Resilient Scheduling of Moldable Parallel Jobs to Cope with Silent Errors

Anne Benoit[1], Valentin Le Fèvre[1,2], Lucas Perotin[1], Padma Raghavan[3], Yves Robert[1,4], Hongyang Sun[1,3,5]

1. ENS Lyon & INRIA, France
2. Barcelona Supercomputing Center, Spain
3. Vanderbilt University, USA
4. University of Tennessee Knoxville, USA
5. University of Kansas, USA

hongyang.sun@ku.edu

JLESC Workshop 2021 (Virtual)

What's This Work About?

On large-scale HPC platforms:

- Scheduling parallel jobs is important to improve application performance and system utilization
- Handling job failures/errors is critical as fault rates increase dramatically with size of system

This work combines parallel job scheduling and (silent) error handling for moldable parallel jobs running on large HPC platforms

Moldable Parallel Jobs

Moldable jobs can have processor allocations decided by the system dynamically, but once allocated the processors cannot be changed during the jobs' execution

- Moldable jobs can easily adapt to the amount of available resources (contrarily to rigid jobs)
- Moldable jobs are easy to design and implement (contrarily to malleable jobs)
- Many computational kernels in scientific libraries are provided as moldable jobs

Scheduling Model

- n moldable jobs to be scheduled on P identical processors
- execution time $t_j(p_j)$ of each job j $(= 1, 2, \ldots, n)$ is a function of processor allocation p_j $(= 1, 2, \ldots, P)$
- jobs are subject to arbitrary failure scenarios (i.e., # times to fail), which are unknown ahead of time (i.e., semi-online)
- minimize the makespan (i.e., successful completion time of all jobs)

Speedup Model

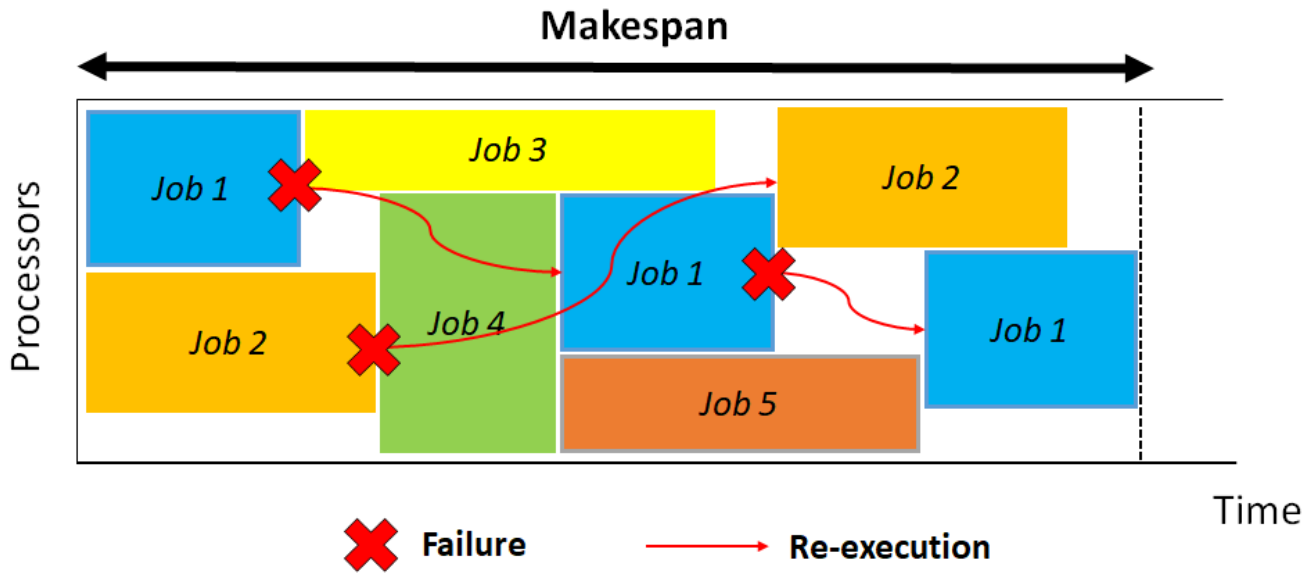
- Roofline model: $t_j(p_j) = \frac{w_j}{\min(p_j, \bar{p}_j)}$
  for some $1 \leq \bar{p}_j \leq P$
- Communication model: $t_j(p_j) = \frac{w_j}{p_j} + (p_j - 1)c_j$
  where c_j is the communication overhead
- Amdahl's model: $t_j(p_j) = w_j\left(\frac{1-\gamma_j}{p_j} + \gamma_j\right)$
  where γ_j is the inherently sequential fraction
- Mixed Model: $t_j(p_j) = \frac{w_j(1-\gamma_j)}{\min(p, \bar{p}_j)} + w_j\gamma_j + (p_j - 1)c_j$
  combining previous models
- Power Model: $t_j(p_j) = \frac{w_j}{p_j^{\delta_j}}$
  where $\delta_j \in [0, 1]$ is a constant parameter
- Monotonic model: $t_j(p_j) \geq t_j(p_j + 1)$ and $a_j(p_j) \leq a_j(p_j + 1)$
  i.e., execution time non-increasing and area is non-decreasing
- Arbitrary model: $t_j(p_j)$ is an arbitrary function of p_j

