Scheduling solutions for data-driven large-scale applications

Ana Gainaru, Hongyang Sun, Guillaume Aupy, Padma Raghavan

May 11, 2018

Introduction

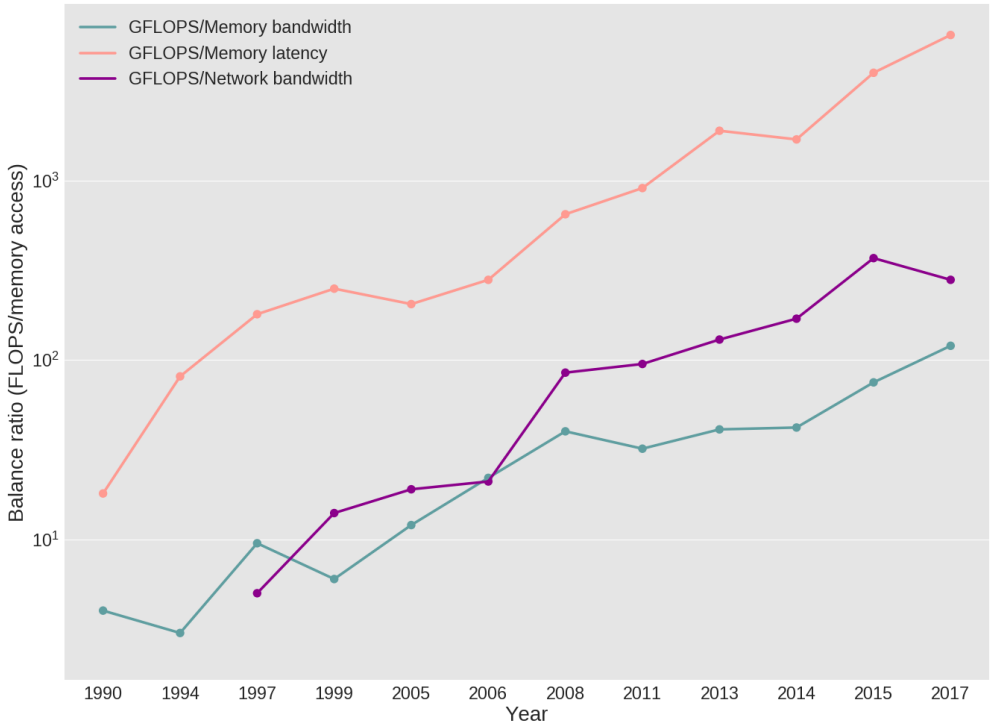


Figure: Memory access performance increase over time compared to the FLOPs increase in HPC systems (data from Top500)

Peak FLOPs to memory bandwidth - 14.2% increase rate per year
Peak FLOPs to memory latency - 24.5% increase rate per year
Peak FLOPs to network bandwidth - 22.3% increase rate per year

Introduction

- Computational power keeps increasing (Intrepid: 0.56 PFlops, Mira: 10 PFlops).
- Bandwidth between processors and file system increases at slowlier rate (Intrepid: 88 GB/s, Mira: 240 GB/s).
- Applications generate more and more data.

$$\Rightarrow \text{Memory bound executions}$$

Introduction

- Computational power keeps increasing (Intrepid: 0.56 PFlops, Mira: 10 PFlops).
- Bandwidth between processors and file system increases at slowlier rate (Intrepid: 88 GB/s, Mira: 240 GB/s).
- Applications generate more and more data.

$$\Rightarrow \text{Memory bound executions}$$

- Congestion
- Wasted cycles waiting for data
- Unpredictability (load imbalance, hardware changes)

Outline

1. Congestion
   - I/O scheduling
   - Simulations and experiments
2. Wasted cycles
   - Multi-resource scheduling
3. Unpredictability

I/O characterization of applications

[Work done with INRIA]

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. [Burst Buffers]

Idea: use the periodic behavior to compute periodic schedules.

Model

K periodic applications: $\text{App}^{(k)}(w^{(k)}, \text{vol}_{\text{io}}^{(k)}, \beta^{(k)})$.

