Campaign Scheduling

Vinicius Pinheiro
Lab. for Parallel and Distributed Computing
University of São Paulo, Brasil
Email: vinicius.pinheiro@ime.usp.br

Krzysztof Rzadca
Institute of Informatics
University of Warsaw, Poland
Email: krzadca@mimuw.edu.pl

Denis Trystram
Grenoble Institute of Technology
Institut Universitaire de France
Email: trystram@imag.fr

***Abstract*—We study the problem of scheduling in parallel systems with many users. We analyze scenarios with many submissions issued over time by several users. These submissions contain one or more jobs; the set of submissions are organized in successive *campaigns*. Jobs belonging to a single campaign are sequential and independent, but any job from a campaign cannot start until all the jobs from the previous campaign are completed. Each user's goal is to minimize the sum of flow times of his campaigns.**

We define a theoretical model for Campaign Scheduling and show that, in the general case, it is NP-hard. For the single-user case, we show that an ρ-approximation scheduling algorithm for the (classic) parallel job scheduling problem is also an ρ-approximation for the Campaign Scheduling problem. For the general case with k users, we establish a fairness criterion inspired by time sharing. We propose FAIRCAMP, a scheduling algorithm which uses campaign deadlines to achieve fairness among users between consecutive campaigns. We prove that FAIRCAMP increases the flow time of each user by a factor of at most $k\rho$ compared with a machine dedicated to the user. We also prove that FAIRCAMP is a ρ-approximation algorithm for the maximum stretch.

By simulation, we compare FAIRCAMP to the First-Come-First-Served (FCFS). We show that, compared with FCFS, FAIRCAMP reduces the maximum stretch by up to 3.4 times. The difference is significant in systems used by many ($k > 5$) users.

Our results show that, rather than just individual, independent jobs, campaigns of jobs can be handled by the scheduler efficiently and fairly.

I. Introduction

The hardware industry perpetrated several advances during the last decades, leveraging the high performance computing technology and spreading its applicability in a wide variety of fields such as structural analysis, oil exploration, atmospheric simulation, weather prediction, seismic data processing, bio-informatics, defense applications and, most recently, molecular dynamics [1]. One of the main challenges in high performance computing (HPC) is how to schedule jobs from different users while providing performance and fairness guarantees. Classic scheduling algorithms, such as FCFS (First-Come-First-Served), backfilling and priority queues, are not well-adapted for multi-user environments as their main goal is to achieve optimal system performance measured by makespan or throughput. However, users are selfish—rather than the overall system efficiency, each user cares about the performance of his own jobs.

High performance computing (HPC) systems, like clusters, grids, supercomputers and desktop grids are shared by many users who, in order to execute their jobs, compete for system's resources. The resources of these systems are commonly managed by a centralized scheduler that accepts job submissions and assigns resources. The BOINC platform [2] is one of the most representative examples of a modern HPC system. BOINC manages over $580,000$ hosts that deliver over $2,300$ TeraFLOPs per day to several projects. Each project has its own goals and distinct processing needs that can be represented by objective functions. As early BOINC projects had large workload, BOINC's original goal was to provide high throughput (completing large number of jobs). Yet, popularity of BOINC attracted other types of projects. Nowadays, response-time users are increasingly common [3]. Their jobs are divided into successive campaigns (batches of independent jobs) released sequentially over time. For such users, throughput is not meaningful; such users prefer a low flow time of campaigns. How to take advantage of this multi-user submission pattern is a problem that has not been well-addressed by the HPC community.

In this work, we analyze scenarios with successive submissions over time issued from different users. Each submission contains at least one sequential job and the set of jobs of one submission is what we call a *campaign*. In a campaign, the jobs are independent, sequential, and can be processed in parallel. However, there is a barrier at the end of each campaign. Any job from a campaign cannot be started until all the jobs from the previous campaign are completed. In other words, as the submission of a new campaign depends on the outcome of the previous campaign, campaigns belonging to a single user must be scheduled one after the other. The objective of each user is to minimize the flow time of each campaign; the flow time is defined as the time interval between the campaign's submission and the completion of the last task of the campaign. The problem of Campaign Scheduling is an extension of the multi-users scheduling problem (MUSP, [4]). In MUSP, each user has a set of sequential jobs to be scheduled; however, unlike in the campaign scheduling, user's goal was either: the makespan or the sum of job's completion times.

We show that the campaign structure can be used to achieve better schedule than proposed by classic algorithms. More

This work has been supported by the French-Polish bilateral scientific cooperation programme "Polonium" and by the CAPES/COFECUB Program (project number 4971/11-6)

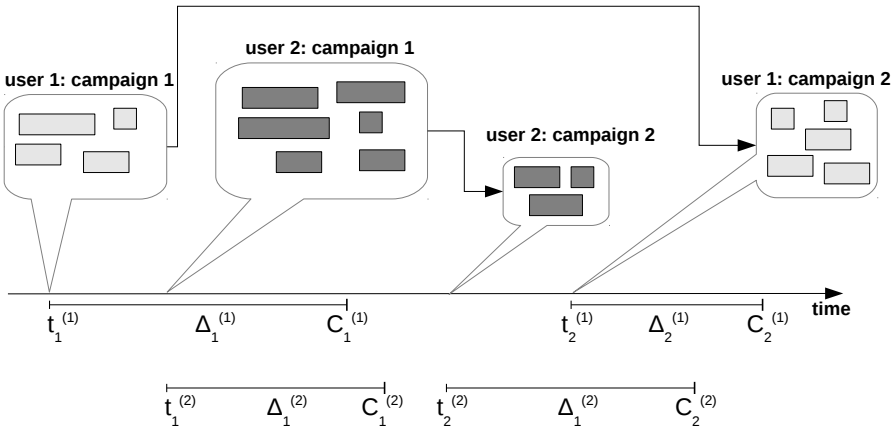

Fig. 1. Campaign Scheduling with 2 users (user 1 in light gray, user 2 in dark gray)