Failure Model

- Jobs can fail due to silent errors (or silent data corruptions)
- A lightweight silent error detector (of negligible cost) is available to flag errors at the end of each job's execution
- If a job is hit by silent errors, it must be re-executed (possibly multiple times) till successful completion

A failure scenario $\mathbf{f} = (f_1, f_2, \ldots, f_n)$ describes the number of failures each job experiences during a particular execution.

Example: $\mathbf{f} = (2, 1, 0, 0, 0)$ *for an execution of 5 jobs.*



Main Results

We proposed two resilient scheduling algorithms with analysis of approximation ratios* and simulation results.

1. A list-based scheduling algorithm, called LPA-LIST, and approximation results for common speedup models.
2. A batch-based scheduling algorithm, called BATCH-LIST, and approximation result for the arbitrary speedup model.
3. Extensive simulations to evaluate and compare (average and worst-case) performance of both algorithms against baseline heuristics.

*A scheduling algorithm ALG is said to be a c-approximation if its makespan is at most c times that of an optimal algorithm OPT, i.e., $T_{\text{ALG}} \leq c \cdot T_{\text{OPT}}$, for any job set under any failure scenario.

(1) LPA-LIST: Algorithm

Two-phase approach:

- **Phase 1**: Allocate processors to jobs using the Local Processor Allocation (LPA) strategy
  - Minimize a local ratio individually for each job based on the property of the job
  - The processor allocation remains unchanged for different execution attempts of the job
- **Phase 2**: Schedule jobs with fixed processor allocations using the List Scheduling (LIST) strategy.
  - Organize all jobs in a list according to any priority order
  - Schedule jobs one by one at earliest time (with backfilling if possible)
  - If job fails after an execution, insert it back into queue for rescheduling. Repeat until the job completes successfully

(1) LPA-LIST: Approximations

Speedup Model	Approximation Ratio
Roofline	2
Communication	3†
Amdahl	4
Mixed	6
Power	$\Theta(P^{1/4})$
Monotonic/Arbitrary	$\Theta(P^{1/2})$

- **Pros**: Simple to implement, and constant approximation for common speedup models
- **Cons**: Uncoordinated processor allocation, and high approximation for monotonic/arbitrary model