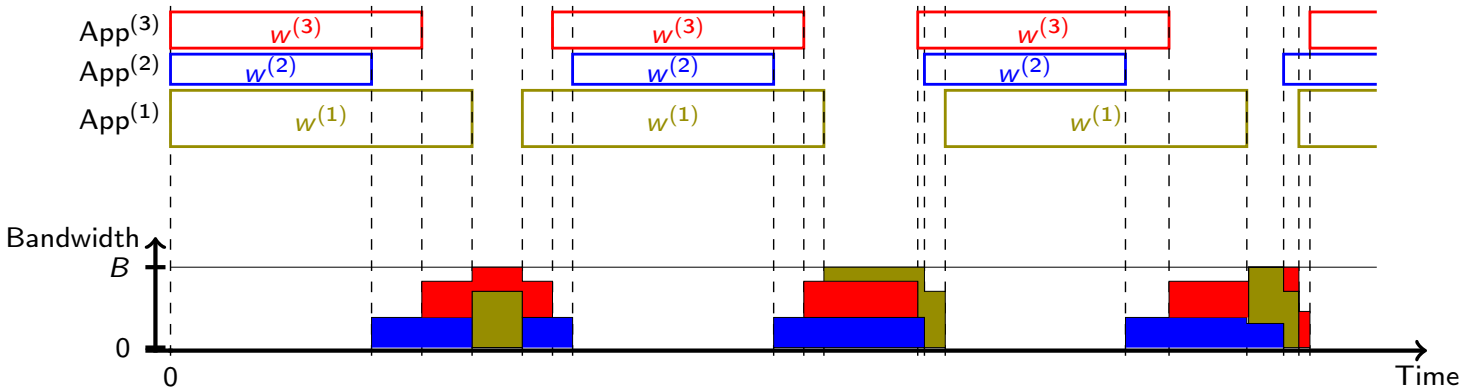


Figure: Scheduling the I/O of three periodic applications (top: computation, bottom: I/O).

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

Objectives

r_k: release time of App$^{(k)}$, d_k time when last instance is finished.

$$\tilde{\rho}^{(k)}(t) = \frac{\sum_{i \leq n^{(k)}(t)} w^{(k,i)}}{t - r_k} \qquad \rho^{(k)} = \frac{w^{(k)}}{w^{(k)} + \text{time}_{\text{io}}^{(k)}}$$

SysEfficiency: maximize peak performance:

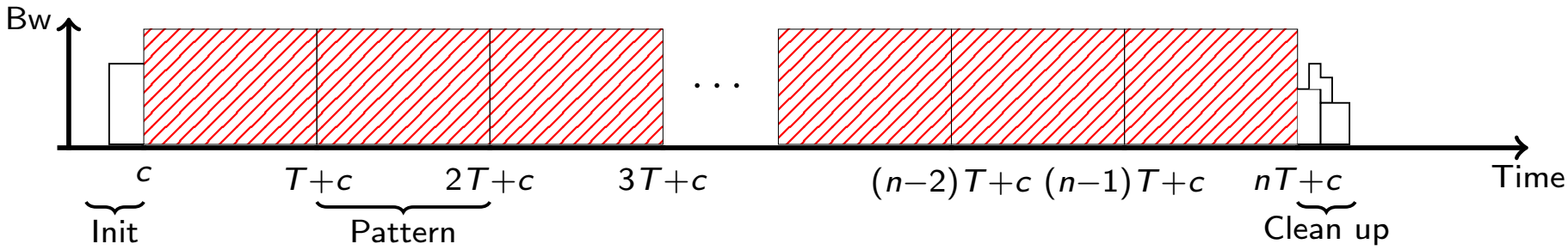
$$\text{maximize } \frac{1}{N} \sum_{k=1}^{K} \beta^{(k)} \tilde{\rho}^{(k)}(d_k). \tag{1}$$

Dilation: minimize largest slowdown:

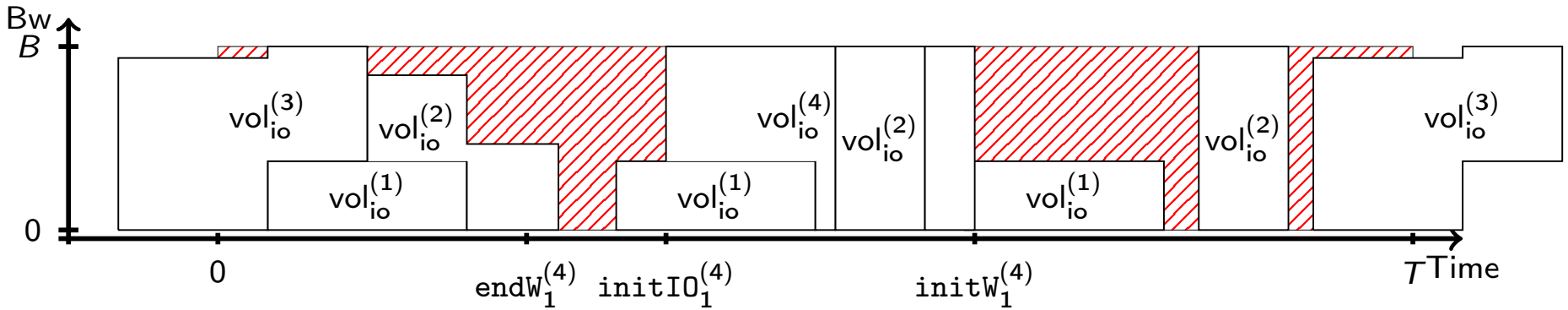
$$\text{minimize } \max_{k=1..K} \frac{\rho^{(k)}}{\tilde{\rho}^{(k)}(d_k)}. \tag{2}$$

Periodic schedules

Online algorithms add an overhead: lots of data transfers to a centralized system $\Rightarrow$ more congestion.



(a) Periodic schedule (phases)



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

Finding a schedule

The problem is NP-complete, given the number of instances.
We want an offline, efficient (SysEfficiency, Dilation) heuristic.

Questions:

1. Pattern length T?
2. How many instances of each application?
3. How to schedule them in a proper way?

Finding a schedule

The problem is NP-complete, given the number of instances.
We want an offline, efficient (SysEfficiency, Dilation) heuristic.

Questions:

1. Pattern length T?
2. How many instances of each application?
3. How to schedule them in a proper way?

Answers:

1. Iterative search, pattern length grows exponentially.
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. We greedily insert instances: Insert-In-Pattern.

Outline

1. Congestion
   - I/O scheduling
   - Simulations and experiments
2. Wasted cycles
   - Multi-resource scheduling
3. Unpredictability

Experiment setup

- Comparison between simulations and a real machine (32-node cluster for a total of 640 cores, $b = 0.01GB/s$, $B = 3GB/s$).
- We use periodic behaviors from the literature.

$App^{(k)}$	$w^{(k)}$ (s)	$vol_{io}^{(k)}$ (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

Table: Details of each application.

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

Table: Details of each set.