TABLE I
SUMMARY OF NOTATION

k	number of users
(u)	user index
α^u	number of campaigns of user u
i	campaign index
$t_i^{(u)}$	submission time of campaign i of user u
$n_i^{(u)}$	number of jobs of campaign i of user u
$\triangle_i^{(u)}$	length of campaign i of user u
$J_{i,j}^{(u)}$	job J of campaign i of user u
$\sigma_{i,j}^{(u)}$	start time of job $J_{i,j}^{(u)}$
$C_{i,j}^{(u)}$	completion time of job $J_{i,j}^{(u)}$
$C_i^{(u)}$	completion time of a campaign i of user u

specifically, we propose an algorithm that makes a compromise between user fairness and execution performance. We bound the flow time of user's campaigns by a function proportional to the number of users competing for the resources and the expected execution time of a campaign on a machine entirely dedicated to a single user. The function sets deadlines to each campaign; the scheduling algorithm orders campaigns by its deadlines.

The rest of this paper is organized as follows. In the next section, we present a formal model of Campaign Scheduling. In Section III, we give an overview of the state-of-the-art of scheduling with multiple users. Section IV is devoted to the analysis of the offline problem for a single user. In Section V, we extend the analysis of the offline problem to encompass multiple users. Section VI is dedicated to the analysis of the online problem regarding fairness among users and to the description of our solution. The theoretical results are assessed through simulations (Section VII). Finally, we present our conclusions and future work in Section VIII.

II. DEFINITIONS AND NOTATIONS

Our model consists of k users (indexed by u) sharing the resources on a parallel platform composed of m identical processors (indexed by q) managed by a centralized scheduler. Figure 1 illustrates the model; Table I summarizes the notation.

A user workflow is composed by $\gamma^{(u)}$ sequential campaigns. Each user campaign $i \in [1, \gamma^{(u)}]$ is submitted at a time denoted by $t_i^{(u)}$ and is composed of independent and non-preemptive sequential jobs. The submission time $t_i^{(u)}$ of a campaign is not fixed as it depends on the termination of the previous campaign. A campaign is defined as the set of jobs released in one submission, with $n_i^{(u)}$ being the number of jobs in the i-th campaign and $n^{(u)}$ the total number of jobs released in the workflow [1]. $J_{i,j}^{(u)}$ denotes a job from the i-th campaign issued by user u; j is the index of the job in a campaign; $p_{i,j}^{(u)}$ is the known job length (clairvoyant scheduling model). The job start time is denoted by $\sigma_{i,j}^{(u)}$ while its completion time is denoted by $C_{i,j}^{(u)}$. As any job from campaign i cannot be started until all the jobs from the previous campaign $i-1$ have been completed, $\min_j \sigma_{i,j}^{(u)} \geq \max_{j'} C_{i-1,j'}^{(u)}$, where $0 < i < \gamma^{(u)}$.

The completion time of campaign i is denoted by $C_i^{(u)}$ ($C_i^{(u)} \triangleq \max_j C_{i,j}^{(u)}$) and the campaign's length of user u is denoted by $\Delta_i^{(u)} \triangleq C_i^{(u)} - t_i^{(u)}$. There is no idle time between campaigns; thus, campaign's length can be also denoted by $\Delta_i^{(u)} \triangleq C_i^{(u)} - C_{i-1}^{(u)}$, where $C_0 = 0$.

For each user u, the objective is to optimize the sum of campaigns' lengths $\Sigma\Delta_i^{(u)} = \Sigma_i(C_i^{(u)} - t_i^{(u)})$. This objective is motivated by interactive applications, in which an user is more interested in results of the individual stages rather than the job throughput obtained in a time frame.

In the *offline* version of the problem, all the campaigns and their jobs are known in advance (the number and the lengths of the jobs). In the *online* problem, campaigns are known only after being submitted.

[1]There is a particular case where the jobs can be infinitely divided into smaller pieces (i.e. fine grained). That is the case of BOINC divisible loads [2]. In BOINC the scheduler can interrupt jobs of one user without loss of computation. Campaigns from two or more users can even be split in several parts and interleaved, without idle spaces between them.

The *offline* model may seem inappropriate at first, since it is hard to predict the number of campaigns and their respective job lengths. But this model is well-suited for some types of applications, like clairvoyant fork-join applications. Furthermore, the *offline* set can always be used as a reference to analyze the competitiveness of *online* algorithms.

Using a natural extension of the classical three-field notation [5], the resulting problem can be denoted by $P|camp|\sum \Delta_i^{(u)}$.

III. State-of-art

The main works related to our research address the problem of scheduling with multiple users competing for the use of the resources. The Multi-Users Scheduling Problem (MUSP) was first studied on a single processor with two users by Agnetis et al. [6] and on multiple processors by Saule and Trystram [4].

Agnetis et al. [6] provided a $< \overline{1}, \overline{1} >$-approximation of the problem for two users on a single processor. Saule and Trystram [4] analyzed the Multi-Users Scheduling Problem (MUSP), namely, the problem of scheduling independent sequential jobs belonging to k different users on m identical processors. In this problem, each user selects an objective function among makespan and sum (weighted or not) of completion times. This is an offline problem where all the jobs are known in advance and can be immediately executed. This problem becomes strongly NP-hard as soon as one user aims at optimizing the makespan. For the case where all users are interested in the makespan, denoted by $MUSP(k : C_{max})$, the authors showed that the problem can not be approximated with a vector ratio better than $(1, 2, \ldots, k)$. This is a natural extension of the approximation ratios notation where the u-th number of the vector corresponds to the approximation ratio on the u-th user objective.

