Online Scheduling with Redirection for Parallel Jobs

Adrien Faure, Giorgio Lucarelli, Olivier Richard, Denis Trystram

Online Scheduling with Redirection for Parallel Jobs

Adrien Faure[‡*], Giorgio Lucarelli[†], Olivier Richard[*], Denis Trystram[*]

[‡]*Atos*
Echirolles, France
[*]*Univ. Grenoble Alpes, CNRS, INRIA, Grenoble INP, LIG*
Grenoble, France
[†]*LCOMS, University of Lorraine*
Metz, France

***Abstract*—An important component of High Performance Computing (HPC) clusters is the job scheduling algorithm, which decides the allocation and the scheduling of the jobs in the system. Such scheduling algorithms need to be scalable to confront the growth both in size and in complexity of the modern clusters. We propose in this paper a new algorithm for scheduling parallel jobs with redirection. Specifically, our algorithm redirects the jobs whose execution affects significantly an important number of other jobs. A redirected job is stopped and restarted from the beginning in a dedicated part of the cluster. We show the effectiveness of our method through an intensive experimental campaign of simulations of production cluster log traces.**

***Index Terms*—Scheduling, Parallel jobs, Redirection.**

I. Introduction

The need for efficient automatic tools for managing the resources in large scale modern parallel and distributed platforms become more important as their complexity increases [1]. On the first hand, we need simple enough mechanisms able to deliver an allocation of jobs to the processors at scale, but on the other hand, such mechanisms should include all features needed to deal with specific situations [2].

We consider the problem of scheduling parallel jobs without preemption in multi-processor clusters using the concept of job redirection, where a job can be killed and restarted into a set of dedicated processors. Scheduling with redirection has been previously studied for sequential independent jobs where promising results have been presented [3]. As far as we know, this idea has never been presented in HPC. We validate our approach through an intensive simulation campaign based on the analysis of logs extracted from three production management systems. We compare our approach with a standard scheduling policy, namely FCFS with EASY backfilling [4], [5], and we show that scheduling parallel jobs with redirection improves the average bounded slowdown objective [6].

II. Definition and Notation

A job is a program submitted by a user to the queue of the computing platform. We characterize a job j by its *submission time* r_j, its *walltime* w_j, its number of *required resources* q_j, and its *processing time* p_j (known only at the end of the execution). The *waiting time* $wait_j$ of job j is the time it spends in the system after its release date and until its *start time* $start_j$. Then, we define the *flow time* F_j of the job j to be the total time it spends in the system, that is $F_j = wait_j + p_j$.

We consider the problem of scheduling a set of jobs into an HPC platform, that is of finding an allocation of each job to a set of free resources as well as assigning its starting time.

To evaluate our scheduling algorithm, we use the bounded slowdown (BSLD), which is defined for job j as follows:

$$BSLD_j = \max\left(\frac{F_j}{\max\left(p_j, \tau\right)}, 1\right) \tag{1}$$

where τ is a constant that prevents small jobs to have a high impact. In our study we set $\tau = 60$ seconds. Accordingly, we define the average and max slowdown as follows:

$$\begin{aligned} BSLD_{avg} &= \tfrac{1}{n} \sum_{j\in[n]} BSLD_j \ , \\ BSLD_{max} &= \max_{j\in[n]} (BSLD_j) \ . \end{aligned} \tag{2}$$

III. Scheduling parallel jobs with Redirection

A. General Description of the Redirection

Redirection is a generic mechanism that can be used in conjunction with any other scheduling algorithm. This is mainly because the redirection does not directly impact the scheduling decisions, instead it can independently choose to redirect a job to improve later scheduling decisions. The idea is to identify jobs that are worth to be redirected and move them into a dedicated part of the cluster — or another independent cluster. To set up the redirection, we propose to split the resources of the platform into two independent groups: the *principal group* and the *redirection group*. The size of the redirection group is determined by a parameter α, the *percentage of allocation*. Depending on the total number of available resources and on the properties of the jobs targeting the HPC platform, the size of the two groups needs to be adapted. The *principal group* contains $(1-\alpha)m$ processors while the redirection group contains the remaining αm processors of the platform, where m is the total number of processors. Both groups can be scheduled independently.

