# OStrich: Fair Scheduling for Multiple Submissions

Joseph Emeras, Vinicius Pinheiro, Krzysztof Rzadca, Denis Trystram

# OStrich: Fair Scheduling for Multiple Submissions

Joseph Emeras[1], Vinicius Pinheiro[2], Krzysztof Rzadca[3], and Denis Trystram[4]

[1] Laboratoire d'Informatique de Grenoble CEA - CNRS, France,
joseph.emeras@imag.fr
[2] Lab. for Parallel and Distributed Computing University of São Paulo, Brasil,
vinicius.pinheiro@ime.usp.br
[3] Institute of Informatics University of Warsaw, Poland,
krzadca@mimuw.edu.pl
[4] Grenoble Institute of Technology and Institut Universitaire de France
trystram@imag.fr

**Abstract.** Campaign Scheduling is characterized by multiple job submissions issued from multiple users over time. This model perfectly suits today's systems since most available parallel environments have multiple users sharing a common infrastructure. When scheduling individually the jobs submitted by various users, one crucial issue is to ensure *fairness*. This work presents a new fair scheduling algorithm called *OStrich* whose principle is to maintain a virtual time-sharing schedule in which the same amount of processors is assigned to each user. The completion times in the virtual schedule determine the execution order on the physical processors. Then, the campaigns are interleaved in a fair way by *OStrich*. For independent sequential jobs, we show that *OStrich* guarantees the *stretch* of a campaign to be proportional to campaign's size and the total number of users. The *stretch* is used for measuring by what factor a workload is slowed down relative to the time it takes on an unloaded system. The theoretical performance of our solution is assessed by simulating *OStrich* compared to the classical FCFS algorithm, issued from synthetic workload traces generated by two different user profiles. This is done to demonstrate how *OStrich* benefits both types of users, in contrast to FCFS.

**Keywords:** Job scheduling; Fairness; Job campaigns; Multi-User; Workload Traces

## 1 Introduction

High performance computing (HPC) systems are usually shared by multiple users who compete for the usage of the processors in order to execute their jobs. Most of such systems embrace users and projects around a common infrastructure that simplifies resource management and application execution through a centralized scheduler. In the past, most users were oriented toward the maximization of the throughput but the popularization of those systems attracted other types of

users. Nowadays, the users turned to the optimization of the response-time [1]. Workload of response-time users is composed of multiple submissions released sequentially over time [2–5]. For such users, the criterion of throughput is not meaningful as they are more interested in the flow time of each submission.

In this work, the problem of multiple submissions on parallel system is narrowed to the notion of *Campaign Scheduling*, introduced in [6] and analyzed under restrictive assumptions. The campaign scheduling problem models a user submission pattern commonly found in parallel systems used for scientific research: a user submits a set of jobs, analyzes the outcomes and then resubmits another set of jobs. In other words, the campaigns are sets of jobs issued from a user and they must be scheduled one after the other since the submission of a new campaign depends on the outcome of the previous one As this pattern is an interactive process, the objective from the user point of view is to minimize the time each campaign spent in the system, namely the campaign's flow time. As soon as a campaign finishes, soon the user is ready to submit the next one.

Common approaches such as FCFS (First-Come-First-Served), backfilling mechanisms and classical list scheduling strategies are not well-adapted for multi-user environments. FCFS, for instance, can be unfair to users who always submit small jobs: it can take an arbitrarily long time to execute since the processors may be fully occupied with jobs from another users. In turn, policies whose priority is based on job length like SPT (Shortest Processing Time first) [7] and LPT (Longest Processing Time first) [8] are subject to job starvation if applied without a dynamic priority mechanism. Backfilling can be used to fill the idle gaps between jobs and increase system utilization but they deliver no individual guarantees to users regarding performance neither equitable treatment [9].

We propose in this paper a new on-line scheduling algorithm (called *OStrich*) that explicitly maintains fairness between the users by bounding the worst-case stretch of each campaign. We show that user's performance does not depend on the total load of the system, but only on the number of users competing for the processors and on its own workload. We are also able to quantify and optimize the *performance* of each user.

The rest of this paper is organized as follows. In the next section, we give an overview of the state-of-the-art of scheduling with multiple users. In Section 3 we present a formal model of Campaign Scheduling. This model is an extension of what was defined in [6]. Section 4 is dedicated to the description of our solution, a new algorithm for the problem of campaign scheduling with multiple users. The theoretical results are depicted and analyzed in Section 5. Our solution is assessed through simulations that uses user profiles based on short and long job lengths. This is presented in Section 6. Finally, we present our conclusions and future work in Section 7.

## 2 State-of-the-art

The main works related to this paper address the problem of optimizing various users criteria and the use of fair policies on parallel systems. The Multi-Users

Scheduling Problem was introduced for a single processor and two users by Agnetis et al. [10]. This study focused on optimization problems. The authors show that when both users are interested each in their makespan, the problem can be solved in polynomial time. Saule and Trystram [11] extended the analysis for scheduling independent sequential jobs belonging to $k$ different users on $m$ identical processors. This is an offline problem where all the jobs are known in advance and can be immediately executed. They proved that the problem becomes strongly NP-hard as soon as one user aims at optimizing the makespan.