Remark that the term "no vector-ratio better than $(\rho_1, \rho_2, \ldots, \rho_k)$" stands for the component wise relation, which means that the vector-ratio $(\rho_1, \ldots, \rho_{i-1}, \rho_i - \epsilon, \rho_{i+1}, \ldots, \rho_k)$ is not feasible. However, this formulation does not prevent a $(\rho_1, \ldots, \rho_i - \epsilon, \ldots, \rho_j + \epsilon, \ldots, \rho_k)$ vector-ratio from existing.

A. Inapproximability and Pareto optimality

The proof stating that $MUSP(k : C_{max})$ can not be approximated with a performance vector-ratio better than $(1, 2, \ldots, k)$ considers an instance where each user owns m jobs, all the jobs having the same unit length [4]. In this instance, the absolute best makespan that can be achieved for each user is 1 while in any efficient schedule, one user will have a makespan of 1, another one will have a makespan of 2, and so on. Thus, it is impossible to obtain an algorithm that guaranties a vector-ratio better than $(1, 2, \ldots, k)$.

Note that in an efficient schedule, the set of jobs of one user is scheduled all at once, as single blocks, one after the other. If we take a job scheduled at time t_i and change its position with a job scheduled at time $t_j > t_i$, we end up with a schedule whose vector-ratio is worst than $(1, 2, \ldots, k)$ since two users will have t_j as their C_{max} values. The resulted vector-ratio will be $(1, 2, \ldots, t_j, \ldots, t_j, \ldots, k)$. This is analogous for $t_i > t_j$. Figure 2 illustrates the trade-offs between the objectives of two users. User 1 (light gray) owns two jobs of length 4 while user 2 (dark gray) has two jobs of lengths 3 and 7. All the scheduling possibilities are presented as points on the graph where the x-axis and y-axis represents the C_{max} of user 1 and user 2 respectively. Point Z represents the C_{max} lower bound of both users (*zenith*) while the upper bound is represented by point N (*nadir*). Points A and E are optimal in the sense that there is no better solution that improves one objective without degrading another one. This is the definition of a Pareto set [7]. Points B and F are derived from A by exploring different positions for the first task of user 2. But both solutions degrades the C_{max} of user 1 without improving the C_{max} of user 2. The same applies for points C and D relative to E. Thus, according to the concept of *Pareto dominance*, we say that B, C, D, and F are Pareto-dominated solutions. In multi-objective optimization problems, we are interested in finding solutions that are not Pareto-dominated by any other solution.

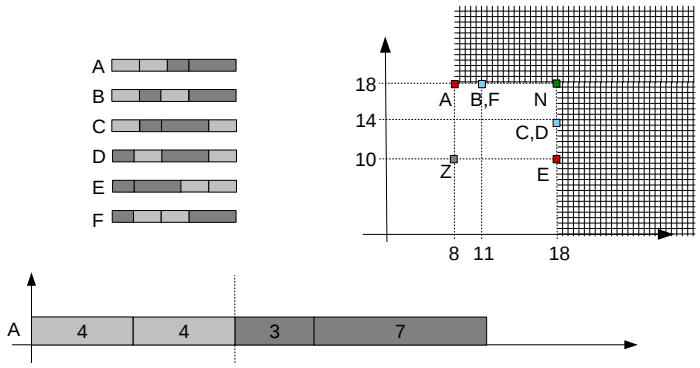

Fig. 2. Pareto optimality for MUSP (user 1 in light gray, user 2 in dark gray). Z denotes the zenith; N the nadire; remaining letters correspond to the schedules on the left.

Based on these observations, Saule and Trystram [4] proposed an algorithm for $MUSP(k : C_{max})$ called MULTICMAX and prove that it is a $(\rho, 2\rho, \ldots, k\rho)$-approximation. Each position of this vector ratio represents the performance ratio of one of the k users and ρ is the approximation ratio of an algorithm for the single-user case (Hochbaum and Shmoys proposed a PTAS to this problem by using dual approximation [8]). The algorithm MULTICMAX works as follows: for each user u, compute a schedule $S^{(u)}$ with a ρ-approximation algorithm. Then, sort the users by non-decreasing values of $C_{max}^{(u)}(S^{(u)})$. Finally, schedule jobs of user u according to $S^{(u)}$ between $\Sigma_{u'<u} C_{max}^{(u')}(S^{(u')})$ and $\Sigma_{u'\leq u} C_{max}^{(u')}(S^{(u')})$. Considering again the example of Figure 2, solution A would be the one generated by this algorithm.

The theorem and the proof stating that MULTICMAX is a $(\rho, 2\rho, \ldots, k\rho)$-approximation of $MUSP(k : C_{max})$ can be seen in [4]. This theorem is valid for a given unknown permutation of users as one user can not know in advance his/her rank in the algorithm. The vector-ratios are computed

relatively to an absolute best solution which is usually unfeasible, but the authors emphasize that this is reasonable since it ensures the performance degradation of each user. Indeed, despite of dealing with sequential tasks, this algorithm generates a final schedule that might contain idle spaces. Those may appear between the users' schedules allowing many portions of the system to remain unusable. As an example, consider an instance with two users, each of them with $\lfloor m/2 \rfloor$ jobs of length p and $m > 1$. MULTICMAX generates a final schedule of length $2p$ using $\lfloor m/2 \rfloor$ machines, while the optimal schedule is of length p, with each user occupying one half of the machines. But this is a situation where the absolute best makespan for both users is feasible, which is not the general case. So, still in this work, the authors presented a class of solutions where each user submits a reasonable number of jobs that follows a linear function on the number of machines. They showed that these solutions are MULTICMAX with $\rho = 2$ and that it contains efficient schedules that are close to the Pareto set.