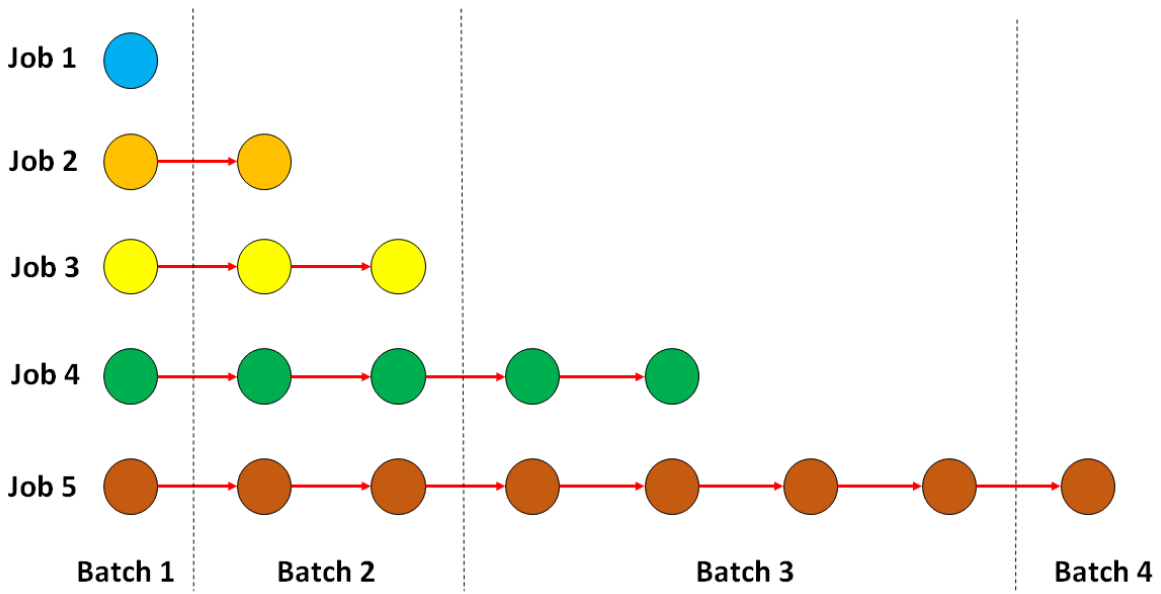
†This improves upon the previous best ratio (4) for this model obtained without failure considerations: [*Havill and Mao. Competitive online scheduling of perfectly malleable jobs with setup times, European Journal of Operational Research, 187:1126–1142, 2008*]

(2) Batch-List: Algorithm

Batched approach:

- Different execution attempts of the jobs are organized in batches, which are executed one after another
- In each batch k $(= 1, 2, \dots)$, all pending jobs are executed a maximum of 2^{k-1} times
- Uncompleted jobs in each batch will be processed in the next batch

Example: an execution of 5 jobs under a failure scenario $\mathbf{f} = (0, 1, 2, 4, 7)$



(2) Batch-List: Algorithm + Approximation

For jobs within each batch k $(= 1, 2, \dots)$:

- Near optimal processor allocations (within a factor of $1 + \epsilon$) using the Mt-Allotment algorithm[‡]
- Scheduling is done using the List strategy

