Resilient Scheduling of Moldable Jobs on Failure-Prone Platforms

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

1. Laboratoire LIP, ENS Lyon, France
2. Vanderbilt University, USA
3. University of Tennessee Knoxville, USA

lucas.perotin@ens-lyon.fr

Groupe de travail SCALE

What Is This Paper About?

On large-scale HPC platforms:

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

This paper combines job scheduling and failure handling for moldable parallel jobs running on large HPC platforms that are prone to failures.

Outline

Parallel job models

In the scheduling literature:

- **Rigid jobs**: Processor allocation is fixed.
- **Moldable jobs**: Processor allocation is decided by the system but cannot be changed.
- **Malleable jobs**: Processor allocation can be dynamically changed.

We focus on moldable jobs, because:

- They can easily adapt to the amount of available resources (contrarily to rigid jobs)
- They are easy to design/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

- Job j $(1 \leq j \leq n)$: Choose processor allocation p_j $(1 \leq p_j \leq P)$
- Execution time $t_j(p_j)$ of each job j is a function of p_j
- Area is $a_j(p_j) = p_j \times t_j(p_j)$
- Jobs are subject to arbitrary failure scenarios, which are unknown ahead of time (i.e., semi-online)
- Minimize the makespan (successful completion time of all jobs)

Speedup models

- Roofline model: $t_j(p_j) = \frac{w_j}{\max(p_j, \bar{p}_j)}$, 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: $t_j(p_j) = w_j(\frac{1-\gamma_j}{p_j} + \gamma_j)$,
  where γ_j is the inherently sequential fraction
- Mix model: $t_j(p) = \frac{w_j(1-\gamma_j)}{\min(p, \bar{p}_j)} + w_j\gamma_j + (p-1)c_j$,
  is the generalization of the three previous models
- Power model: $t_j(p) = w_j / p^{\delta_j}$,
  observed in some linear algebra applications
- 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

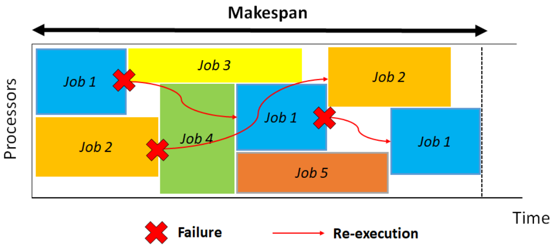
- Rigid jobs: p_j is fixed and hence execution time is t_j

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*



Problem complexity

- Scheduling problem clearly NP-hard (failure-free is a special case)
- A scheduling algorithm $\textsc{Alg}$ is said to be a *c-approximation* if its makespan is at most c times that of an optimal scheduler for all possible sets of jobs, and for all possible failure scenarios, i.e.,

$$T_{\textsc{Alg}}(\mathbf{f}, \mathbf{s}) \leq c \cdot T_{\textsc{Opt}}(\mathbf{f}, \mathbf{s}^*)$$

- $T_{\textsc{Opt}}(\mathbf{f}, \mathbf{s}^*)$ denotes the optimal makespan with scheduling decision $\mathbf{s}^*$ under failure scenario $\mathbf{f}$

Outline

1. The model
2. Moldable jobs
   - LPA-LIST Scheduling Algorithm
   - BATCH-LIST Scheduling Algorithm
   - Performance Evaluation

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 several 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.

Outline

1 The model

2 Moldable jobs
- LPA-LIST Scheduling Algorithm
- BATCH-LIST Scheduling Algorithm
- Performance Evaluation

(1) LPA-LIST Scheduling Algorithm

Two-phase scheduling approach:

- **Phase 1**: Allocate processors to jobs using the Local Processor Allocation (LPA) strategy.
  - Minimize a local ratio individually for each job as guided by the property of the LIST scheduling (next slide).
  - The processor allocation will remain unchanged for different execution attempts of the same 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 the jobs one by one at the earliest possible time (with backfilling whenever possible);
  - If a job fails after an execution, insert it back into the queue for rescheduling. Repeat this until the job completes successfully.