B. Similarities with Campaign Scheduling

What is interesting regarding our work is that the $MUSP(k : C_{max})$ problem is equivalent to Campaign Scheduling with one campaign and ready jobs (i.e. $\forall u, \forall i, t_i^{(u)} = 0$). We denote this problem by $P|camp = 1|\Delta_i^{(u)}$, where $camp = 1$ stands for one campaign only. This equivalence is due to the fact that $C_{max}^{(u)} = \Delta_i^{(u)}$ for $camp = 1$. The performance guarantees obtained from MULTICMAX can be applied in the same way for this problem in the offline set.

In Campaign Scheduling, despite the arrival of the first k campaigns, there are no synchronicity in the arrival of campaigns from different users. This comes from the fact that campaigns are of different lengths. So, in order to apply MULTICMAX for successive campaigns, we can think of two options:

The first option is to wait for the arrival of one or more user submissions in order to form a batch of jobs. Then, we execute the same algorithm in this batch to obtain a ρ-approximation inside the batch. The problem is that we do not know how much time we have to wait in order to form the batch: some auxiliary offline information (i.e. arrival dates of the submissions) or an extra mechanism (i.e. timeout) are needed. From that point, we can apply an algorithm to schedule the batch. For this scenario, MULTICMAX is a $(\rho + \varepsilon_1, 2\rho + \varepsilon_2, \ldots, k'\rho + \varepsilon_{k'})$-approximation algorithm for $P|camp = 1|\Delta_i^{(u)}$, where ε_i is the time that user i must wait between its submission and the start of the batch and $k' \leq k$ is the number of users participating in the batch.

The second option is to schedule the jobs as soon as the submissions arrive using some order to be determined. If we use the submission order (FCFS), we can not give any guarantee to the user because we do not know when the machines will be available before he/she submits. At the moment of submission, all the machines might be occupied by some jobs execution that was previously submitted by other users. Knowing when the machines will be available leads us to the problem of scheduling with machine release times. This problem is a generalization of the classical multiprocessor scheduling problem where each machine is available only at a machine dependent release time. This was explored first in [9] by Lee and later in [10] by Kellerer. Lee proposed an algorithm called Modified LPT (MLPT), with a guaranteed approximation ratio of $\frac{4}{3}$ that was later shown [11] to be tight. Kellerer extended this result presenting more a sophisticated approximation algorithm with a worst case bound of $\frac{5}{4}$ for the problem. Even so, this result gives no absolute approximations for Campaign Scheduling since it implies that the user must wait an arbitrary time between the submission and the start of execution of his/her jobs. Furthermore, any online algorithm that dynamically changes the order of execution of the jobs is subject to starvation: low priority jobs previously scheduled might be postponed indefinitely as soon as submissions with higher priority jobs arrives before the execution of them.

IV. Offline scheduling of single user's campaigns

In this section, we analyze an offline version of the multi-campaign problem restricted to a single user. The problem constitutes an absolute lower bound for the multi-user case, since no objective value can be lower than the one achieved as the user was the single one in the system. We denote this problem by $P|camp|\sum \Delta_i$. Note that for any "reasonable" (Definition 1) schedule this problem is equivalent to $P|camp|C_{\max}$ (Lemma 1).

We show that the problem is NP-hard. Then, we show that the optimal makespan of $P|camp|\sum \Delta_i$ is at most γ times longer than the optimal makespan of an instance with the same jobs, but no campaigns ($P||C_{\max}$). Finally, we show that a ρ-approximation algorithm for $P||C_{\max}$ is also a ρ-approximation for $P|camp|\sum \Delta_i$.

Proposition 1: $P|camp|\sum \Delta_i$ is NP-hard. The boundary problem is $P2|q_i = 1, camp|C_{max}$ (sequential tasks, arbitrary processing times).

Proof: (straightforward) The proof is by reduction from the two-processor scheduling problem $P2||C_{max}$. An instance of $P2||C_{max}$ can be converted to an instance of $P2|q_i = 1, camp|C$ by placing all the jobs in the same campaign ($J_j \rightarrow J_{1,j}$). ■

For the subsequent analysis, we restrict the type of analyzed schedules to *campaign-compact* schedules.

Definition 1: A *campaign-compact* schedule ς_{cc} (also called a non-delay schedule) is a schedule in which jobs from a subsequent campaign start immediately after the completion of the previous campaign, i.e., $\forall i \in \{1, \ldots, \gamma-1\} \exists j : s_{(i+1),j} = C_i$.

A schedule that is not campaign-compact can be transformed to a campaign-compact schedule. The transformation reduces completion times of some jobs and the length of the

whole schedule. Thus, the optimal schedule is also campaign-compact.

The following lemma binds the completion time C_i of a campaign with the durations; moreover, it shows that $C_\gamma = C_{\max} = \sum \Delta_i$.

Lemma 1: Completion time of i-th campaign C_i is equal to the sum of durations of the previous campaigns and the current campaign: $\forall_i C_i = \sum_{i'=1}^{i} \Delta_{i'}$

Proof: Follows directly from the definition of Δ_i. ∎

The notion of a campaign restricts the set of feasible schedules; thus the optimal schedule for the campaign problem $P|camp|\sum \Delta_i$ might be longer than the optimal schedule for a similar instance with the same jobs, but no constraints.

Proposition 2: An optimal schedule for an instance of the problem $P|camp|C_{max}$ with γ campaigns is at most γ times longer than the optimal schedule for an instance of $P||C_{max}$ with the same jobs, that is: $C^*_{max} \leq \gamma C^{nc,*}_{max}$, where $C^{nc,*}_{max}$ denotes the optimal C_{max} of $P||C_{max}$ (i.e. "nc" for no Campaign Scheduling). This bound is tight.

Proof: $C^*_{max} = \sum_i \Delta^*_i$ (Lemma 1.), where Δ^*_i is the duration of the i-th campaign in the optimal schedule. $\Delta^*_i \leq C^{nc,*}_{max}$, as each campaign is a subset of jobs. Thus $C^*_{max} \leq \gamma C^{nc,*}_{max}$.