Fairness is an important issue while designing scheduling policies and it has gained growing attention in the last decade. However, it is still a fuzzy concept that has been handled in many different ways, varying according to the target problems. In [12] and [13], several metrics are proposed for expressing the degree of unfairness in various systems. Both works evaluate the unfairness of algorithms such as FCFS, backfilling and processor sharing, but fairness is associated with the jobs and their service requirements. Thus, the concept of fairness is seen as "fairness between jobs" instead of "fairness between users" as we propose.

Another common approach in scheduling is to use the distributive fairness. It consists in allocating the available processors among the competing users according to some fixed policy. Some examples are the fair share policies of Maui Scheduler [14], the fair-share factor of SLURM Multifactor Priority Plugin [15] and the Hadoop Fair Scheduler [16]). Users' satisfaction (i.e. utilities), however, is a function of not only the assigned processors, but also the needs. If the needs are unequal, even if the processors are allocated according to the assigned shares, the resulting utilities will differ. Moreover, strict sharing of processors according to fair share policy may be inefficient: forcing the system to execute jobs from all the users concurrently may slow down the execution of all the users while executing one user after the other would be better for some users without worsening the execution times of the others.

In contrast to distributive fairness, our goal is to ensure *fair distribution of utilities*, i.e., the performance users get from the system. It is crucial to use a suitable measure of utility. One of the accepted and used metrics is the stretch, i.e. the flow time normalized by the job's processing time. The stretch and flow metrics were first studied by Bender et al. [17] for continuous job streams. Stretch optimization was also studied for independent tasks without preemption [18], Bag-of-Tasks applications [19,20], multiple parallel task graphs [21] and for sharing broadcast bandwidth between clients requests [22].

The concept of campaign is related to the bag-of-task model [23, 24] and the groups of jobs model [25]. However, campaigns differ from bag-of-tasks as the jobs belonging to a single campaign have different lengths, sizes and dependencies. Unlike the group of jobs model, we assume that a user does not submit the subsequent campaign until the previous one was completed. Our scheduling algorithm uses the concept of a virtual completion time. Similar ideas are used for fair queuing in communication networks (see [26] and the references within).

## 3 Model and Problem Definition

The model consists of $k$ users (indexed by $u$) sharing the processors on a parallel platform composed of $m$ identical processors. The processors are managed by a centralized scheduler. Figure 1 illustrates some of the used notations. A user workload is composed of successive campaigns where each campaign, indexed by $i$ is submitted at a time denoted by $t_i^u$ and is composed of a set of independent and non-preemptive sequential jobs. We consider an on-line problem in which any particular campaign $i$ (and its jobs) is unknown to the scheduler until it is submitted. A campaign is defined as the set $J_i^u$ containing the jobs released by a user $u$ in one submission; $n_i^u$ denotes the number of jobs of a campaign and $n^u$ the total number of jobs released in all the campaigns of user $u$.

The jobs inside a campaign are indexed by $j$. The job's length is denoted by $p_{i,j}^u$ and, so, the total workload within campaign $i$ is: $W_i^u = \sum_j p_{i,j}^u$. A job, once started, cannot be interrupted nor preempted. The job start time is denoted by $s_{i,j}^u$ and its completion time is denoted by $C_{i,j}^u$.

The start time of a campaign is denoted by $s_i^u$. It is defined as the time the first job starts, $s_i^u \triangleq \min_j s_{i,j}^u$. The campaign's completion time $C_i^u$ is the time the last job completes, $C_i^u \triangleq \max_j C_{i,j}^u$.

The campaign's flow time, denoted as $\Delta_i^u$, is equal to the amount of time the jobs of a campaign stay in the system: $\Delta_i^u \triangleq C_i^u - t_i^u$.

The campaign's stretch is denoted by $D_i^u$ and is defined as a generalization of a job's stretch. Formally, a job stretch $D_{i,j}^u$ is equal to the relative degradation of its flow time, $D_{i,j}^u = (C_{i,j}^u - t_{i,j}^u)/p_{i,j}^u$, where $p_{i,j}^u$ is the job length (and, thus, the job's optimum flow time) [17]. Determining a single campaign's optimum flow time means solving the general scheduling problem, so it is NP-hard. Thus, instead, we use a lower bound on campaign's flow time defined by $l_i^u = \max(W_i^u/m, p_{\max}^u)$, where $p_{\max}^u = \max_j p_{i,j}^u$. Consequently, the campaign's stretch is defined as $D_i^u = \Delta_i^u / l_i^u$.

A user $u$ cannot submit her-his next campaign $i+1$ until her-his previous campaign $i$ completes, thus $t_{i+1}^u \geq C_i^u$. The time between the completion of campaign $i$ and the submission of the next one $(i+1)$, called the *think time*, is denoted as $tt_{i+1}^u = t_{i+1}^u - C_i^u$.