(1) LPA-LIST Scheduling Algorithm

Given a processor allocation $\mathbf{p} = (p_1, p_2, \ldots, p_n)$ and a failure scenario $\mathbf{f} = (f_1, f_2, \ldots, f_n)$:

- $A(\mathbf{f}, \mathbf{p}) = \sum_j a_j(p_j)$: total area of all jobs;
- $t_{\max}(\mathbf{f}, \mathbf{p}) = \max_j t_j(p_j)$: maximum execution time of any job.

Property of LIST Scheduling

For any failure scenario $\mathbf{f}$, if the processor allocation $\mathbf{p}$ satisfies:

$$A(\mathbf{f}, \mathbf{p}) \leq \alpha \cdot A(\mathbf{f}, \mathbf{p}^*) \ ,$$
$$t_{\max}(\mathbf{f}, \mathbf{p}) \leq \beta \cdot t_{\max}(\mathbf{f}, \mathbf{p}^*) \ ,$$

where $\mathbf{p}^*$ is the processor allocation of an optimal schedule, then a LIST schedule using processor allocation $\mathbf{p}$ is $r(\alpha, \beta)$-approximation:

$$r(\alpha, \beta) = \begin{cases} 2\alpha, & \text{if } \alpha \geq \beta \\ \frac{P}{P-1}\alpha + \frac{P-2}{P-1}\beta, & \text{if } \alpha < \beta \end{cases} \quad (1)$$

Eq. (1) is used to guide the local processor allocation (LPA) for each job.

(1) LPA-LIST Scheduling Algorithm

Approximation results of LPA-LIST for some speedup models:

Speedup Model	Approximation Ratio
Roofline	2
Communication	3†
Amdahl	4
Mix	6
Power	$\sqrt{P}$
Monotonic	$\Theta(\sqrt{P})$

Advantages and disadvantages of LPA-LIST:

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

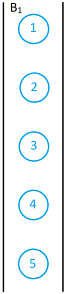
†For the communication model, our approx. ratio (3) improves upon the best ratio to date (4), which was obtained without any resilience considerations: [*Havill and Mao. Competitive online scheduling of perfectly malleable jobs with setup times, European Journal of Operational Research, 187:1126–1142, 2008*]

Outline

Batched scheduling approach:

- Different execution attempts of the jobs are organized in batches that 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 $\mathbf{f} = (1, 5, 1, 0, 3)$.

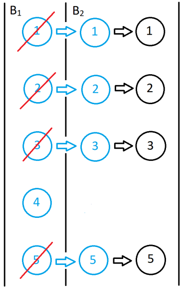


(2) Batch-List Scheduling Algorithm

Batched scheduling approach:

- Different execution attempts of the jobs are organized in batches that are executed one after another;
- In each batch k $(= 1, 2, \ldots)$, 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 $\mathbf{f} = (1, 5, 1, 0, 3)$.

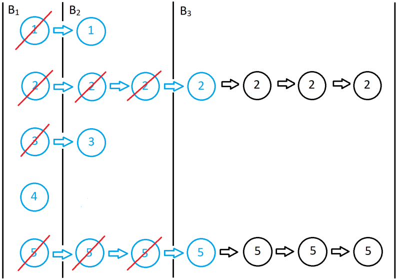


(2) Batch-List Scheduling Algorithm

Batched scheduling approach:

- Different execution attempts of the jobs are organized in batches that are executed one after another;
- In each batch k $(= 1, 2, \ldots)$, 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 $\mathbf{f} = (1, 5, 1, 0, 3)$.

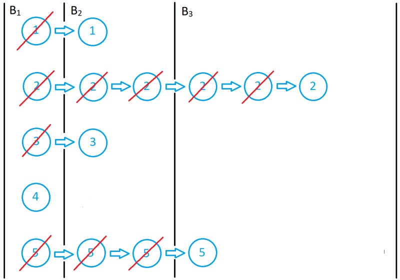


(2) BATCH-LIST Scheduling Algorithm

Batched scheduling approach:

- Different execution attempts of the jobs are organized in batches that 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 $\mathbf{f} = (1, 5, 1, 0, 3)$.