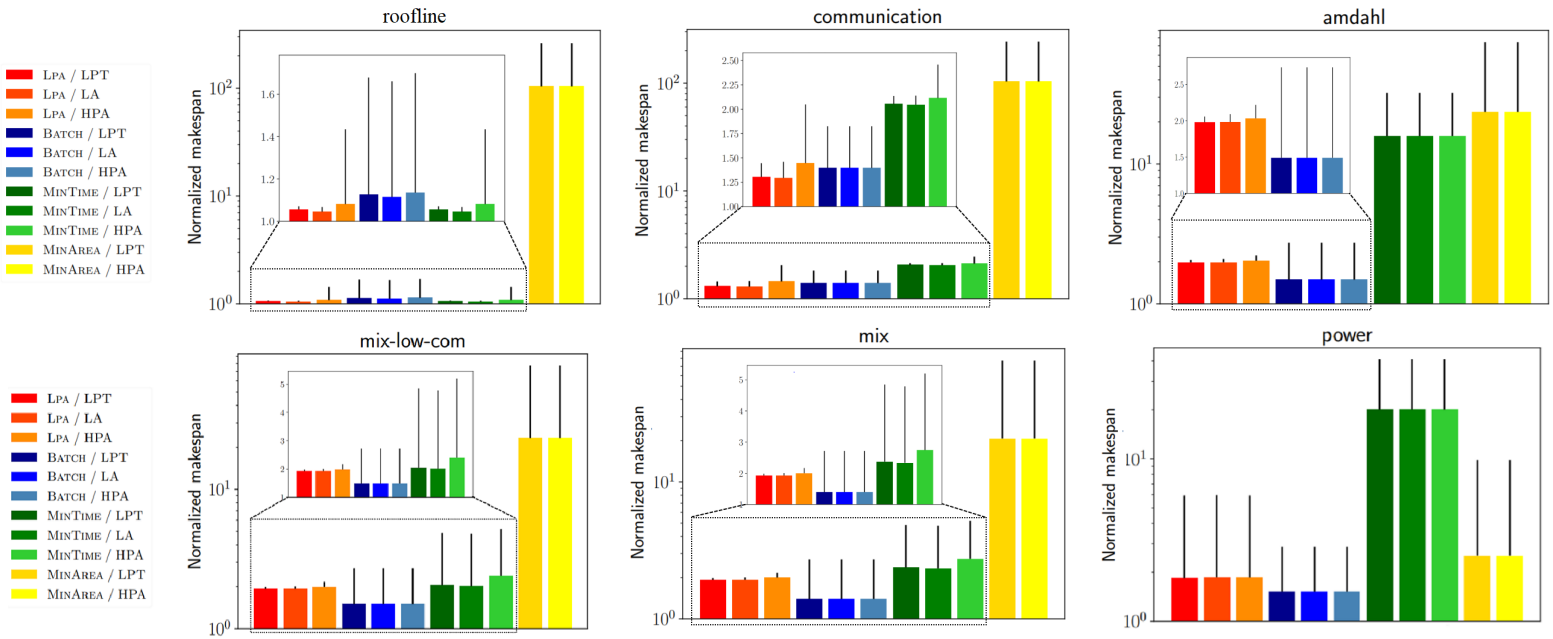
Approximation Result

The Batch-List algorithm is $\Theta((1 + \epsilon) \log_2(f_{\max}))$-approximation for arbitrary speedup model, where $f_{\max} = \max_j f_j$ is the maximum number of failures of any job in a failure scenario.

[‡] [*Lepère, Trystram, and Woeginger. Approximation algorithms for scheduling malleable tasks under precedence constraints. European Symposium on Algorithms, 2001*]

Simulation Results — with $P = 7500$, $n = 500$, and $\lambda = 10^{-7}$

- LPA performs better for roofline model, while BATCH performs better for the other models
- Both perform significantly better than MINTIME and MINAREA (baselines), which allocate processors to minimize execution time and area of each job, respectively
- Results hold for different job priority rules and parameter settings



Simulation Results — Summary

- Over the whole set of simulations, our best algorithm (LPA or BATCH) is within a factor of 1.6 of the optimal on average, and within a factor of 4.2 of the optimal in the worst case

Speedup Model		Roofline	Communication	Amdahl	Mix-low-com	Mix	Power
LPA	Expected	1.057	1.312	1.961	1.896	1.867	1.861
	Maximum	1.219	2.241	2.349	1.987	1.995	9.655
BATCH	Expected	1.158	1.434	1.529	1.548	1.571	1.549
	Maximum	1.999	2.449	2.874	3.674	4.164	3.975
MINTIME	Expected	1.057	2.044	15.567	2.810	2.704	20.386
	Maximum	1.219	2.666	49.795	12.611	27.174	61.726
MINAREA	Expected	114.079	122.199	23.594	16.875	9.686	2.571
	Maximum	1217.13	871.38	199.572	259.163	120.9	27.109

What's Next?

Theory:

- Considering fail-stop errors, and using checkpointing to improve efficiency of scheduling
- Resilient scheduling of workflows of jobs with dependencies

Practice:

- Analyzing average-case performance (e.g., when some failure scenarios occur with higher probability)
- Validating performance using realistic datasets with practical job speedup and failure traces