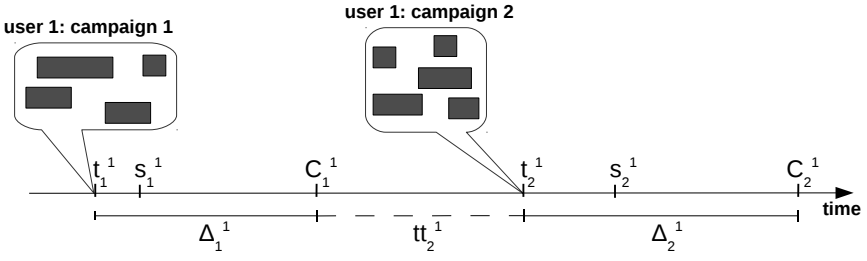



Fig. 1: Campaign Scheduling notations (two submissions from one user)

The objective is to minimize the per-user and per-campaign stretch $D_i^u$. We consider stretch (in contrast to the flow time), as it weights the responsiveness of the system by the assigned load; it is natural to expect that small workloads will be computed faster than larger ones. We consider it on a per-user basis, as this results in fairness of the system towards individual users. Moreover, considering stretch of each campaign (rather than the overall stretch) guarantees that not only the final result, but also the intermediate ones are timely computed.

The problem of minimizing per-user and per-campaign stretch $D_i^u$ is NP-hard, as when restricted to a single user ($k = 1$) and to a single campaign, it is equivalent to the classical problem of minimization of the makespan on identical parallel processors ($P||C_{\max}$) [6, 11].

## 4 Algorithm

We propose in this section a new scheduling algorithm called *OStrich*. The algorithm guarantees the worst-case stretch of each campaign of each user $D_i^u$ to be proportional to the campaign's workload and the number of active users in the system. OStrich's principle is to create a virtual fair-sharing schedule that determines the execution priorities of the campaigns in the real schedule. The algorithm maintains a list of ready-to-execute campaigns ordered by their priorities and interactively selects the next job from the highest priority campaign. Any scheduling policy can be used to determine the execution order of jobs within a single campaign; for instance LPT [8] (or, more appropriately, MLPT [27]) or Shortest Processing Time (SPT) [7].

The virtual fair-sharing schedule is maintained by dividing the processors between the active users at each given moment. The processors are divided *evenly* among the users, independently of users' submitted workload. The priority of a user's campaign is determined by its virtual completion time, i.e. the completion time in the virtual schedule. The campaign with the shortest virtual completion time has the priority of execution. This virtual completion time is denoted by $\tilde{C}_i^u$ for a campaign $J_i^u$ (more generally, we will use $\tilde{x}$ for denoting a variable $x$ in the virtual schedule). That way, if a user $u$ submits a campaign at time $t_i^u$, its virtual completion time is defined as the total workload of the campaign divided by its share of processors, added by its virtual start time. More formally:

$$\tilde{C}_i^u(t) = \tilde{W}_i^u/(m/\tilde{k}(t)) + \tilde{s}_i^u = \tilde{k}(t)\tilde{W}_i^u/m + \tilde{s}_i^u. \tag{4.1}$$

Note that the share of a user is defined as the number of processors $m$ divided by the number of active users at moment $t$, denoted by $\tilde{k}(t)$. By active users, we mean the users with unfinished campaigns at time $t$, according to the virtual schedule. Formally, $\tilde{k}(t)$ is defined as $\tilde{k}(t) = \sum_u^k \mathbb{1}\{u,t\}$ where $\mathbb{1}\{u,t\}$ is an indicating function that returns 1 if $\exists i \mid \tilde{C}_i^u > t_e$ and 0 otherwise.

A campaign starts in the virtual schedule after it is submitted, but also not sooner than the virtual completion time of the previous campaign (the previous campaign can be completed earlier in the real scheduler than in the virtual

schedule). So, according to this:

$$\tilde{s}_i^u = max(t_i^u, \tilde{C}_{i-1}^u). \tag{4.2}$$

This condition guarantees that at each time moment, at most one campaign of each user is executing in the virtual schedule, as it happens on the real scheduler. Thus, the number of allocated processors depends on the number of active users, and not the system load. Additionally, *OStrich* does not allow a campaign to start before its virtual start time ($s_i^u \geq \tilde{s}_i^u$). This mechanism keeps the real schedule in accordance with the fair principles of the virtual schedule: a user is not able to take a greater share of the processors than what is assigned in the virtual schedule.

The virtual completion time of the campaigns can be updated on two events: the submission of a new campaign and the completion of a campaign in the virtual schedule. These events may change the number of active users $\tilde{k}(t)$ and, thus, modify the virtual completion times of other active campaigns. Besides, at each event $e$ occurring at time $t_e$, the virtual workload of a campaign ($\tilde{W}_i^u$) must be redefined based on how much it is left to be executed in the virtual schedule. The remaining workload of a campaign is defined by taking the time passed since the last event occurrence $t_{e-1}$ and multiplying it by campaign's share of processors on that time interval. Considering all the events passed after the campaign's submission, the workload formula is $\tilde{W}_i^u = \sum_j p_{i,j}^u - \sum_e (m.(t_e - t_{e-1})/\tilde{k}(t_{e-1}))$.