For the tightness of the bound, consider an instance with $n = m$ jobs of a unit length. An optimal schedule with no campaigns schedules all the jobs in parallel, with the completion time of $C^{nc}_{max} = 1$. If all the jobs belong to different campaigns ($\gamma = n$), they must be executed sequentially, thus $C^*_{max} = \gamma$. ∎

Consequently, the notion of campaign "costs" the system at most γ, the total number of campaigns.

How should the jobs and the campaigns be scheduled? The following result states that we can use any ρ-competitive algorithm for the standard job scheduling problem to obtain a ρ-approximation of the campaign problem.

Proposition 3: Any scheduling algorithm A that is an ρ-approximation for $P||C_{max}$ is also an ρ-approximation for $P|camp|\sum \Delta_i$. The algorithm for $P|camp|\sum \Delta_i$ (denoted by camp(A)) creates a schedule by, firstly, performing γ executions of A, that is, executing A separately for jobs from each campaign; then shifting the schedule for i-th campaign by C_{i-1}.

Proof: From Lemma 1, $C_{max} = \sum_i \Delta_i$. As the algorithm used to schedule each campaign is an ρ-approximation, $\Delta_i \leq \rho\Delta^*_i$ (where Δ^*_i is an optimal makespan of i-th campaign). Thus, $C_{max} \leq \rho \sum_i \Delta^*_i$, and as $C^*_{max} = \sum_i \Delta^*_i$ (Lemma1), $C_{max} \leq \rho C^*_{max}$. ∎

In summary, the barrier between subsequent campaigns affects the system by increasing the makespan at most γ times. Yet, the resulting scheduling problem is similar to the standard scheduling; and has the same approximation ratio.

V. Offline scheduling with multiple users

In this section, we study the Campaign Scheduling problem $P|camp|\sum \Delta^{(u)}_i$ with multiple users, each having multiple campaigns. The problem is NP-hard as the boundary problems $P|camp|\sum \Delta_i$ and $P|camp = 1|\Delta^{(u)}_i$ are NP-hard (Section III and Section IV). We show that the problem cannot be approximated better than $(1, 2, \ldots, k)$; then, we apply a multi-campaign version of the MULTICMAX algorithm to obtain $(\rho, 2\rho, \ldots, k\rho)$-approximation. Finally, we argue that this algorithm is not "fair" to users and that another solution should be proposed.

Proposition 4: $P|camp|\sum \Delta^{(u)}_i$ can not be approximated with a performance vector-ratio better than $(1, 2, \ldots, k)$.

Proof: Let us consider the following instance. All campaigns of all k users have m jobs of unit length, where m is the number of machines. All users have the same number n of campaigns. Obviously, for each user independently, the best $\sum \Delta^{(u)}_i$ achievable is equal to n. But, in any efficient schedule, one user will have $\sum \Delta^{(u)}_i$ equal to n, another one will have $\sum \Delta^{(u)}_i$ of $2n$ and so on. Thus, it is impossible to guarantee a vector-ratio better than $(1, 2, \ldots, k)$. ∎

Remark that any permutation of this vector-ratio can be obtained (as explained in Section III). But this vector-ratio is not accomplished by all schedules. Consider for example a schedule that interleaves campaigns of users similarly to the round-robin scheduling. Each $\Delta^{(u)}_i$ will be of length 1. So, in this schedule, one user will have $\sum \Delta^{(u)}_i = kn$, another one will have $kn - 1$, and so on until $kn - k$. Clearly, the corresponding vector-ratio will be worse than $(1, 2, \ldots, k)$.

Based on this observations, we propose an algorithm for the offline problem called MULTICAMP. The algorithm schedules users' campaigns in blocks. First, for each campaign i of each user u, the algorithm computes a schedule $\varsigma^{(u)}_i$ with a ρ-approximation algorithm. Then, for each user, the algorithm puts all its campaigns side by side to form a single block, i.e. a campaign-compact schedule $\varsigma^{(u)}$ (see Definition 1). The schedule $\varsigma^{(u)}$ has length of $\sum_i \Delta^{(u)}_i$. Next, the users are sorted by non-decreasing values of $\sum_i \Delta^{(u)}_i$. Finally, the algorithm puts the block $\varsigma^{(u)}$ of user u in the final schedule between $\sum_{u'<u} \sum_i \Delta^{(u')}_i$ and $\sum_{u'\leq u} \sum_i \Delta^{(u')}_i$.

Proposition 5: MULTICAMP is a $(\rho, 2\rho, \ldots, k\rho)$-approximation algorithm of $P|camp|\sum \Delta^{(u)}_i$.

Proof: First, we have to verify the validity of the final schedule. In fact, the campaign's blocks are scheduled in disjoint intervals of length $\sum_i \Delta^{(u)}_i(\varsigma^{(u)})$ according to $\varsigma^{(u)}$.

Second, we verify that the final schedule as a performance vector-ratio of $(\rho, 2\rho, \ldots, k\rho)$. The users are ordered by increasing values of $\sum_i \Delta_i^{(u)}(S_i^{(u)})$ and each block is scheduled in sequence according to this order. Thus, $\sum_i \Delta_i^{(u)} \leq u \sum \Delta_i^{(u)}(S_i^{(u)})$. Moreover, $\varsigma_i^{(u)}$ was generated by a ρ-approximation algorithm. Thus, $\sum_i \Delta_i^{(u)} \leq u \sum \rho \Delta_i^{(u)*} = u\rho \sum \Delta_i^{(u)*}$. ∎

Usually, a performance vector-ratio depending on k, a parameter of an instance, is not an efficient solution. However, the optimal value of each particular objective function is obtained by scheduling the jobs of the user as if she was the only user on the system. The performance vector-ratio represents the distance between the schedule given by our algorithm and these absolutely optimal solutions for every user (the zenith point). In the general case, the zenith solution is not feasible.