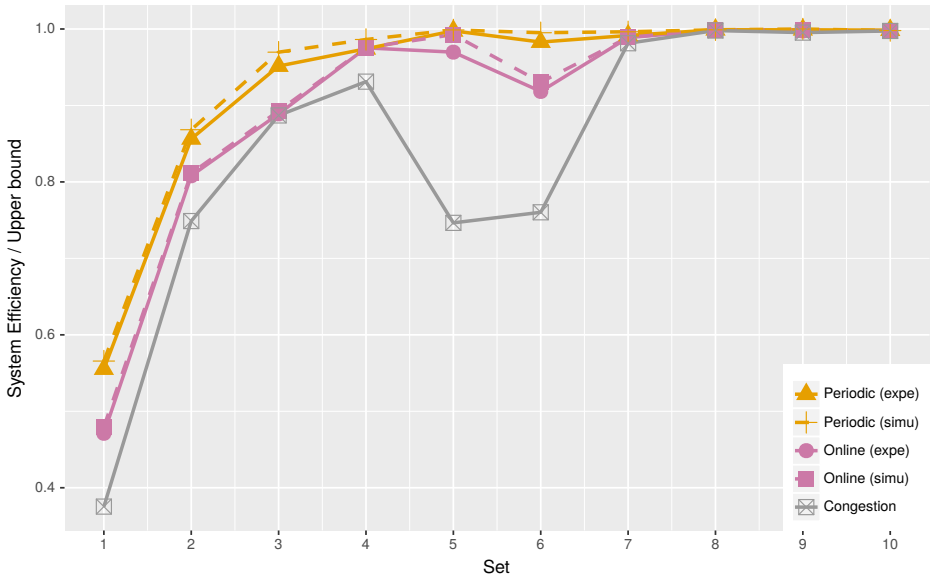
Simulation

We compare results for SysEfficiency and Dilation using either PerSched (with $T_{max} = 20\,T_{min}$ and $\varepsilon = 0.01$) or online heuristics from [?].
We also compare our results to what we get on a real machine without any scheduling algorithm.

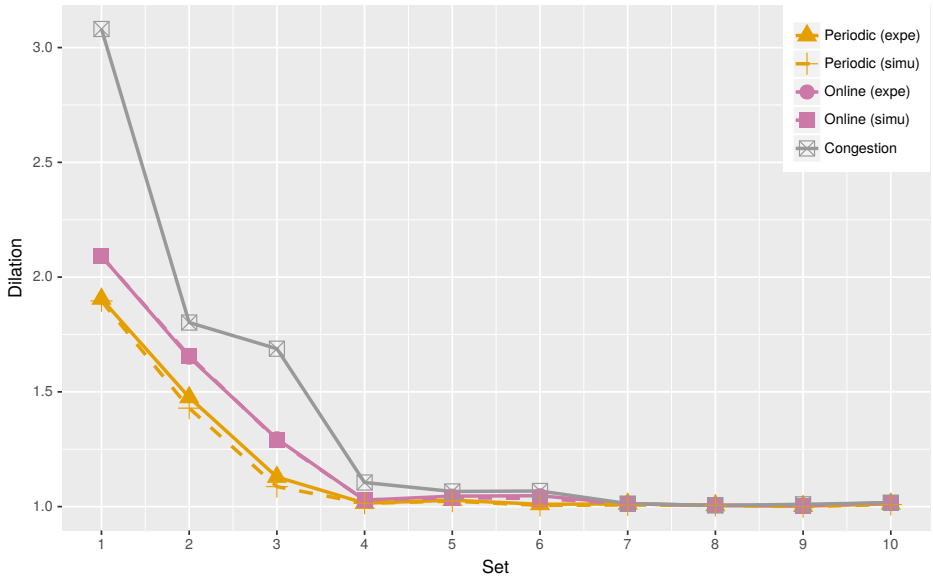
Set	Dilation	SysEff
1	-9.33%	+17.94%
2	-13.81%	+7.01%
3	-15.81%	+8.60%
4	-1.46%	+1.09%
5	-0.49%	+0.62%
6	-2.90%	+6.96%
7	-0.49%	+0.73%
8	-0.00%	+0.00%
9	-0.40%	+0.10%
10	-0.59%	+0.10%

Table: Difference between PerSched and online heuristics.

Results



(c) SysEfficiency/Upper bound



(d) Dilation

Figure: Performance for both experimental evaluation and theoretical (simulated) results. The performance estimated by our model is accurate within 3.8% for periodic schedules and 2.3% for online schedules.