An example of schedule with *OStrich* is available at the appendix A.

## 5 Theoretical Analysis

In this section, the worst case stretch of a campaign is to analyzed. The idea for the proof is to bound the completion time of the last job of a campaign using a "global area" argument compared to the virtual schedule. In this proof, $p_{\max}$ denotes the maximum job length in the system. "Area" is a measure in terms of amount of time × number of processors. The virtual schedule is denoted by V and the real schedule by R. To simplify the formulation of proofs, we will say that the virtual schedule V "executes" jobs, even though V is just an abstraction used for prioritizing real jobs. At time $t$, a job is "executed" by V if in V there is a fraction of processors assigned to this job.

### 5.1 Worst-Case Bound

As V can assign a job an arbitrary fraction of processors (from $\epsilon$ to $m$), a schedule in V is represented by horizontal load streams, one for each active user. Idle intervals can be present in V only when there are no ready jobs to be executed. In turn, R must execute each job on a single processor, thus some processors can be idle even when there are ready jobs. This can happen when $t_i^u < \tilde{s}_i^u$ and the ready jobs of campaign $i$ must wait until moment $\tilde{s}_i^u$ arrives.

So, the question is whether the idle times that might additionally appear in R can cause a systematic delay of R compared to V. The following lemma shows that once R is delayed by an area of $mp_{\max}$, the delay does not grow further, as there is always a ready job to be executed.

The lemma considers only the idle time in the "middle" of the schedule, i.e., after the start time of the first job and up to the start time of the last job; this is sufficient to characterize the on-line behavior of *OStrich*.

**Lemma 1.** *The total idle area in R (counted from the earliest to the latest job start time) exceeds the total idle area in V by at most* $mp_{\max}$.

*Proof.* Consider first a V schedule with no idle times. Assume by contradiction that $t$ is the first time moment when the total idle area in R starts to exceed $mp_{\max}$. Thus, at least one processor is free at time $t$ and there is no ready jobs to be executed. As V has no idle times, at time $t$ the area executed by V exceeds the area executed by R by more than $mp_{\max}$. Thus, the area exceeding $mp_{\max}$ is ready to be executed at R: as a single job has an area of at most $p_{\max}$, an area of $mp_{\max}$ is composed of at least $m$ jobs. Thus, at least $m$ jobs are being executed, or ready to be executed in R. This contradicts the assumption that there is at least one free processor at R at time $t$.

If there is idle time in V, each idle period can be modeled as a set of *dummy* jobs $\{J_I\}$ that are executed by V, but not necessarily (and/or not completely) by R. If R executes $\{J_I\}$ entirely, the thesis is true by the argument from the previous paragraph (as $\{J_I\}$ contributes with the same amount $\sum p_I$ of idle area to V and to R). If R executes $\{J_I\}$ partially (as $\{J'_I\}$, with $0 \leq p'_I \leq p_I$) the contribution of these jobs to the idle area of R ($\sum p'_I$) is smaller than to V ($\sum p_I$). ■

R starts to execute jobs from campaign $J_i^u$ when this campaign has the shortest completion time in V. Yet, it is possible that after some, but not all, jobs from $J_i^u$ have started, another user $v$ submits her/his campaign $J_j^{(v)}$ having a lower area than what remains of $J_i^u$, and thus gaining higher priority. Thus, $J_i^u$ is executed in R in so-called *pieces*: two jobs $J_k, J_l \in J_i^u$ belong to the same piece iff no job from other campaign $J_j^{(v)}$ starts between them ($\nexists J' : J \in J_j^{(v)} \wedge s_{Jk} < s_{J'} < s_{Jl}$).

The following lemma bounds the completion time of the last piece of the campaign. After a campaign is completed in the virtual schedule, it cannot be delayed by any other newly-submitted campaign; thus it has the highest priority and its remaining jobs are executed in one piece (i.e., the last piece). The lemma upper-bounds the virtual area having higher priority by the area of the campaign, as in the worst case $k$ users submit campaigns of equal area, thus ending at the same time in V, and thus being executed in arbitrary order in R.

**Lemma 2.** *The completion time* $C_{i,q}^u$ *of the last piece* $q$ *of a campaign* $J_i^u$ *is bounded by a sum:*

$$C_{i,q}^u \leq t_i^u + k\frac{W_{i-1}^u}{m} + p_{\max} + (k-1)\frac{W_i^u}{m} + p_{\max} + \frac{W_i^u}{m} + p_{\max}^u. \tag{5.1}$$