In MULTICAMP, although the response time of entire campaign blocks (complete workflows) are bounded, individual campaigns are arbitrarily delayed. User u may be disappointed that she has to wait $u-1$ workflows before her workflow starts executing. It should be more fair to the users that everybody was equally affected or proportionally affected according to her workflow length.

In Section III we saw that, considering just one shot of campaigns, interleaving jobs is not a good idea as it leads to inefficient, Pareto dominated solutions. However, the efficient solutions inherits an imbalance between users with respect to the response time, given that campaigns are ordered according to their lengths. With multiple campaigns, we can reduce this imbalance by giving higher priority in the next scheduling decisions for the users that were "unhappy" in the previous ones. We tackle this problem in the next section.

VI. Online scheduling with multiple users

In the online problem, jobs belonging to campaigns are known only at their submission time. An online algorithm should consider both between-user fairness and system performance. As we do not know the lengths of the workflows, both criteria should be respected at any time. In this section, we argue that the maximum *stretch* should be the considered fairness measure. Then, we show that First-Come-First-Served (FCFS) algorithm can produce an arbitrary large stretch, thus it is not applicable for fair scheduling. Finally, we present FAIRCAMP, a scheduling algorithm that bounds the stretch and, at the same time, is k-competitive.

A. Measuring fairness by max stretch

In the context of sharing resources by different parties, fairness is defined as resources being equally available. However, there is no universally accepted model for fairness when scheduling jobs; there are various interpretations of "availability" and "equality".

In this work, as each user is interested in the sum of campaigns response times, we tackle the problem of fairness by measuring *stretch*. More specifically, stretch is defined as the time the task spent in the system normalized by its processing time. Thus, the idea is to optimize a stretch-like function derived from the sum of campaigns response times. We denote this function as $D^{(u)} = \sum_i \Delta_i^{(u)} / \sum_i \Delta_i^{(u)*}$, where $\sum_i \Delta_i^{(u)*}$ is the optimal sum of campaigns response time for user u. To guarantee fairness is necessary to choose an aggregation function. Three of them are normally considered, corresponding to the standard norms L_∞, L_1, L_2: minimizing maximum of stretches, sum of stretches and product of stretches. Ideally, a fair schedule should minimize the max-stretch while producing the same stretch for all users. Thus, minimizing the max-stretch ($max_u D^{(u)}$) is the natural choice.

B. Fairness in FCFS schedules

FCFS is one of the most commonly used policies in job scheduling; however, as we show in this section, it is arbitrary bad both for the fairness and for the system. In FCFS, the jobs are scheduled as soon as they arrive; the start time depends on the availability of the machines at the time of the submission. When an user u submits a campaign i at a moment $t_i^{(u)}$, it will be scheduled after all the previous submissions sent before $t_i^{(u)}$. This means that the jobs from user u may start its execution in a moment $t_i^{(u)} + \varepsilon$, where the length of ε is not bounded. The following proposition formally states this result (see also Figure 3).

Proposition 6: FCFS is at least α-approximation algorithm for $P|camp|\sum \Delta^{(u)}$ and for the max-stretch where α is the ratio between the longest and the shortest workflow.

Proof: Consider m processors and $k = 2$ users. Consider that both users have γ submissions to be made and each submission is composed of m jobs. Jobs of user 1 are of length p and jobs of user 2 are of length 1. Now, consider the following situation: user 1 makes his/her first submission at time 0 and the user 2 makes his/her first submission immediately after the first user in a time $\varepsilon \approx 0$. As the jobs will be executed following a FIFO order, initially all the machines are occupied by the jobs from user 1. User 2 will have to wait a time $p - \varepsilon$ before his/her tasks get scheduled. While the optimal schedule for the first campaign of user 2 is of length 1, the schedule given by FCFS is of length $p + 1 - \varepsilon \approx p + 1$. Consequently, if this submission order is repeated in the subsequent campaigns, the schedule given by FCFS is of length $\approx (p + 1)\gamma$ while the optimal sum of campaigns' completion times ($\sum \Delta^{(u)}$) is of length γ. Thus, we can not give any guarantees to the user 2 for the sum of campaigns' length based solely on an approximation from his/her particular optimal schedule. ∎

Consequently, in the online problem, FCFS is an arbitrary bad strategy for fairness.

C. Fair Online Schedules with FAIRCAMP

In order to maintain fairness between processes, a standard operating system commonly uses a round-robin (RR) strategy

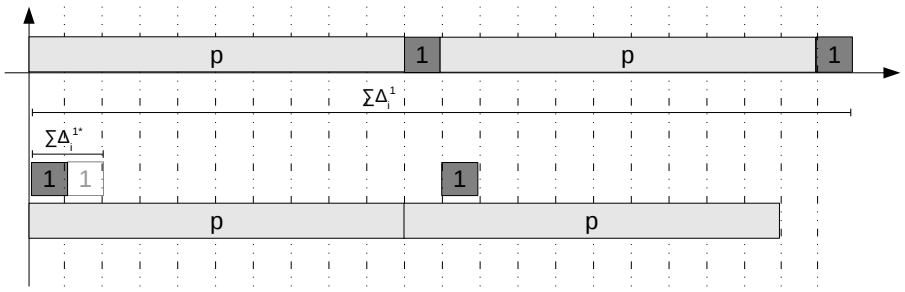

Fig. 3. FCFS competitiveness. $k = 2$, $\gamma = 2$, user 1 in light gray, user 2 in dark gray. Max-stretch is $(2p+2)/2 \approx p$ (for large p); the optimal max-stretch is $(2p+2)/(2p) \approx 1$ (for large p). The faded campaign represents the optimal position of the second campaign for user 2.

with preemption. Each process is given the processor for a fixed quantum of time; when the time expires, the process is preempted and put at the end of the queue. This strategy results in reasonable fairness among processes, as each process is slowed down proportionally to the total *number* of processes, in contrast to the total *load*, as in FCFS or Longest Processing Time (LPT). The problem is that, in HPC scheduling, preemption is usually impossible.

