A Survey on Fairness in Parallel Job Scheduling

Magne Tenstad
Department of Computer Science
NTNU
Trondheim, Norway
tenstad.magne@gmail.com

Job scheduling is a well-studied optimization problem with a wide range of applications, in areas such as manufacturing, transportation, healthcare and computing. In this survey we consider high-performance computing (HPC) environments, where computational tasks are distributed across multiple processors or servers. Traditionally, schedulers have optimized for performance-related metrics such as completion times and utilization. However, these often do not reflect the expectations of different jobs or users in the system. Hence, new metrics based on fairness have been introduced. We aim to answer the following research question:

RQ1 How has fairness been defined in parallel job scheduling?

First, we provide appropriate background on parallel job scheduling, performance metrics and scheduling procedures. Second, we list a number of different definitions of fairness. We categorize, evaluate and compare the definitions. Fairness in relation to performance is briefly discussed. Finally, we conclude our findings.

I. Background

We define the problem of parallel job scheduling, list some traditional performance metrics and present two noteworthy scheduling policies.

A. Parallel Job Scheduling

We consider the problem of *parallel job scheduling* as defined by Klusácek and Rudová [3]. The task is to schedule n jobs on a set of m machines. Each job requires one (in the case of a sequential program) or more machines (in the case of a parallel program). Additionally, we assume each job has a defined *arrival time*, *processing time*, and *owner*. We assume the processing time is perfectly known on arrival, and that jobs cannot be preempted. That is, once a job is started, it cannot be terminated before the entire processing time has elapsed. The reason to include job owners is to reason about fairness between different users of the system.

A solution to the job scheduling problem is a *schedule*. A schedule is a mapping from each job to a start time, and a set of machines for the job to be executed on. Figure 1 shows a schedule of five jobs on three machines. For a schedule to be valid, the jobs cannot overlap in time on any machine. The task of finding a valid schedule is simple in itself, but the problem becomes difficult when we optimize the schedule with respect to different metrics. Later we will introduce metrics of *fairness*, but first we consider some traditional metrics of *performance*.

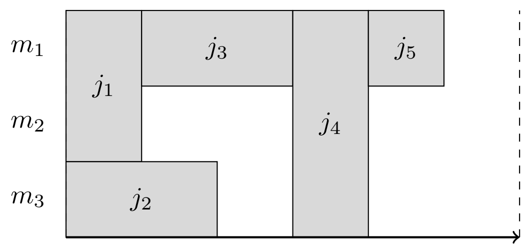

Fig. 1: Five jobs scheduled on three machines in a FCFS manner. j_1 requires two machines and j_4 requires three. The processing times PT of j_2 and j_3 is twice that of the rest.

B. Performance Metrics

Some traditional performance metrics include makespan, wait time, response time, slowdown and utilization. We denote start time as ST and processing time as PT.

- *Makespan (MS)* is the total time required to complete all scheduled jobs, measured from the start of the first job to the end of the last.
- *Wait Time (WT)* is the duration a job spends waiting before it begins execution.
- *Response Time (RT)* is the elapsed time between a job's submission and its completion, including both waiting and processing times. That is, $\text{RT} = \text{WT} + \text{PT}$
- *Slowdown (SD)* is the ratio of the actual response time of the job to the response time if executed without any waiting. That is, $\text{SD} = \text{RT}/\text{PT}$.
- *Utilization (U)* is the ratio of active resources to the total number of resources.

Makespan and utilization is calculated with respect to an entire schedule. However, the values of wait time, response time and slowdown are related to a specific job. Hence, when considering a schedule, we take the average of these values over all jobs in the schedule.

C. Scheduling Procedures

From the sets of jobs and machines, a schedule is determined by a *scheduling procedure*. Many scheduling procedures are extensions of a few key concepts. As points of reference, we briefly introduce First Come First Served and Shortest Job First, along with some of their properties.

