Resilient scheduling of parallel jobs

Anne Benoit

LIP, Ecole Normale Supérieure de Lyon, France

Anne.Benoit@ens-lyon.fr

http://graal.ens-lyon.fr/~abenoit/

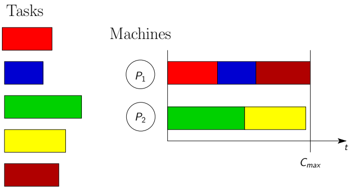
CCDSC, September 6-9, 2022

Motivation

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

We combine job scheduling and failure handling for moldable parallel jobs running on large HPC platforms that are prone to failures



Parallel job models

In the scheduling literature:

- **Rigid jobs**: Processor allocation is fixed by the user and cannot be changed by the system (i.e., fixed, static allocation)
- **Moldable jobs**: Processor allocation is decided by the system but cannot be changed once jobs start execution (i.e., fixed, dynamic allocation)
- **Malleable jobs**: Processor allocation can be dynamically changed by the system during runtime (i.e., variable, dynamic allocation)

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: $t_j(p_j) = w_j\left(\frac{1-\gamma_j}{p_j} + \gamma_j\right)$, where γ_j is the inherently sequential fraction
- 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

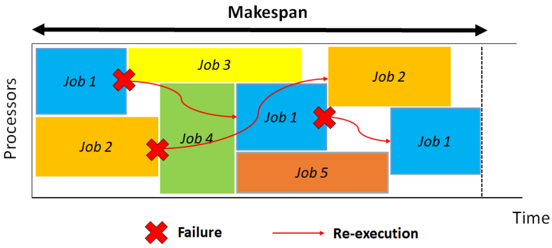
- Rigid jobs: p_j is fixed and hence execution time is t_j

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 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.,

$$T_{\text{ALG}}(\mathbf{f}, \mathbf{s}) \leq c \times T_{\text{opt}}(\mathbf{f}, \mathbf{s}^*)$$

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

Outline

Lower bounds

Rigid jobs: p_j is fixed and job j has execution time t_j

Optimal makespan has two lower bounds:

$$T_{\text{opt}}(\mathbf{f}, \mathbf{s}^*) \geq t_{\max}(\mathbf{f})$$
$$T_{\text{opt}}(\mathbf{f}, \mathbf{s}^*) \geq \frac{A(\mathbf{f})}{P}$$

- $t_{\max}(\mathbf{f}) = \max_{j=1 \ldots n}(f_j + 1) \times t_j$: maximum cumulative execution time of any job under $\mathbf{f}$
- $A(\mathbf{f}) = \sum_{j=1}^{n}(f_j + 1) \times a_j$: total cumulative area

List-based algorithm

Resilient list-based scheduling algorithm, and $O(1)$-approximations for any failure scenario:

- Extends classical batch scheduler that combines reservation and backfilling strategies
- Organizes all jobs in a list (or queue) based on some priority rule
- When a job completes: processors released; if error, inserted back in the queue; remaining jobs scheduled

Approximation results:

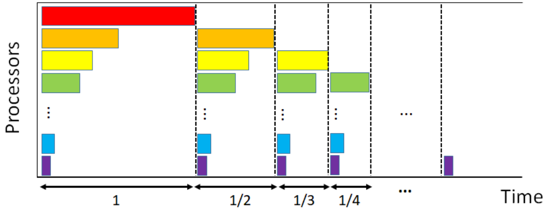
- 2-approximation using Greedy heuristic without reservation
- 3-approximation using Large Job First priority with reservation

The results nicely extend the ones without job failures

[TWY'92: J. Turek, J. L. Wolf, and P. S. Yu. Approximate algorithms scheduling parallelizable tasks. SPAA'92].

Shelf-based algorithm

Resilient shelf-based scheduling heuristic, but $\Omega(\log P)$-approx. for any shelf-based solution in some failure scenario, e.g.:

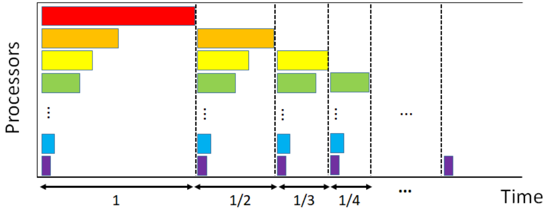


The result defies the $O(1)$-approx. result without failures [TWY'92]

Why not re-execute failed jobs within a same shelf?
Optimal on this example!

Shelf-based algorithm

Resilient shelf-based scheduling heuristic, but $\Omega(\log P)$-approx. for any shelf-based solution in some failure scenario, e.g.:

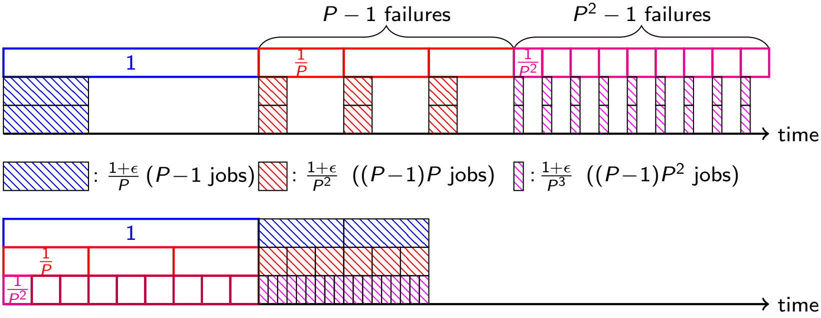


The result defies the $O(1)$-approx. result without failures [TWY'92]

Why not re-execute failed jobs within a same shelf?
Optimal on this example!