Identifying jobs that harm the overall cluster's performance is done by counting the number of submitted jobs during the execution of another job. This gives an estimation of the induced pressure by a job. If a job's counter exceeds a fixed

threshold θ, the system can choose to trigger a redirection. Each job running in the *principal group* holds the number of jobs submitted during its execution. When a job has been chosen to be redirected, it is killed and moved to the redirection group which is exclusively dedicated to these jobs, where redirection is no longer possible. As stated before, we focus on the case where no preemption is allowed, as a result, a redirected job is terminated and restarted from the beginning into the *redirection group*.

Algorithm 1 Scheduler using redirection.

```
1: procedure SCHEDULER(SP1,SP2, j)   ▷ SP1 and SP2 two scheduling policies, j new job
2:     if NbAvail(Resources) ≤ q_j and j 1st in queue then
3:         Start j
4:     else
5:         T ← Redirection(j, θ)   ▷ c.f. Algorithm 2
6:         if T is not None then
7:             Kill T
8:             add T to the queue of the redirection group
9:         end if
10:    end if
11:    Schedule principal group with SP1.
12:    Schedule redirection group with SP2.
13: end procedure
```

Algorithm 2 Algorithm for the redirection mechanism.

```
1: procedure REDIRECTION(j, θ, m' = αm)   ▷ j newly submitted job, θ threshold, m' redirection group size
2:     for k ∈ Runningjobs do
3:         if w_j >= w_k then ▷ restricts impact of huge jobs
4:             Counter_k ← Counter_k 1
5:         end if
6:     end for
7:     T ← {l ∈ Runningjobs, Counter_l > θ, r_j ≤ m'}
8:     if T ≠ ∅ then
9:         r ← conflictRedirectionPolicy(T)   ▷ tunes jobs selection in case of conflict.
10:        Reset every Counter
11:        return job r
12:    end if
13:    return None
14: end procedure
```

B. Set-up and execution

At the initialization, the redirection defines the scheduling policy for both *principal* and *redirection* groups. We set FCFS with EASY for both *principal* and *redirection* groups, and we activate the redirection only for the *principal group*.

Algorithm 1 describes how the redirection is integrated into a scheduling algorithm, while the redirection mechanism itself is detailed in Algorithm 2. The parameters of the latter one are the set of m processors, the percentage of allocation α, and the threshold θ used for tuning the redirection.

When a job j enters the system for the first time (r_j), it is assigned to the queue of the *principal group*. If the queue is empty and there are enough available resources, the job starts directly on the *principal group*. If the queue is not empty or there are not enough resource available to start the job immediately, the job waits and is assigned to the scheduling queue of the *principal group*. At this moment, the redirection increment the counters of all running jobs. Once all submitted jobs have proceeded, the redirection mechanism can decide to redirect a job (Algorithm 2). The decision to redirect is taken when one of the job's counter exceeds the input parameter θ — the redirection threshold. Note that to be illegible for redirection a job j needs to satisfy $q_j \leq \alpha m$. After one occurred, the counters of each running job are reinitialized.

In some cases, several jobs can exceed the threshold at the same time, the redirection mechanism determines which job will be redirected according to a *conflict resolution policy* (*e.g.* select the job with the greatest *walltime*).

Additionally, to control the impact of big jobs on small jobs, we set a filter preventing jobs to trigger redirection of smaller ones. The filter is configured such that if a job j is submitted, only the counter of the jobs k satisfying $q_k \geq q_j$ is increased.

IV. Experimental Setting

A. Simulation and Inputs

The redirection algorithm[1] has been integrated into the Batsim simulator [7], available as an open-source software. Batsim is a platform simulator, it does not simulate the scheduler. Instead, Batsim has a programming interface to communicate with an external scheduler. In our case, the scheduler is an external program implementing the EASY-Backfilling policy with the redirection mechanisms. The user submissions are also managed by Batsim so that the scheduler is not aware of a job until its release date (r_j), which is consistent with a system in production.