FAIRCAMP, the scheduling algorithm we propose, approximates fairness of the round-robin preemptive schedules. For each user, the maximum delay in execution of the current campaign depends only on the previous campaigns of the user and the total number of users. For each submitted campaign we set a deadline $d_i^{(u)}$:

$$d_i^{(u)} = k \cdot \Delta_i^{(u)}(\varsigma_i^{(u)}) + d_{i-1}^{(u)}, \tag{1}$$

where $\Delta_i^{(u)}(\varsigma_i^{(u)})$ is the length of the campaign schedule $\varsigma_i^{(u)}$ generated by a ρ-approximation algorithm using all the processors. The deadline is proportional to the number k of users; and shifted by the deadline of the previous campaign $d_{i-1}^{(u)}$. This deadline is computed based on the max-stretch of $k\rho$ for the whole campaign workflow. If the schedule is feasible, the deadlines guarantee that no user is worse-off than if the supercomputer was shared between the users with a Round-Robin, preemptive scheduler. When a campaign is submitted, we do not know if the workflow is ended or not. Thus, setting a deadline for each campaign is a way to ensure that the max-stretch will be respected for the whole workflow. For example: with $k = 2$ users, user (1) has $\gamma^{(1)} = 2$ campaigns of calculated length $\Delta_1^{(1)} = 5$ and $\Delta_2^{(1)} = 3$, their respective deadlines are $d_1^{(1)} = 2 \cdot 5 = 10$ and $d_2^{(1)} = 2 \cdot 3 + 10 = 16$; user (2) has $\gamma^{(2)} = 3$ campaigns of calculated length $\Delta_1^{(2)} = 3$, $\Delta_2^{(2)} = 3$ and $\Delta_3^{(2)} = 10$, their deadlines are $d_1^{(2)} = 6$, $d_2^{(2)} = 12$, and $d_3^{(2)} = 22$.

FAIRCAMP uses earliest deadline first (EDF) to choose the campaign to be executed. FAIRCAMP is composed of two modules that run in parallel. They share a queue in order to place the campaigns that are ready to execute. The first module generates information about arriving campaigns and puts them into a queue. The second module chooses campaigns by EDF and schedule them as soon the resources are available.

1) Details of FAIRCAMP: The gathering module handles submissions and computes the deadlines.

```
1: ready ← new(queue)
2: while TRUE do
3:    wait(campaign i, user u)
4:    ς_i^(u) ← ρ(i, resources)
5:    Δ_i^(u) ← length(ς_i^(u))
6:    d_i^(u) ← k · Δ_i^(u) + d_{i-1}^(u)
7:    T_i^(u) ← (ς_i^(u), d_i^(u))
8:    enqueue(T_i^(u), ready)
9: end while.
```

In line 3, the process stops and waits some user u to release a new campaign i. Next, in line 4, we generate a schedule for i and the resources list based on a polynomial ρ-approximation algorithm. Then the campaign deadline is calculated from the number of users k, the schedule length $\Delta_i^{(u)}$ and the deadline of the previous user campaign (line 6 and 7). Finally, a tuple containing the schedule and its deadline is generated and added to the queue.

The scheduling module performs EDF over the submitted campaigns.

```
1: while TRUE do
2:    if not empty(ready) and available(resources) then
3:       T_next ← NULL
4:       S_next ← NULL
5:       d_next ← ∞
6:       for all T_i^(u) on ready do
7:          if d(T_i^(u)) < d_next then
8:             T_next ← T_i^(u)
9:             d_next ← d(T_i^(u))
10:         end if
11:      end for
12:      dequeue(Q, T_next)
13:      update(resources, S(T_next))
14:   end if
15: end while.
```

This procedure iterates over the tuples on the queue searching which one has the schedule with the lowest deadline (lines 6-11). In the end, the chosen campaign schedule is placed on the resources (line 13).

2) Feasibility of FAIRCAMP: FAIRCAMP uses deadlines to guarantee fairness. Here, we show that a schedule produced by FAIRCAMP is always feasible, i.e., all the deadlines are met. First, we assume that a campaign misses its deadline and then we show that this leads to a contradiction.

Proposition 7: In a schedule S produced by FAIRCAMP, all campaigns finish their executions before their respective deadlines.

Proof: Consider m processors and k users. For each campaign i, a schedule $\varsigma_i^{(u)}$ is generated whose length is $\Delta_i^{(u)}(\varsigma_i^{(u)})$. This length is calculated by a ρ-approximation algorithm using all the processors. According to FAIRCAMP, each campaign has deadline $d_i^{(u)} = k \cdot \Delta_i^{(u)}(\varsigma_i^{(u)}) + d_{i-1}^{(u)} = k \cdot \Delta_i^{(u)}(\varsigma_i^{(u)}) + k \cdot \Delta_{i-1}^{(u)}(\varsigma_{i-1}^{(u)}) \dots k \cdot \Delta_1^{(u)}(\varsigma_1^{(u)}) = k \cdot \sum_i \Delta_i^{(u)}(\varsigma_i^{(u)})$. This deadline states that the length of each user workflow can be stretched at a maximum factor of k. In other words, $d_i^{(u)}$ also can be seen as the deadline of the partial workload (from the first campaign until campaign i).

Now, consider a schedule S constructed by FAIRCAMP where at least one campaign misses its deadline and, without loss of generality, let campaign i from user u be the first campaign to miss its deadline on S. By definition, $d_i^{(u)} = k \cdot \sum_i \Delta_i^{(u)}(\varsigma_i^{(u)})$, where $\sum_i \Delta_i^{(u)}(\varsigma_i^{(u)})$ is the length of the (partial) workflow issued from u.