Shelf-fill variant: Fill shelfs when error detected

However, there exists a job instance and a failure scenario such that Shelf-fill with the LPT priority rule has an approximation ratio of $\Omega(P)$!



+ *Extensive simulation results of all heuristics using both synthetic jobs and job traces from the Mira supercomputer*

Outline

1. Main results for rigid jobs
2. Main results for moldable jobs
3. Simulation results
4. Conclusion

Main results for moldable jobs

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

(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 p_j will remain unchanged for different execution attempts of the same job j
- **Phase 2**: Schedule jobs with fixed processor allocations using the List Scheduling (LIST) strategy (as in rigid case)
  - 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[1]
Amdahl	4
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

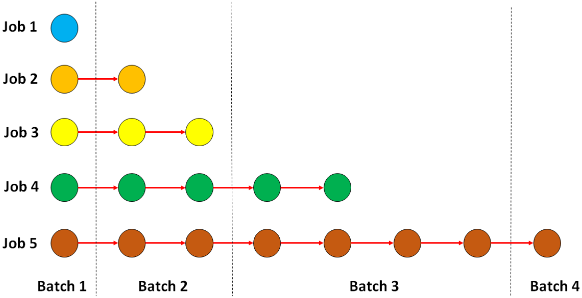
[1] For the communication model, our approx. ratio (3) improves upon the best ratio to date (4), which was obtained without any resilience considerations: [*Havill and Mao. Competitive online scheduling of perfectly malleable jobs with setup times, European Journal of Operational Research, 187:1126–1142,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

(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 $\mathbf{f} = (0, 1, 2, 4, 7)$



(2) BATCH-LIST scheduling algorithm

Within each batch k:

- Processor allocations are done for pending jobs using the MT-ALLOTMENT algorithm[2], 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

[2] 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*]

Outline

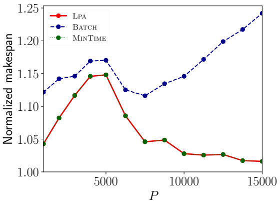
Performance evaluation

We evaluate the performance of our algorithms using simulations

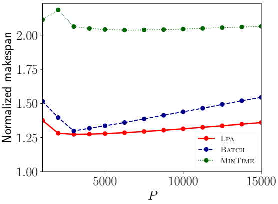
- Synthetic jobs under three speedup models (Roofline, Communication, Amdahl) and different parameter settings
- Job failures follow exponential distribution with varying error rate λ
- Baseline algorithm for comparison:
  - MinTime: allocate processors to minimize execution time of each job and schedule jobs using List
- Priority rules used in List:
  - LPT (Longest Processing Time)
- Results normalized by a lower bound (minimum possible total execution time of a job, minimum possible total area)

Simulation results — with varying number of processors P

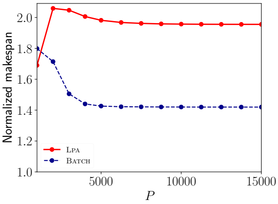
- In Roofline model, LPA (and MINTIME) has better performance, thanks to it simple and effective local processor allocation strategy
- In Communication model, BATCH catches up with LPA and performs better than MINTIME
- In Amdahl's model (where parallelizing a job becomes less efficient due to extra communication overhead), BATCH has the best performance, thanks to its coordinated processor allocation



(a) Roofline model



(b) Communication model



(c) Amdahl's model

Simulation results — Summary

- Both algorithms (LPA and BATCH) perform significantly better than the baseline MINTIME
- Over the whole set of simulations, our best algorithm (LPA or BATCH) is within a factor of 1.47 of the lower bound on average, and within a factor of 1.8 of the lower bound in the worst case

Summary of the performance for three algorithms (over loose bound)

Speedup model		Roofline	Communication	Amdahl
LPA	Expected	1.055	1.310	1.960
	Maximum	1.148	1.379	2.059
BATCH	Expected	1.154	1.430	**1.465**
	Maximum	1.280	1.897	**1.799**
MINTIME	Expected	1.055	2.040	14.412
	Maximum	1.148	2.184	24.813

Outline

Conclusion

Take-aways:

- Future HPC platforms demand simultaneous resource scheduling and resilience considerations for parallel applications
- Resilient scheduling algorithms for rigid and moldable parallel jobs with provable performance guarantees and good performance

Future work:

- Analysis of average-case performance of the algorithms
- Considering alternative failure models (e.g., fail-stop errors)
- Performance validation of algorithms using datasets with realistic job speedup profiles and failure traces

Thanks!!! And a few references:

- Benoit, Le Fèvre, Raghavan, Robert, Sun. Resilient scheduling heuristics for rigid parallel jobs. IJNC 2021.
- B, LF, Perotin, Ra, Ro, S. Resilient scheduling of moldable jobs on failure-prone platforms. Cluster 2020.
- B, LF, P, Ra, Ro, S. Resilient scheduling of moldable parallel jobs to cope with silent errors. IEEE TC 2021.