1) First Come First Served (FCFS): With FCFS, all jobs are processed in the same order in which they arrive. This ensures that no job is *starved*, i.e. continuously delayed by other jobs. The downside is that FCFS has low utilization and worse response times compared to other procedures [8]. To mitigate this, some procedures fill the gaps of the schedule with out-of-order jobs, a method known as *backfilling*. For example, in Figure 1, j_5 could be scheduled to m_2 or m_3 in the gaps before j_4. This would be an example of *conservative* backfilling since it does not delay any scheduled jobs, as opposed to *aggressive* backfilling where that is allowed. Later, we will discuss the FCFS order as a metric of fairness.

2) Shortest Job First (SJF): With SJF, a job with shorter processing time will always be chosen over another job with longer processing time. This is an optimization of average response time. However, it has the potential for process starvation and the out-of-order processing is often considered unfair.

II. Definitions of Fairness

To discuss any metric of fairness, we have to define a set of identities for which to consider the fairness between. As per convention in the field of fair allocation, we refer to these identities as $agents$. In parallel job scheduling, we have three possible identities to consider as agents: *jobs*, *users*, and *machines*. Fairness is usually understood and represented as a job-related metric [6] [10] [8] [9] [7]. However, recent years have seen development in user-related fairness metrics [1] [3] [11] [5]. For theoretical interest, we briefly describe machine-related fairness and its connection to traditional performance metrics. The following metrics of job fairness and user fairness are presented in chronological order by publication date.

A. Job Fairness

Job fairness is the traditional approach to fairness in job scheduling.

1) Job Priority Fairness (JPF): Scheduling of jobs according to priorities has been done since at least as early as 1977 [6], and is still a common method [4]. One such criteria is that no job delays another job of higher priority. This begs the question of how to fairly assign priorities, but to our knowledge that is not often considered.

2) Fair Slowdown (FSD): Motivated by the need of a metric to compare conservative and aggressive backfilling procedures, Srinivasan et al. propose fair slowdown [10]. First, they present a strict definition of fairness: that no later arriving job should start before any earlier arriving job. However, they note that only an FCFS scheduling policy without backfilling would be fair under this definition. Hence, the condition is relaxed such that no job is started any later than when it would start under FCFS-Conservative [1]. In other words, although later arriving jobs may overtake queued jobs, it is not considered unfair because they do not delay them. Let FST_{FCFS} denote the fair start time under FCFS-Conservative. Then, a job's fair slowdown is given by

$$\text{FSD} = \frac{\text{FST}_{\text{FCFS}} - \text{WT} + \text{PT}}{\text{PT}}. \tag{1}$$

Fair slowdown is calculated for each job of the schedule. The total unfairness of the schedule is the percentage of jobs that have a higher slowdown than their fair slowdown.

3) Fair Start Time (FST): Sabin et al. return to the notion that no later arriving job should delay any earlier arriving job [8]. However, they argue that the relative performance of a "base scheme" (i.e. FCFS-Conservative, in the case of FSD) confounds the evaluation. They propose a scheme that is procedure-agnostic. To determine the fair start time for an incoming job j, a simulation of all preceding enqueued jobs is performed, using the same scheduling procedure that is to be tested. The resulting start time of j in this simulation is the fair start time. In other words, a job's fair start time is the start time if no job were to arrive after it.

Similar to the case of fair slowdown, a job is unfairly scheduled if its start time is after its fair start time. These delays are accumulated to consider the unfairness of an entire schedule. Furthermore, each job's machine usage is taken into account. Let m_i denote the number of machines required by job i. Then, the *Average Per Machine Miss Time* is given by

$$\text{APMMT} = \frac{\sum_{i \in \text{jobs}} \max\left(\text{ST}_i - \text{FST}_i, 0\right) \cdot m_i}{\sum_{i \in \text{jobs}} m_i}. \tag{2}$$

The differences between the works of Srinivasan et al. and Sabin et al. is the choice of scheduling procedure to calculate fair start times. In other aspects, like the definition of fair slowdown and average miss time, they are complementary.