The performance of our algorithm on the metrics depends on a set of input parameters, namely, the size of the platform, the workload as well as the inputs related to the redirection mechanism itself. The latter are the threshold of redirection (θ) and the parameter α specifying the proportion of processors used for redirected jobs. To understand the behavior of the redirection, we ran thousands of simulations with several sets of parameters for the redirection to determine the best possible parameters combination.

B. Workloads Description

The workload is a static set of jobs that need to be scheduled during a simulation, and it is given as a simulation input. As in a production cluster, the scheduler only knows the *walltime* of the jobs. During the simulation, at the release time of a job, Batsim sends to the scheduler the given *walltime* of the job and the number of requested resources. The actual processing time remains known only by Batsim, therefore Batsim kills the

[1]The redirection mechanism implementation is available on-line with the data and the analysis https://gitlab.inria.fr/adfaure/evipar

jobs exceeding their walltime. Since the users overestimate the execution times of their jobs, we choose to give to the scheduler the exact job processing time.

The web site *parallel workload archive* (PWA)[2] hosts a consequent number of workload. To evaluate our algorithm, we run an intensive campaign of simulation on workloads extracted from logs of production clusters — namely, Curie, Intrepid and Ricc, all provided by PWA. For each of the production cluster logs, we extract 20 weeks of jobs. The extraction routine extract weeks (168 hours) with a mean utilization of at least 70%, ensuring that the traces are loaded enough to benefits from the redirection — indeed under-loaded traces do not represent any challenge and can be handled by any other traditional schedulers.

For the redirection, we split the cluster resources into two groups — the greater the parameter α, the more resources we reserve for the redirection. If the simulation is configured to use fewer processors in the *principal group* than the number of processors of the cluster the workload stem from, some jobs may be unable to execute. To cope with this problem and still execute every job of the original workload, we increase the number of resources based on the parameter α. Once the resource groups are created the *principal group* holds the same number of resources as the original cluster. Note that to ensure the fairness, we also give the extra resources to the algorithms without redirection.

C. Redirection Parameters: α and θ

As stated before, θ is the value of the redirection threshold and α determines the size of the *redirection group*. Both parameters need to be wisely tuned to find the best possible redirection performance. The threshold of redirection will impact the sensitivity of the redirection. A high value will rarely be reached leading to idle time for the *redirection group*, while setting θ to a low value will trigger a large number of redirections, leading to a *redirection group* overloaded. A large value of α allocates a lot of processors for the redirected jobs, leaving room for a lot of redirected jobs at the cost of reducing the size of the *principal group*. On the other hand, reducing α increases the number of jobs not eligible for the redirection.

Both parameters need to be carefully configured, in order to obtain the best performance for the redirection mechanism. To determine the best configuration, we ran simulations using all the combinations for $\theta \in \{1, 2, 5, 10, 15, 25, 50, 100, 125\}$ and $\alpha \in \{0.1, 0.15, 0.20, 0.25\}$, for each of the 20 extracted weeks. The next section describes how we select the best parameters.

It is worth mentioning that the best parameter configuration also depends on factors such as the distribution of the size of the jobs of a particular workload. For instance, the minimum job size allowed for the Intrepid cluster is 256, meaning that a *redirection group* of size smaller than 256 leads to a loss of resources without any positive effect on the performance.