(2) Batch-List Scheduling Algorithm

Within each batch k:

- Processor allocations are done for pending jobs using the Mt-Allotment algorithm‡, which guarantees near optimal allocation (within a factor of $1 + \epsilon$).
- The maximum of 2^{k-1} execution attempts of the pending jobs are scheduling using the List strategy.

Approximation Result of Batch-List

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.

‡The algorithm has runtime polynomial in $1/\epsilon$ and works for jobs in SP-graphs/trees (of which a set of independent linear chains is a special case). [*Lepère, Trystram, and Woeginger. Approximation algorithms for scheduling malleable tasks under precedence constraints. European Symposium on Algorithms, 2001*]

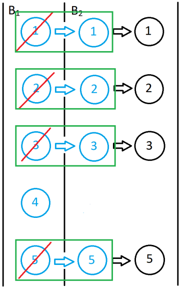
(2) Batch-List Scheduling Algorithm

Batched scheduling approach:

Approximation Result of Batch-List

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.

Example: an execution of 5 jobs under a failure scenario $\mathbf{f} = (1, 5, 1, 0, 3)$.



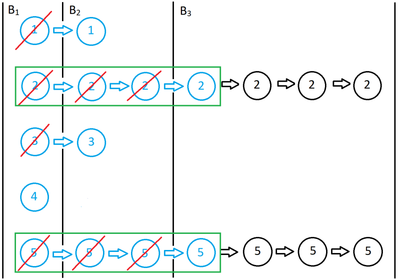
(2) BATCH-LIST Scheduling Algorithm

Batched scheduling approach:

Approximation Result of BATCH-LIST

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.

Example: an execution of 5 jobs under a failure scenario $\mathbf{f} = (1, 5, 1, 0, 3)$.



Outline

1. The model
2. Moldable jobs
   - LPA-LIST Scheduling Algorithm
   - BATCH-LIST Scheduling Algorithm
   - Performance Evaluation

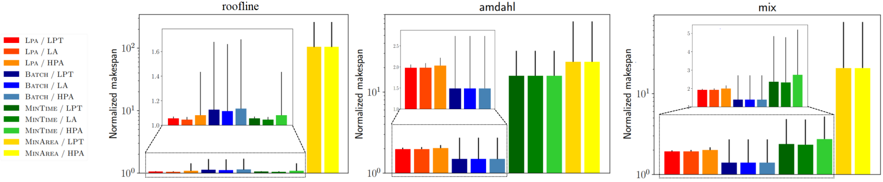
Performance Evaluation

We evaluate the performance of our algorithms using simulations.

- Synthetic jobs under three speedup models (Roofline, Communication, Amdahl) and different parameter settings;
- Job failures follow exponential distribution with varying error rate λ;
- Baseline algorithms for comparison:
  - MinTime: allocates processors to minimize execution time of each job and schedules jobs using List;
  - MinArea: allocates processors to minimize area of each job and schedules jobs using List.
- Priority rules used in List:
  - LPT (Longest Processing Time);
  - HPA (Highest Processor Allocation);
  - LA (Largest Area).

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

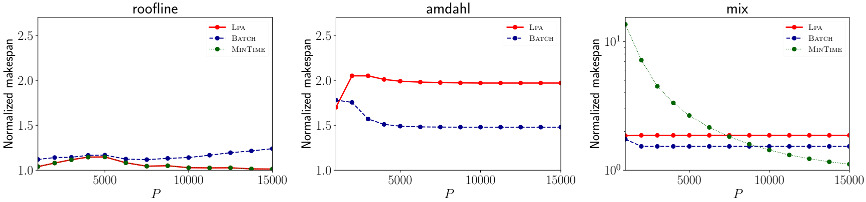
- LPA and BATCH generally perform better than the baselines;
- MINTIME performs well for Roofline model, is average for Mix model but performs badly for Amdahl's model;
- MINAREA performs the worst for all models;
- LPT and LA priorities perform similarly, but better than HPA.