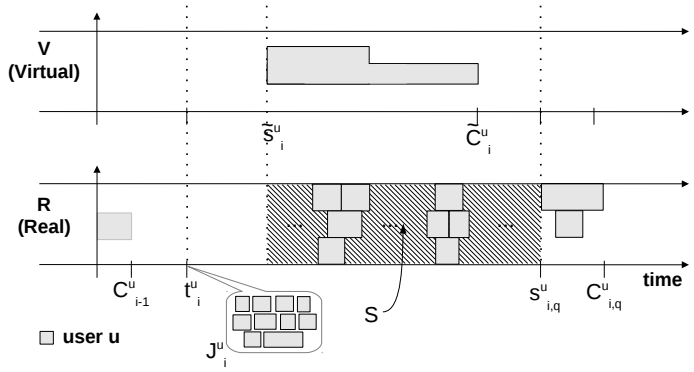



Fig. 2: Analysis of *OStrich*: bound for the campaign stretch

*Proof.* In (5.1), $t_i^u + kW_{i-1}^u/m$ expresses the maximum time a campaign may wait until the virtual completion time of the previous campaign $J_{i-1}^u$ of the same user; $(k-1)W_i^u/m$ bounds the time needed to execute other users' campaigns that can have higher priority; $W_i^u/m + p_{\max}^u$ bounds the execution time of the campaign $J_i^u$; and two $p_{\max}$ elements represent the maximum lateness of R compared to V and the maximum time needed to claim all the processors.

From (4.1), (4.2) and knowing that $t_i^u \geq \tilde{s}_{i-1}^u$ (the next campaign cannot be released before the previous one starts), $\tilde{s}_i^u \leq t_i^u + kW_{i-1}^u/m$.

There is no idle time in R in the period $[\tilde{s}_i^u, s_{i,q}^u)$, otherwise, the last piece could have been started earlier.

We denote by $A$ the area of jobs executed in R after the time moment $\tilde{s}_i^u$ and until $\tilde{s}_{i,q}^u$. We claim that $A \leq mp_{\max} + (k-1)W_i^u + W_i'^u$, where $W_i'^u$ is the area of jobs from campaign $J_i^u$ executed until $s_{i,q}^u$. The Figure 2 facilitates the visualization of these notations, including the area $A$ (shaded area).

To prove the claim, we analyze job $J$ executed in R in the period $[\tilde{s}_i^u, s_{i,q}^u)$. First, $J$ is not executed in V after $s_{i,q}^u$. If $J$ is in V after $s_{i,q}^u$, $J$ has lower priority than jobs from campaign $J_i^u$, so *OStrich* would not choose $J$ over jobs from campaign $J_i^u$.

Second, if $J$ is executed in V before $\tilde{s}_i^u$, it means that R is "late" in terms of executed area: but the total area of such "late" jobs it at most $mp_{\max}$ (from Lemma 1).

Thus, if $J$ has a corresponding area in the virtual schedule executed in the period $[\tilde{s}_i^u, s_{i,q}^u)$, the area $A$ of the jobs started in the real schedule in this period is equal to the area of the virtual schedule between $[\tilde{s}_i^u, s_{i,q}^u)$ plus the lateness $mp_{\max}$. Recall that from time $s_{i,q}^u$ till the start of the last job of $J_i^u$, the campaign $J_i^u$ has the highest priority (as it is not interrupted by any other campaign). Thus, at the latest, $s_{i,q}^u$ corresponds to the time moment $\tilde{C}_i^u$ in the virtual schedule when the campaign $J_i^u$ completes (plus the lateness $p_{\max}$). Thus, by definition of the virtual schedule, $s_{i,q}^u \leq p_{\max} + \tilde{s}_i^u + k\frac{W_i^u}{m}$.

Starting from $s^u_{i,q}$, the remaining jobs of $J^u_i$ start and complete. $J^u_i$ can claim all processors at the latest $p_{\max}$ after $s^u_{i,q}$. Then, by using classic lower bounds, it takes $W^u_i/m + p^u_{\max}$ to complete the campaign. ■

The following theorem states that in *OStrich* the stretch of a campaign depends only on the number of active users and the relative area of two consecutive campaigns.

**Theorem 1.** *The stretch of a campaign is proportional to the number of active users $k$[5] and the relative area of consecutive campaigns. $D^u_i \in O(k(1 + \frac{W^u_{i-1}}{W^u_i}))$.*

*Proof.* The result follows directly from Lemma 2. Recall that, for campaign $J^u_i$, the stretch $D^u_i$ is defined by $D^u_i = (C^u_i - t^u_i)/l^u_i = (C^u_i - t^u_i)/\max(W^u_i/m, p^u_{\max})$. Also, $C^u_i = C^u_{i,q}$, i.e. the completion time of a campaign is equal to the completion time of its last "piece". Replacing $C^u_i$ by the definition of $C^u_{i,q}$ taken from Lemma 2, we have

$$D^u_i \leq \frac{\frac{kW^u_{i-1}}{m} + 3p_{\max} + \frac{kW^u_i}{m}}{\max(W^u_i/m, p^u_{\max})} \leq \frac{\frac{kW^u_{i-1}}{m} + 3p_{\max} + \frac{kW^u_i}{m}}{W^u_i/m} \leq k(1 + \frac{W^u_{i-1}}{W^u_i}) + 3mp_{\max}.$$

For a given supercomputer, $m$ is constant; similarly, the maximum size of a job $p_{\max}$ can be treated as constant, as it is typically limited by system administrators. Hence, $D^u_i \in O(k(1 + W^u_{i-1}/W^u_i))$. ■