4) Resource Equality (RE): So far, we have only considered fairness in the sense of job ordering, i.e. wait times. On the other hand, resource equality considers fair allocation of resources (i.e. machines) between jobs. The idea is to evenly divide the resources among all jobs which are active in the system [9]. By *resource time*, we refer to processing time multiplied by the number of consumed machines, $\text{PT} \cdot m$. Let $m_{\text{active}}(t)$ and $\text{jobs}(t)$ denote all active machines and all active jobs at timestep t, respectively. The deserved resource time for job j is given by

$$d_j = \int_{\text{arrival}_j}^{\text{departure}_j} m_j \cdot \min\left(\frac{m_{\text{active}}(t)}{\sum_{i \in \text{jobs}(t)} m_i}, 1\right) dt. \tag{3}$$

If more jobs require resources, a particular job's deserved resource time decreases. The total unfairness of a schedule is given by

$$\text{RE} = \frac{\sum_{i \in \text{jobs}} \max\left(d_i - \text{PT}_i m_i, 0\right)}{|\text{jobs}|}. \tag{4}$$

This is the average violation across all jobs. A violation occurs if a job j is given less resource time than what it deserves according to d_j.

[1]FCFS-Conservative is a FCFS-based procedure with conservative backfilling. It is used in place of plain FCFS because it has better utilization, improving start times, while still being considered fair.

5) Net Benefit (NB): Nguburi and Vliet state that the difference in actual and fair start times is not necessarily the most relevant comparison. They propose a net benefit approach, comparing the benefit of scheduling a job under one procedure versus another [7]. They choose wait time as the property for comparison, but note that other properties may be used. Instead of considering how two procedures a and b compare to an ideal fair scheduler, $\delta_a = t_{\text{fair}} - t_a$ and $\delta_b = t_{\text{fair}} - t_b$, they realize that a direct comparison is independent of t_{fair}:

$$\delta_a - \delta_b = (t_{\text{fair}} - t_a) - (t_{\text{fair}} - t_b) = t_b - t_a \tag{5}$$

From this, they choose FCFS as a base schedule procedure and look at how different schedulers compare to it: $t_i - t_{\text{FCFS}}$. We would argue that the reference to an ideal fair scheduler has been lost in this transformation, and that the end result is simply a comparison to a baseline. Their main contribution boils down to the consideration of the number of jobs that benefit from one scheduler instead of another.

B. User Fairness

Every metric of job fairness can be transformed into a user fairness metric by taking the average over all jobs for each user. However, Klusácek and Rudová argue that existing job-related fairness metrics are not satisfactory with respect to different users of a system [3].

1) Dominant Resource Fairness (DRF): Ghodsi et al. argue that cloud computing and multi-core processors has increased the need for allocation policies for environments with multiple resources and heterogeneous user demands [1]. They propose dominant resource fairness. It considers which type of the user's requested resources that make up the largest share of the available resources of that type. For example, some users require more CPU and other require more RAM. For fairness, DRF applies max-min fairness across users' dominant shares. Let user A request $(x \text{ CPU}, 4x \text{ GB})$ and user B $(3y \text{ CPU}, y \text{ GB})$ from a total of 9 CPUs and 18 GB RAM. The fair allocation is given by the solution to the following optimization problem:

$$\begin{aligned}
&\max\ x, y && && \text{(Maximize allocations)} \\
&\text{s.t.} && x + 3y \leq 9 && \text{(CPU constraint)} \\
& && 4x + y \leq 18 && \text{(RAM constraint)} \\
& && \frac{4x}{18} = \frac{3y}{9} && \text{(Equalize dominant shares)}
\end{aligned} \tag{6}$$

In this case, A gets (3 CPU, 12 GB) and B gets (6 CPU, 2 GB).

2) Normalized User Wait Time (NUWT): Klusácek and Rudová aim to keep good performance regarding classical criteria such as slowdown or wait time, while maintaining the fairness among different users of the system [3]. They propose normalized user wait time. Recall that WT denotes wait time and $\text{PT} \cdot m$ denotes resource time. The normalized user wait time of user u is given by

$$\text{NUWT}_u = \frac{\sum_{j \in \text{jobs}_u} \text{WT}_j}{\sum_{j \in \text{jobs}_u} \text{PT}_j m_j} \tag{7}$$

This is the average wait time, normalized by resource time. The normalization is used to prioritize less active users over those who utilize the system resources very frequently [3]. The closer the resulting NUWT_u values of all users are to each other (i.e. the lower the variance), the higher is the fairness.

