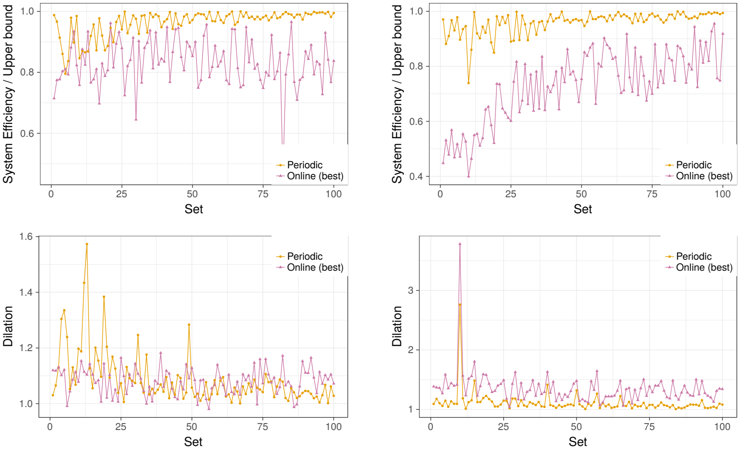


Figure: DILATION

More data

- We generate more sets of applications
- Simulate on instanciations of Intrepid and Mira.



Intrepid Mira

List of open issues / Future steps

- Study of robustness: what if $w^{(k)}$ and $\text{vol}_{\text{io}}^{(k)}$ are not exactly known?
- Integrating non-periodic application
- Burst-buffers integration/modeling
- Coupling application scheduler to IO scheduler
- Evaluation on real applications

Conclusion

- Offline periodic scheduling algorithm for periodic applications. Algorithm scales well (complexity depends on the unmber of applications, not on the size of the machine).
- Very good expected performance.
- Very precise model on very friendly benchmarks.
- Right now, more a proof of concept.

Paper, data, code: `https://github.com/vlefevre/IO-scheduling-simu`