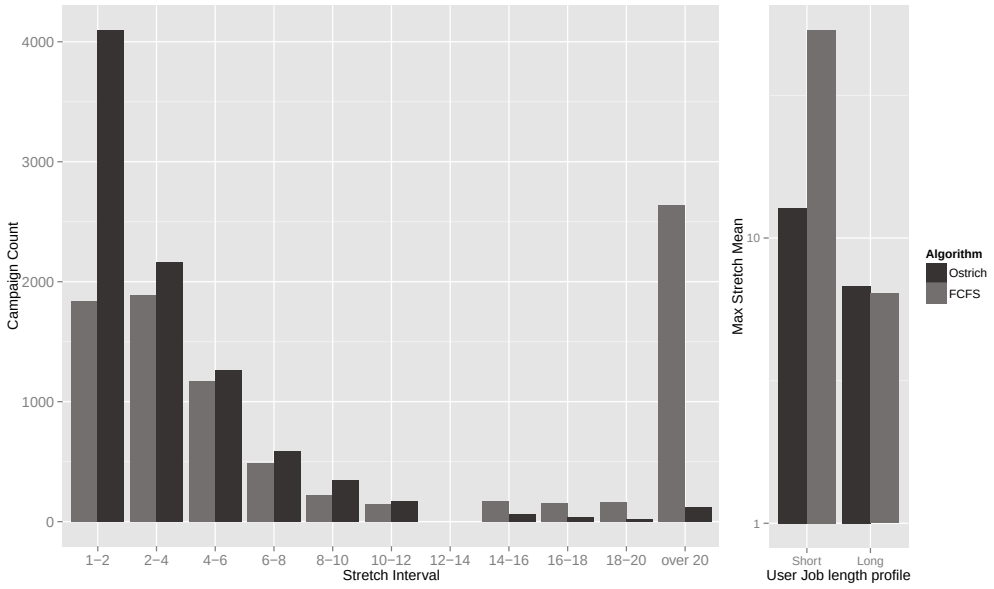
It is worth noting that the stretch does not dependent on the current total load of the system. Heavily-loaded users do not influence the stretch of less-loaded ones. Also, this bound is tight (the proof is available at [28]).

## 6 Simulations

In this section, we analyze the performance of *OStrich*. We present a simulation to demonstrates that *OStrich* results in lower stretch values against. The results produced by *OStrich* are compared with a FCFS classical algorithm that schedules campaigns in a First-In-First-Out order. The simulator plays the role of a centralized scheduler: it takes an instance of a user workload as inputs; and it calculates the campaign stretches and the max-stretch per user obtained by each algorithm in an environment composed of $m = 64$ identical processors.

We run 40 instances where instance is composed of $10^4$ jobs. For each job we set its length $p$ according to the user profile. We defined 2 user profiles: *short* and *long*. Short users submit short jobs with lengths uniformly taken from the range $[1; 3.6 \times 10^3]$ (seconds as time unit). Long users submit long jobs with lengths uniformly taken from the range $[3.6 \times 10^3; 3.6 \times 10^4]$. Each job starts a new campaign with probability of 0.02; otherwise, it belongs to the previous campaign. If the job starts a new campaign, we set the owner of this campaign according to a uniform distribution.

[5] The number of active users may vary on time. Here, we assume that $k$ is the biggest value it assume during the execution of the campaign.




(a) Stretch values by intervals (b) Max campaign stretch per user

Fig. 3: Ostrich vs FCFS: comparing stretch values of campaigns

In general, the results confirm our expectations and show that *OStrich* results in significantly lower max stretches than FCFS. The Figure 3a shows the distribution of stretch values for all campaigns. The number of campaigns with stretch lower than 2 for *OStrich* is more than twice the number obtained with FCFS. More important, though is the number of high stretch values above: while on *OStrich* this number decreases rapidly towards 0 as stretch increases, with FCFS it is bigger than 2600 above 20, representing 42.3% of the total. The occurrence of stretch values above 20 is only 117 for *OStrich* (1.3%).

The Figure 3b shows the max stretch average per user profile (in a log scale) and here we can see how *OStrich* accomplishes its purpose: short users are penalized by FCFS with big stretch values (whose average is above 50) while *OStrich* does not heavily discriminate users by their profiles, guaranteeing a more fair treatment for all the users (average of 12.8 for short users). For long users, FCFS and *OStrich* have almost the same performance (average of 6.3 for FCFS and 6.8 for *OStrich*).

## 7 Concluding Remarks

We have presented in this work a new scheduling algorithm for the problem of scheduling multiple submissions issued by many users. *OStrich* algorithm has been designed to handle the problem of fairness between users by defining exe-

cution priorities according to a criterion based on *stretch*. The principle of the proposed solution is to dynamically determine the priorities between the campaigns based on a fair share virtual schedule. We proved that *OStrich* delivers performance guarantee for the max stretch value of a user campaign that depends only on the user workload and on the number of active users.

The performance of our algorithm is assessed by running simulations on workloads composed of two types of users sharing a parallel system. The results show that, for short job user profiles, *OStrich* achieves lower stretches than the FCFS while it remains as good as FCFS for long job users; moreover distribution of stretches among users is more equal. Therefore, *OStrich* delivers a good compromise between fairness and user performance without worsening overall performance.