Scaling results

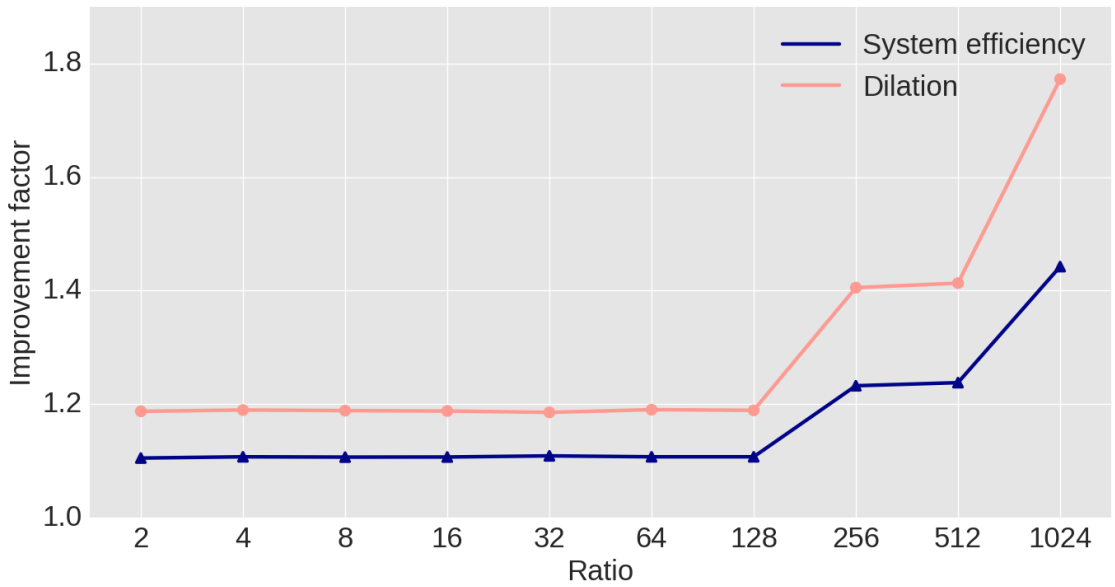


Figure: Comparison between online heuristics and periodic on synthetic applications, for different ratios of compute over I/O bandwidth.

Future

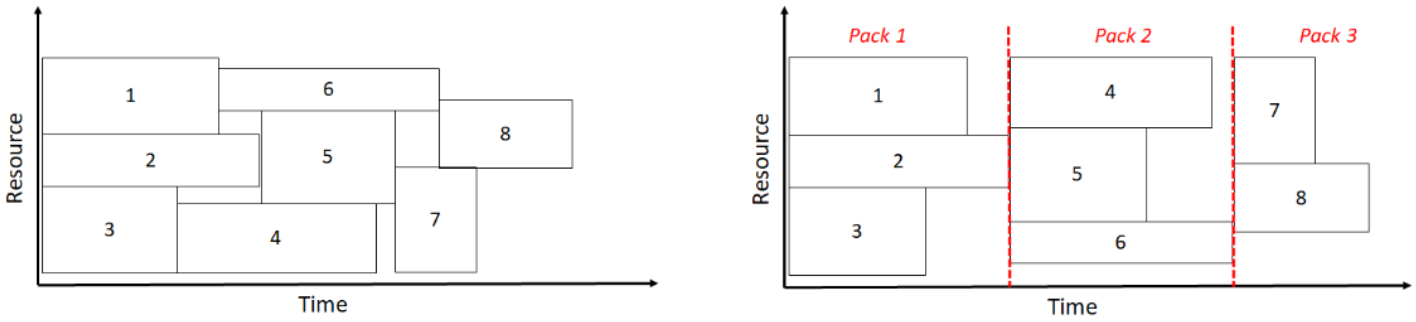
- Offline periodic scheduling: Up to 18% SysEfficiency, 16% Dilation improvement compared to online heuristics.
- We validated the model $\Rightarrow$ more simulations can be conducted without requiring actual experiments (at scale K stays the same, difference between $N.b$ and B increases).
- Deal with variability

Outline

1. Congestion
   - I/O scheduling
   - Simulations and experiments
2. Wasted cycles
   - Multi-resource scheduling
3. Unpredictability

Multi-resource scheduling

- Applications are modeled as independent tasks whose execution times vary depending on the different resources available to them
- Schedule multiple resources to each application in order to minimize makespan



(a) list scheduling

(b) pack scheduling

Figure: An example of list scheduling and pack scheduling for the same set of tasks (packs separated by dotted red lines)

Model

Two solutions

1. Two-phase approach, similarly to the one considered in single-resource scheduling
   - Phase 1: Determines a resource allocation matrix for the tasks
   - Phase 2: Constructs a rigid schedule based on the fixed resource allocation from the first phase

2. Transformation strategy that reduces the multi resource scheduling problem to the 1- resource scheduling problem, which is then solved and whose solution is transformed back to the original problem.