3) Expected End Time (EET): Tóth and Klusáček state that existing fairness criteria do not take into account the resource demands of jobs (even though resource equality and dominant resource fairness were already defined). They observe that user expectations toward job wait times mainly correlate with the complexity of the job requirements and the number of jobs present in the batch. In order to follow these expectations, they propose expected end time [11]. We will not provide the complete algorithm for calculation of EET. To summarize, it is based on the number of jobs, number of resources required per job and the expected capacity of the system. It is calculated once for each user, and the objective is to minimize the percentage in which EET is exceeded.

4) Finish-Time Fairness (FTF): Mahajan et al. advocate for the *sharing incentive*, which states that if n users[2] are sharing a cluster, a user's jobs should not run slower on the shared cluster compared to a dedicated cluster with $1/n$ of the resources [5]. Let T_{sh} denote the actual, shared finish-time. Let T_{id} denote the finish-time of the user's jobs in its own independent and exclusive $1/n$ share of the cluster. Then, finish-time fairness is given by

$$\rho = \frac{T_{\text{sh}}}{T_{\text{id}}} \tag{8}$$

To evaluate the fairness of a schedule, the distribution of ρ across users is analyzed. A tighter distribution (i.e. low variance), and low maximum value of ρ indicate higher fairness [5].

C. Machine Fairness

To the best of our knowledge, the term *machine fairness* has not yet been used in the context of job scheduling. However, we consider it a relevant third category, complementing job fairness and user fairness. To some extent, machine fairness is similar to traditional system performance metrics. For example, fairness of machine completion times is an optimization of makespan. Similarly, fairness of machine inactivity is an optimization of utilization. In fact, this approach has been utilized to connect the field of job scheduling to the field of *fair item allocation*. In 2024, Huang and Segal-Halevi made progress within analysis of a fair chore allocation algorithm by reducing it to a well-known job scheduling algorithm [2]. They proved the relevance of the connection between the fields, which future work could expand upon.

Fairness Metric	Agent	Value	Objective
Makespan	Machine	Finish Time	Max-min
Utilization	Machine	Inactivity	Max-min
Job Priority Fairness	Job	Wait Time	By priority
Fair Slowdown	Job	Slowdown	Min % violations
Fair Start Time	Job	Wait Time	Min avg. violation
Resource Equality	Job	Resource Time	Min avg. violation
Net Benefit	Job	Wait Time	Max benefit
Dominant Resou…	User	Resource Share	Max-min
Normalized User…	User	Wait Time	Min variance
Expected End Time	User	Finish Time	Min % violations
Finish-Time Fairness	User	Finish Time	Min variance

TABLE I: Overview of fairness metrics.

III. Comparison and Evaluation

Table I provides an overview of the fairness metrics included in this survey. Since makespan and utilization are known performance metrics, they will not be discussed further. We will also not argue that any fairness criteria is strictly better than another, since this a question about the preferences of the users of the system.

A noteworthy observation is that the first five (up to 2009) consider fairness between jobs, while the most recent four (from 2011 to 2020) consider fairness between users. This suggests that the introduction of user-level metrics have been successful, and that they continue to be developed.

Most metrics are calculated from wait time or finish time, while some address the distribution of resources. RE calculates a deserved resource time for each job. DRF provides a fair share metric for heterogeneous resources. Similarly, the availability of resources is accounted for in the calculation of fair finish times for both EET and FTF.

In the case of FSD, FST, RE and EET, a threshold value is provided. The total fairness is then based on violations of this value, for example if a job starts after its fair start time. FSD and EET aggregate violation counts, while FST and RE aggregate by average violation size. The first approach is particularly a risk to fairness, since one large violation is considered better than two small violations. For optimal fairness, we would suggest a max-min approach, in which the maximal violation is minimized. However, this can be computationally intensive and may not be reasonable in practice.

A common concern when it comes to fairness metrics is the potential downgrade in performance. That aspect has not been devoted much attention to in this work. However, we observe that many of the works that propose fairness metrics also present complementary scheduling procedures. A shared characteristic of these procedures is that they perform well in terms of both fairness and traditional performance metrics. We believe there are two reasons for this: 1) Any proposal of underwhelming performance would be discarded due to the prioritizations of the field. 2) All metrics we have described are performance optimizations with regards to each job or each user, and it is not too surprising that this is not at the expense of global system performance.