Next, we plan to study how *OStrich* can be used to achieve fairness with parallel jobs, which introduces another relevant issues like overall system utilization and backfilling impact on stretch minimization.

## References


1. Donassolo, B., Legrand, A., Geyer, C.: Non-cooperative scheduling considered harmful in collaborative volunteer computing environments. In: Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. CCGRID '11 (2011) 144–153
2. Feitelson, D.: Workload Modeling for Computer Systems Performance Evaluation. (2005)
3. Zakay, N., Feitelson, D.G.: On identifying user session boundaries in parallel workload logs. In: Proc. of the 16th Workshop on Job Scheduling Strategies for Parallel Processing, The Hebrew University, Israel (may 2012)
4. Mehrzadi, D., Feitelson, D.G.: On extracting session data from activity logs. In: Proceedings of the 5th Annual International Systems and Storage Conference. SYSTOR '12 (2012) 3:1–3:7
5. Shmueli, E., Feitelson, D.: Using site-level modeling to evaluate the performance of parallel system schedulers. In: Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2006. MASCOTS 2006. 14th IEEE International Symposium on. (sept. 2006) 167 – 178
6. Pinheiro, V., Rzadca, K., Trystram, D.: Campaign scheduling. In: IEEE International Conference on High Performance Computing (HiPC), Proceedings. (2012) Accepted for publication.
7. Bruno, J., Coffman, J.E.G., Sethi, R.: Scheduling independent tasks to reduce mean finishing time. Commun. ACM **17**(7) (July 1974) 382–387
8. Graham, R.L.: Bounds on multiprocessing timing anomalies. SIAM JOURNAL ON APPLIED MATHEMATICS **17**(2) (1969) 416–429
9. Srinivasan, S., Kettimuthu, R., Subramani, V., Sadayappan, P.: Characterization of backfilling strategies for parallel job scheduling. In: Parallel Processing Workshops, 2002. Proceedings. International Conference on. (2002) 514–519
10. Agnetis, A., Mirchandani, P.B., Pacciarelli, D., Pacifici, A.: Scheduling problems with two competing agents. Operations Research **52**(2) (2004) 229–242
11. Saule, E., Trystram, D.: Multi-users scheduling in parallel systems. In: Proc. of IEEE International Parallel and Distributed Processing Symposium 2009, Washington, DC, USA (may 2009) 1–9

12. Raz, D., Levy, H., Avi-Itzhak, B.: A resource-allocation queueing fairness measure. SIGMETRICS Perform. Eval. Rev. **32**(1) (June 2004) 130–141
13. Sabin, G., Kochhar, G., Sadayappan, P.: Job fairness in non-preemptive job scheduling. In: Proceedings of the 2004 International Conference on Parallel Processing. ICPP '04 (2004) 186–194
14. Jackson, D.B., Snell, Q., Clement, M.J.: Core algorithms of the maui scheduler. In: Revised Papers from the 7th International Workshop on Job Scheduling Strategies for Parallel Processing. JSSPP '01 (2001) 87–102
15. Yoo, A., Jette, M., Grondona, M.: Slurm: Simple linux utility for resource management. In Feitelson, D., Rudolph, L., Schwiegelshohn, U., eds.: Job Scheduling Strategies for Parallel Processing. Volume 2862 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2003) 44–60
16. Zaharia, M.: The Hadoop Fair Scheduler. http://developer.yahoo.net/blogs/hadoop/FairSharePres.ppt
17. Bender, M.A., Chakrabarti, S., Muthukrishnan, S.: Flow and stretch metrics for scheduling continuous job streams. In: Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms. SODA '98, Philadelphia, PA, USA, Society for Industrial and Applied Mathematics (1998) 270–279
18. Bender, M.A., Muthukrishnan, S., Rajaraman, R.: Improved algorithms for stretch scheduling. In: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms. SODA '02, Philadelphia, PA, USA, Society for Industrial and Applied Mathematics (2002) 762–771
19. Legrand, A., Su, A., Vivien, F.: Minimizing the stretch when scheduling flows of biological requests. In: Proceedings of the eighteenth annual ACM symposium on Parallelism in algorithms and architectures. SPAA '06, New York, NY, USA, ACM (2006) 103–112
20. Celaya, J., Marchal, L.: A fair decentralized scheduler for bag-of-tasks applications on desktop grids. In: Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on. (may 2010) 538 –541
21. Casanova, H., Desprez, F., Suter, F.: Minimizing stretch and makespan of multiple parallel task graphs via malleable allocations. In: Parallel Processing (ICPP), 2010 39th International Conference on. (sept. 2010) 71 –80
22. Wu, Y., Cao, G.: Stretch-optimal scheduling for on-demand data broadcasts. In: Computer Communications and Networks, 2001. Proceedings. Tenth International Conference on. (2001) 500 –504
23. Iosup, A., Sonmez, O., Anoep, S., Epema, D.: The performance of bags-of-tasks in large-scale distributed systems. In: Proceedings of the 17th international symposium on High performance distributed computing, ACM (2008) 97–108
24. Beaumont, O., Carter, L., Ferrante, J., Legrand, A., Marchal, L., Robert, Y.: Centralized versus distributed schedulers for bag-of-tasks applications. Parallel and Distributed Systems, IEEE Transactions on **19**(5) (2008) 698–709
25. Iosup, A., Jan, M., Sonmez, O., Epema, D.: The characteristics and performance of groups of jobs in grids. In: Euro-Par 2007 Parallel Processing. Springer (2007) 382–393
26. Ghodsi, A., Sekar, V., Zaharia, M., Stoica, I.: Multi-resource fair queueing for packet processing. ACM SIGCOMM Computer Communication Review **42**(4) (2012) 1–12
27. Lee, C.Y.: Parallel machines scheduling with nonsimultaneous machine available time. Discrete Appl. Math. **30** (January 1991) 53–61
28. Emeras, J., Pinheiro, V., Rzadca, K., Trystram, D.: Fair Scheduling for Multiple Submissions. Research Report RR-LIG-033, LIG, Grenoble, France (2012)