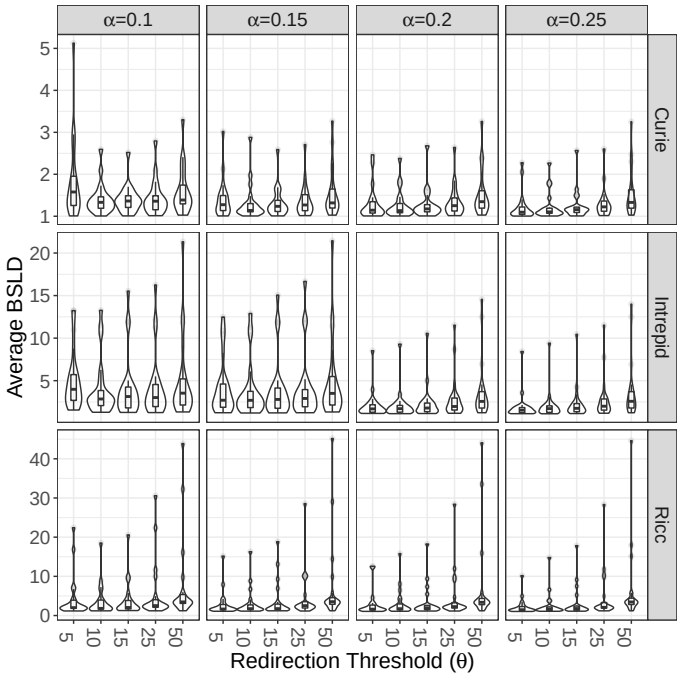

Fig. 1: **Lower is better** — For each of the 20 workloads extracted from Curie, we computed $BSLD_{avg}$. Each violin is created by 20 such workloads under a certain set of parameters. One violin depicts the distribution of the data, and contains a box (inside). The black horizontal line of the box is the median ratio and the red squares represent average ratio. The redirection threshold is in the x-axis, while the y-axis shows the observed metric. Each grid represents a different α value.

V. Experimental Results

A. Parameters tuning

In Fig. 1, the different columns of the grid represent the different values for parameter α, while each line shows the result of a particular cluster (Curie at the top, Intrepid in the middle and Ricc at the bottom). The x-axis represents the different threshold (θ) values we used. Considering θ, note first that small values generate a lot of redirection, and as a result, increases the load of on the *redirection group*. On the opposite side, high threshold values will never trigger any redirection, letting the processors in the redirection group idle. Redirection leads to an important improvement for $BSLD_{avg}$ when using small and moderate values of θ. Specifically, we observe that there is an optimal threshold value for $BSLD_{avg}$ which is between $[10, 50]$, $[5, 15]$ and $[5, 25]$ for respectively Curie, Intrepid and RICC.

Considering the allocation percentage (α), we first note that a small value may cause congestion in the redirection group, especially if there are a lot of redirected jobs. On the second hand, a very big value would improve the performance in the redirection group, but it is not realistic as we want to keep a reasonable total number of processors. We observe

[2]https://www.cs.huji.ac.il/labs/parallel/workload

that the impact of α to the *max flow-time* objective is very important due to the increased number of processors used in the redirection group as we increase α.

We can retain from the results of Fig. 1, that the choice of the parameters is very important for the performance of our mechanism while the two parameters are strongly related: in case of a small redirection threshold which implies a lot of redirected jobs, the size of the redirection group should be larger in order to execute them without important delays.

B. Comparison to EASY back-filling

In this section, we focus on the $BSLD_{avg}$ metric and we compare FCFS/EASY back-filling when using or not the proposed redirection mechanism. The y-axis of Fig. 2 shows the ratio of the corresponding objective function of FCFS with EASY back-filling policy (without redirection) over FCFS/EASY back-filling with redirection. The redirection improves the performance if the ratio is smaller than 1. We observe Fig. 2 that by appropriately choosing the parameters θ and α, the performance of FCFS/EASY back-filling can be improved by a factor of 10% for Curie, 20% for Intrepid and 40% for Ricc when considering the $BSLD_{avg}$ objective. On another hand, the impact of redirection is not so beneficial for $BSLD_{max}$ since the redirected jobs are restarted and the job with the $BSLD_{max}$ value tends to appear in the redirection group. However, there is always a couple of parameters for which both objectives are improved, *e.g.* $\alpha = 0.15$ and $\theta = 10$ for Curie where the average improvement for both $mean$ and $max\ BSLD$ is around 10%s.

VI. Conclusion

We proposed a mechanism based on redirection of parallel jobs that can be used on the top of other queuing scheduling and allocation policies. The selection of the jobs that should be redirected is done with respect to their impact on the performance of the other jobs in the queue. Although the redirection mechanism causes an additional load due to the kill-restart policy applied to the redirected jobs, it can improve the performance of the system for average bounded slow-down as shown in the intensive simulation campaign. Interestingly, the experimental results show that the redirection can exploit the benefits of preemption, even in cases where this is not explicitly allowed. Redirection presents interesting results for $BSLD_{avg}$ without harming the $BSLD_{max}$ which is a metric based on the satisfaction of the user. We showed that some workloads are more responsive to the redirection; further investigations are needed to understand these variations.