[2]Mahajan et al. actually discuss *apps* as collections of multiple jobs, but this corresponds to our definition of a user.

IV. Conclusion

Fairness in parallel job scheduling can be categorized into *job fairness*, *user fairness*, and *machine fairness*, depending on what identities fairness is considered between. For job fairness, we have reviewed *job priority fairness*, *fair slowdown*, *fair start time*, *resource equality* and *net benefit*. For user fairness, we have reviewed *dominant resource fairness*, *normalized user wait time*, *expected end time* and *finish-time fairness*. Machine fairness is related to traditional performance metrics and may be of theoretical interest due to its connection to fair item allocation.

We have observed a shift from traditional job-related fairness, to fairness metrics on a user-level. It has also become common to include resource availability in the calculation of fairness values. Some even consider heterogeneous resource demands. The choice of a fairness metric depends on the system in question, and the preferences of the system's users. Future work may dive deeper into the relation between fairness and performance.

References

[1] Ali Ghodsi et al. "Dominant resource fairness: fair allocation of multiple resource types". In: *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*. NSDI'11. Boston, MA: USENIX Association, 2011, pp. 323–336.

[2] Xin Huang and Erel Segal-Halevi. *A Reduction from Chores Allocation to Job Scheduling*. 2024. arXiv: 2302.04581 [cs.GT]. URL: https://arxiv.org/abs/2302.04581.

[3] Dalibor Klusácek and Hana Rudová. "Performance and Fairness for Users in Parallel Job Scheduling". In: *Job Scheduling Strategies for Parallel Processing*. Ed. by Walfredo Cirne et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 235–252. ISBN: 978-3-642-35867-8.

[4] Arkadiusz Madej et al. "Priority-based Fair Scheduling in Edge Computing". In: *2020 IEEE 4th International Conference on Fog and Edge Computing (ICFEC)*. 2020, pp. 39–48. DOI: 10.1109/ICFEC50348.2020.00012.

[5] Kshiteej Mahajan et al. "Themis: Fair and Efficient GPU Cluster Scheduling". In: *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. Santa Clara, CA: USENIX Association, Feb. 2020, pp. 289–304. ISBN: 978-1-939133-13-7. URL: https://www.usenix.org/conference/nsdi20/presentation/mahajan.

[6] Bo Munch-Andersen and Torben U. Zahle. "Scheduling according to job priority with prevention of deadlock and permanent blocking". In: *Acta Informatica* 8.2 (June 1, 1977), pp. 153–175. ISSN: 1432-0525. DOI: 10.1007/BF00289247. URL: https://doi.org/10.1007/BF00289247.

[7] John Ngubiri and Mario van Vliet. “A metric of fairness for parallel job schedulers”. In: *Concurrency and Computation: Practice and Experience* 21.12 (2009), pp. 1525–1546. DOI: https://doi.org/10.1002/cpe.1384. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.1384. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.1384.

[8] G. Sabin, G. Kochhar, and P. Sadayappan. “Job fairness in non-preemptive job scheduling”. In: *International Conference on Parallel Processing, 2004. ICPP 2004.* 2004, 186–194 vol.1. DOI: 10.1109/ICPP.2004.1327920.

[9] Gerald Sabin and P. Sadayappan. “Unfairness Metrics for Space-Sharing Parallel Job Schedulers”. In: *Job Scheduling Strategies for Parallel Processing*. Ed. by Dror Feitelson et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 238–256. ISBN: 978-3-540-31617-6.

[10] Srividya Srinivasan et al. “Selective Reservation Strategies for Backfill Job Scheduling”. In: *Job Scheduling Strategies for Parallel Processing*. Ed. by Dror G. Feitelson, Larry Rudolph, and Uwe Schwiegelshohn. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 55–71. ISBN: 978-3-540-36180-0.

[11] Šimon Tóth and Dalibor Klusáček. “User-Aware Metrics for Measuring Quality of Parallel Job Schedules”. In: *Job Scheduling Strategies for Parallel Processing*. Ed. by Walfredo Cirne and Narayan Desai. Cham: Springer International Publishing, 2015, pp. 90–107. ISBN: 978-3-319-15789-4.