Phase 1

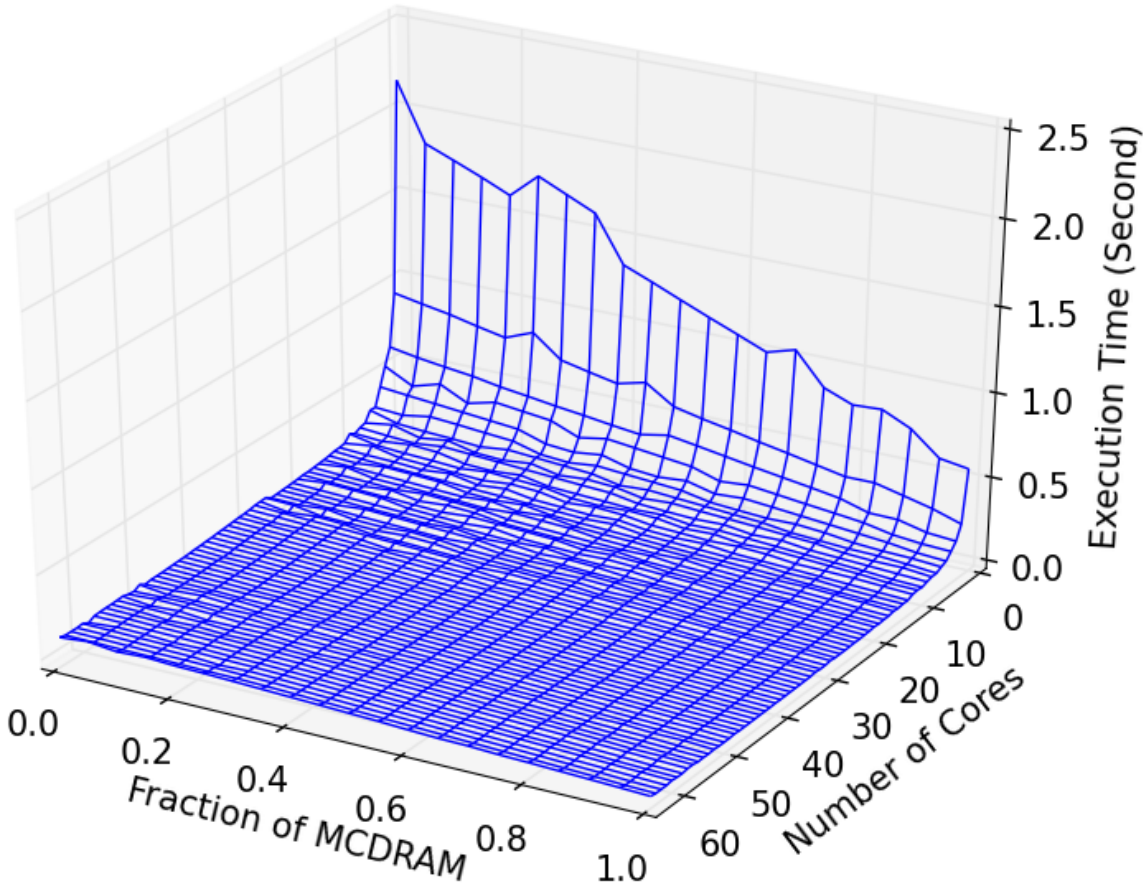


Figure: Profiles of the triad application in the Stream benchmark when its memory footprint occupies the entire fast memory

Phase 2

List scheduler

- Arrange the set of tasks in a list
- When a task completes, scan the list of remaining tasks in sequence and schedules the first one that fits (there is sufficient amount of resource to satisfy the task under each resource type)
- The complexity of the algorithm is $O(n^2 * d)$, since scheduling each task incurs a cost of $O(n * d)$ by scanning the taskList and updating the resource availability.