(a) Roofline model (b) Amdahl's model (c) Mix model

Simulation Results — with varying number of processors P

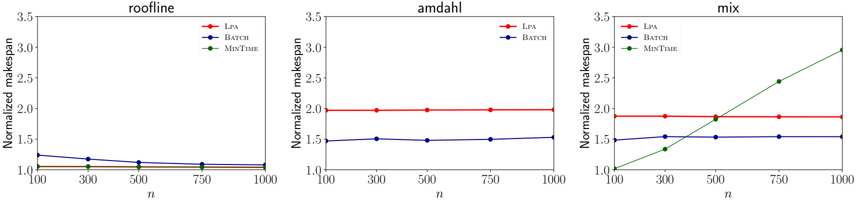
- In Roofline model, Lpa (and MinTime) has better performance, thanks to it simple and effective local processor allocation strategy.
- In Amdahl's model (where parallelizing a job becomes less efficient due to extra communication overhead), Batch has the best performance, thanks to its coordinated processor allocation.
- In Mix model, Batch is slightly better than Lpa while MinTime is near optimal with a lot of processors, but takes poor decisions with few processors;



(d) Roofline model (e) Amdahl's model (f) Mix model

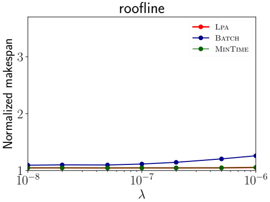
Simulation Results — with varying number of jobs n

- Same pattern of relative performance (as in last slide) for the three algorithms under the three speedup models;
- In Roofline and Communication models, having more jobs reduces number of available processors per job, thus reducing the total idle time between batches ⇒ performance gap between Batch and Lpa is decreasing (instead of increasing as in last slide).

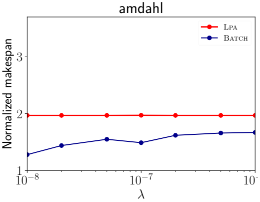


(g) Roofline model (h) Amdahl's model (i) Mix model

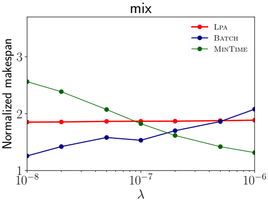
- Same pattern of relative performance (as in last two slides) for the three algorithms under the three speedup models;
- A higher error rate increases the number of failures per jobs, which has little impact on LPA and MINTIME, but degrades performance of BATCH (corroborating our approximation results).



(j) Roofline model



(k) Amdahl's model



(l) Communication model

- Both of our algorithms (LPA and BATCH) perform significantly better than the baseline (MINTIME and MINAREA);
- BATCH is alwats within a factor of 1.58 of the optimal on average, and within a factor of 4.164 of the optimal in the worst case.
  → Adapts nicely to any to all speedup model
- LIST also have good performance, and is better than BATCH in roofline and communication models.

Table: Summary of the performance for three algorithms.

Speedup Model		Roofline	Com	Amdahl	Mix	Power
LPA	Expected	1.057	1.312	**1.961**	1.867	1.861
	Maximum	1.219	2.241	2.349	1.995	**9.655**
BATCH	Expected	1.158	1.434	1.529	**1.571**	1.549
	Maximum	1.999	2.449	2.874	**4.164**	3.975
MINTIME	Expected	1.057	2.044	15.567	2.704	**20.386**
	Maximum	1.219	2.666	49.795	27.174	**61.726**

Conclusion

Take-aways:

- Future shared clusters demand simultaneous resource scheduling and resilience considerations for parallel applications;
- We proposed two resilient scheduling algorithms for moldable parallel jobs with provable performance guarantees;
- Extensive simulation results demonstrate the good performance of our algorithms under several common speedup models.

Future Work/Work in progress:

- Considering the use of checkpointing to improve efficiency of scheduling;
- Analysis of average-case performance of the algorithms (e.g., when some failure scenarios occur with higher probability);