VII. Acknowledgment

The experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria.

We thank gracefully all the contributors of the Parallel Workloads Archive for making the workloads available.

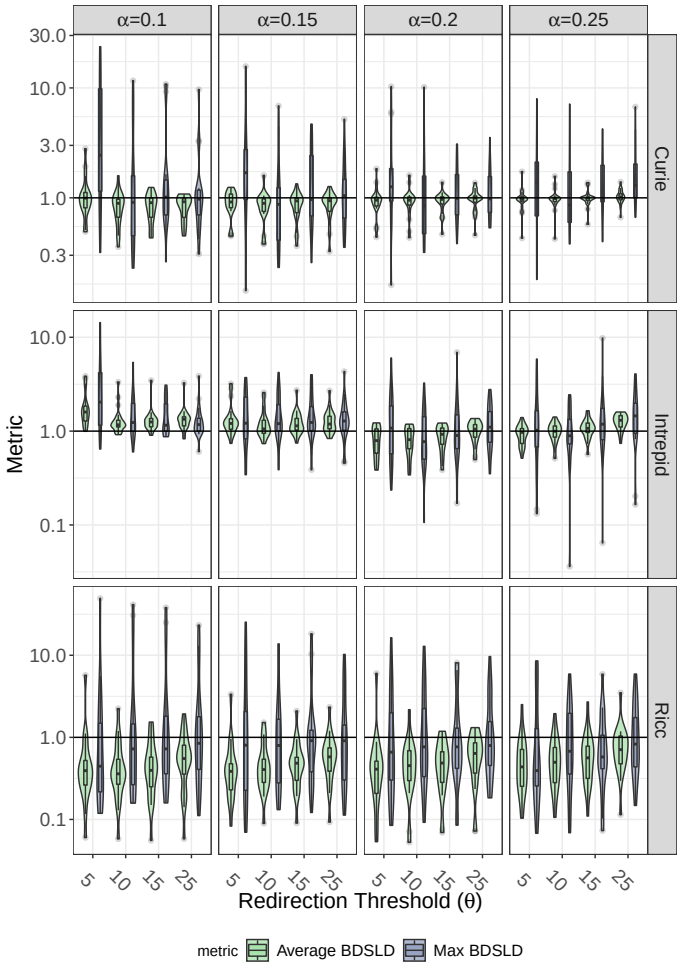

Fig. 2: **Lower is better, under horizontal line means that the redirection is more effective** — For each of the 20 workloads extracted from a cluster, we computed the ratio of $BSLD_{avg}$ without redirection over $BSLD_{avg}$ with redirection, and we did the same for the $BSLD_{max}$ (dark grey boxes). Each box is induced by 20 such ratios. The black line is the median ratio and the red square the average ratio. The figure presents the results for each cluster, the uppers figures deal with Curie, the middle with Intrepid while the bottom one concerns Ricc.

References

[1] J. J. Dongarra *et al.*, "The international exascale software project roadmap," *IJHPCA*, 2011.

[2] D. Glesser, "Road to exascale: Improving scheduling performances and reducing energy consumption with the help of end-users." Ph.D. dissertation, University of Grenoble, France, 2016.

[3] G. Lucarelli, F. M. Mendonca, and D. Trystram, "A new on-line method for scheduling independent tasks," in *CCGrid*, 2017.

[4] D. G. Feitelson and A. M. Weil, "Utilization and predictability in scheduling the IBM SP2 with backfilling," in *IPPS/SPDP*, 1998.

[5] É. Gaussier, J. Lelong, V. Reis, and D. Trystram, "Online tuning of easy-backfilling using queue reordering policies," *TPDS*, 2018.

[6] D. G. Feitelson, "Metrics for parallel job scheduling and their convergence," in *JSSPP*, 2001.

[7] M. Poquet, "Simulation approach for resource management," Ph.D. dissertation, Grenoble Alpes University, France, 2017.