# A Example

The figure 4 shows an example with the real and the virtual schedule generated by the *OStrich* algorithm in a system with 6 identical processors from $t = 0$ to $t = 7$. This example shows 4 submissions issued from 3 different users: two submissions at time $t = 0$, from users 1 and 2, and two submissions at times $t = 2$ and $t = 5$, from user 3. From $t = 0$ to $t = 2$, two users, 1 and 2, are the only ones in the system ($\tilde{k}(t) = 2$; $0 \leq t \leq 2$). The virtual schedule is constructed by sharing the processors equally between them and, according to (4.1), their virtual completion times are $\tilde{C}_1^1 = 16$ and $\tilde{C}_1^2 = 6$. The real schedule contains only jobs from user 2 since her/his virtual completion time is the smallest (thus, he has execution priority).

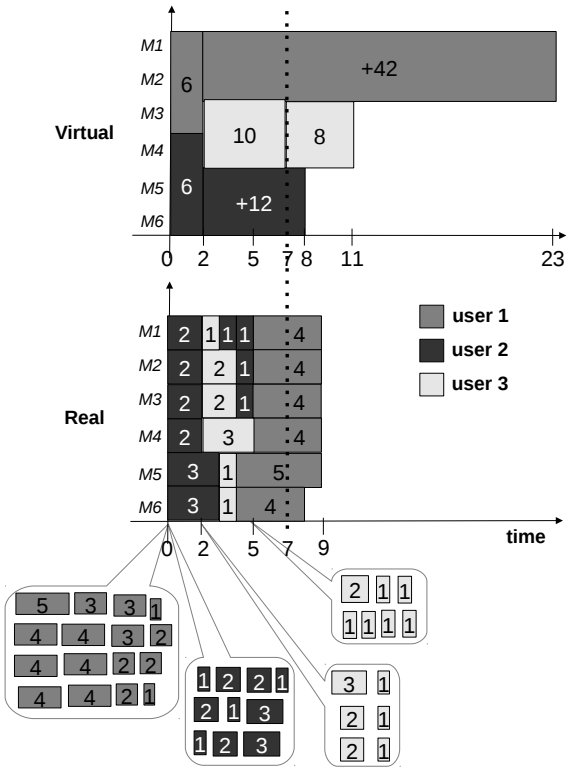



Fig. 4: Virtual and real schedule generated by the *OStrich* algorithm with 3 users

The situation changes at $t = 2$ when user 3 submits her/his first campaign. By that time, the users 1 and 2 had each a share of 6 in the virtual schedule but their virtual completion times were not exceeded. Now, the processors are equally shared between 3 users ($\tilde{k}2 = 3$). The virtual completion of user 3 is

set to $\tilde{C}_1^3 = 7$ and the virtual completion times of users 1 and 2 are updated to $\tilde{C}_1^1 = 23$ and $\tilde{C}_1^2 = 8$, according to their remaining workloads. User 3 is the user with the highest priority. In the real schedule, the first campaign of user 2 is interrupted—while executing jobs are not interrupted, once user 2's job completes, user 3's job starts. So, in order to use processors $M_5$ and $M_6$, user 3 must wait until $t = 3$.

From $t = 3$ to $t = 5$ the first campaign of user 3 is finished and also the first campaign of user 2, as its remaining jobs are executed. Additionally, some jobs of the first campaigns of user 1 finally start to execute. At $t = 5$ the second campaign of user 3 is submitted, but note that the virtual completion time of her/his first campaign is $\tilde{C}_1^3 = 7$. As $s_2^3 < \tilde{C}_1^3$, $\tilde{s}_2^3 = \tilde{C}_1^3$ and her/his virtual completion time is set to $\tilde{C}_2^3 = 11$. The user 3 retakes the priority but note that even if her/his first campaign is finished in the real schedule at $t = 5$, her/his next campaign must wait until $\tilde{s}_2^3 = 7$ to be taken into account.

The result of *OStrich* is a schedule with campaigns being interleaved and executed in many pieces, according to the changing priorities between users.