Pack scheduler

- Tasks are sorted in non-increasing order of execution times
- Tasks are assigned one by one to the last pack if they fit
- Otherwise, a new pack is created and the task is assigned to the new pack
- The complexity of the algorithm is $O(n(logn + d))$, which is dominated by the sorting of the tasks and by checking the fitness of each task in the pack

Results

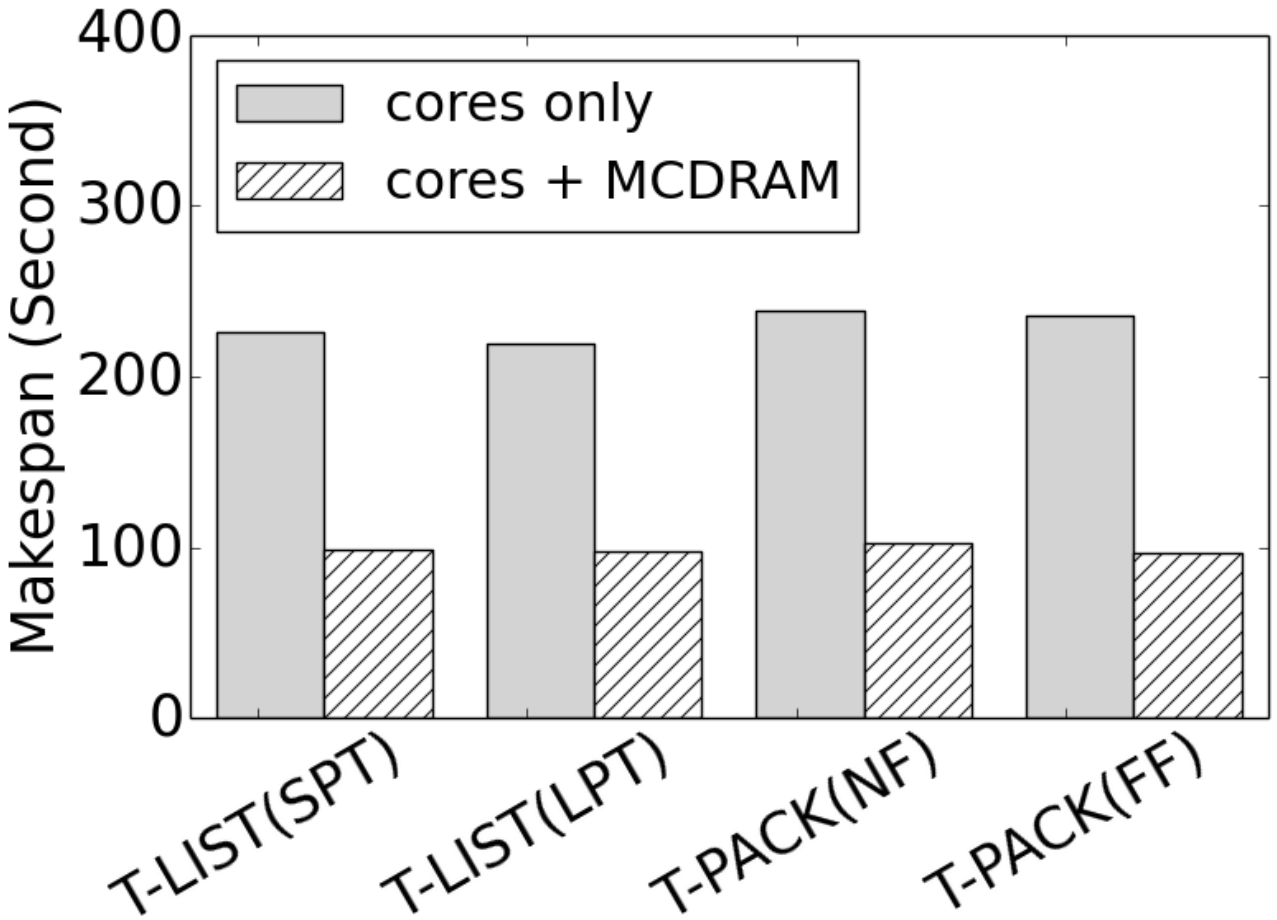


Figure: Makespan comparison with and without scheduling fast memory as a resource

Results

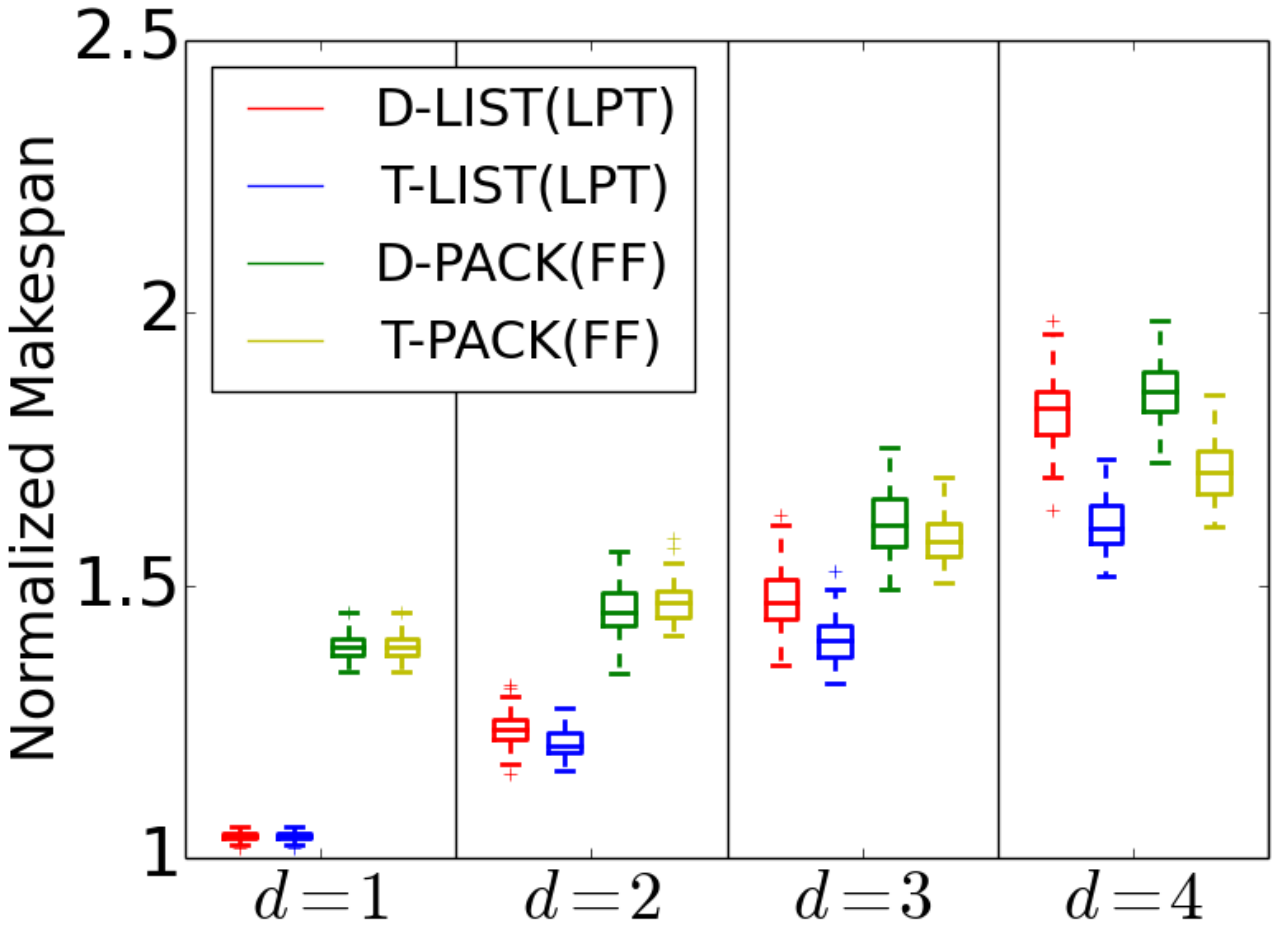


Figure: Performance with up to four different categories of resources

Future

- The list-based solutions to explore more effectively the gaps between successive task executions
  - The makespan difference between the two scheduling paradigms becomes smaller as more resources are included,
- Transform-based solutions are shown to be superior compared to the direct-based solutions in terms of both normalized makespan and algorithms' running times
- Schedulers that work on application models (include variability)

Outline

1. Congestion
   - I/O scheduling
   - Simulations and experiments
2. Wasted cycles
   - Multi-resource scheduling
3. Unpredictability