Two conclusions can be observed from this scenario. First, the sum of the length of the workloads issued from the other $k-1$ users, between the beginning of the schedule ($t = 0$) and the beginning of campaign i, is bigger than $(k-1) \cdot \sum_i \Delta_i^{(u)}(\varsigma_i^{(u)})$. Otherwise, the schedule would have an idle space, violating the campaign-compact constraint (Definition 1). All the campaigns after this idle space (including campaign i) could be shifted to end before $d_i^{(u)}$. So, more formally,

- $\sum_{v \neq u}^{k} \sum_j^{\gamma^{(v)}} \Delta_j^{(v)}(\varsigma_j^{(v)}) > (k-1) \cdot \sum_i \Delta_i^{(u)}(\varsigma_i^{(u)})$,

where $\gamma^{(v)}$ is the number of campaigns on each partial workload for $\forall v \neq u$.

Second, the deadline of each partial workload is equal or less than the deadline of i, otherwise it would be not executed before i. More formally, $\forall v \neq u$:

- $d_{\gamma^{(v)}}^{(v)} \leq d_i^{(u)}$;
- $k \cdot \sum_j^{\gamma^{(v)}} \Delta_j^{(v)}(\varsigma_j^{(v)}) \leq k \cdot \sum_i \Delta_i^{(u)}(\varsigma_i^{(u)})$;
- $(k-1) \cdot \sum_j^{\gamma^{(v)}} \Delta_j^{(v)}(\varsigma_j^{(v)}) \leq (k-1) \cdot \sum_i \Delta_i^{(u)}(\varsigma_i^{(u)})$.

But this clearly contradicts the first conclusion. ∎

3) Performance of FAIRCAMP: We analyze FAIRCAMP from two perspectives. First, Corollary 1 shows that FAIRCAMP is efficient for the performance of each user, as $\sum \Delta_i^{(u)}$ is increased by at most $k\rho$ measured relatively to a system dedicated to the user. Second, Theorem 1 demonstrates that FAIRCAMP is fair, as it is a ρ-approximation for minimizing the maximum stretch over all users.

Corollary 1: FAIRCAMP is $(k\rho, \dots, k\rho)$-approximation for $P|camp|\sum \Delta_i^{(u)}$.

Proof: By definition, the length of an user workflow is denoted by $\sum \Delta_i^{(u)}$. The deadline of a workflow is the deadline of the last campaign $d_{\gamma^{(u)}}^{(u)} = k \cdot \sum_{i=1}^{\gamma^{(u)}} \Delta_i^{(u)}(\varsigma_i^{(u)}) = k\rho \cdot \sum_{i=1}^{\gamma^{(u)}} \Delta_i^{(u)*}$, where $\sum_{i=1}^{\gamma^{(u)}} \Delta_i^{(u)*}$ is the optimal length of the workflow. Since all deadlines are met, each workflow is stretched by a factor of $k\rho$, at maximum. ∎

Theorem 1: FAIRCAMP is ρ-approximation for max-stretch that does not depend on k.

Proof: Let us consider a solution S constructed by FAIRCAMP for an instance of $P|camp|\Delta_i^{(u)}$. Now, consider without loss of generality [2], that the first l scheduled campaigns belong to user u while the campaign $l+1$ belongs to user v. Assume that s_l is the start time and C_l is completion time of campaign l. Similarly, s_{l+1} and C_{l+1} are the start and completion time of campaign $l+1$. As campaign $l+1$ starts immediately after campaign l, $s_{l+1} = C_l$. The arrival time of campaigns l and $l+1$ are $t_l = s_l$ and $t_{l+1} = 0$, respectively.

As S was constructed by FAIRCAMP, the deadline of $l+1$ is bigger or equal to the deadline of l, otherwise, campaign $l+1$ would be scheduled earlier (after one of the l first campaigns). So, $k \cdot \Delta_{l+1}^{(v)}(\varsigma_{l+1}^{(v)}) \geq k \cdot \sum_1^l \Delta_i^{(u)}(\varsigma_i^{(u)})$. We also can denote this as $C_{l+1} - C_l \geq C_l$ or, more conveniently, $C_{l+1} \geq 2C_l$.

Regarding only these $l+1$ campaigns, the stretch of user u is $D^u = C_l / \sum_1^l \Delta_i^{(u)*} = \sum_1^l \Delta_i^{(u)} / \sum_1^l \Delta_i^{(u)*} = \rho \sum_1^l \Delta_i^{(u)*} / \sum_1^l \Delta_i^{(u)*} = \rho$, while user v has $D^v = (C_{l+1} - t_{l+1}) / \Delta_{l+1}^{(v)*}) = \rho(C_{l+1} - t_{l+1}) / (C_{l+1} - s_{l+1}) = \rho(C_{l+1}) / (C_{l+1} - C_l)$. As $(C_{l+1}) / (C_{l+1} - C_l) > 1$, then $D^v > \rho$ and $max(D^u, D^v) = D^v$.

Assume by contradiction that a better solution S' is achieved by changing the positions of campaigns l and $l+1$. In this solution, the stretch of u is $D'^u = \rho(C_{l+1} - s_l) / (C_l - s_l)$ and the stretch of user v is $D'^v = \rho(C_{l+1} - C_l + s_l) / (C_{l+1} - C_l)$. For this solution to be better, both stretches of S' should be lower than D^v:

- $D^v > D'^v$
- $\rho(C_{l+1}) / (C_{l+1} - C_l) > \rho(C_{l+1} - C_l + s_l) / (C_{l+1} - C_l)$
- $(C_{l+1}) > (C_{l+1} - C_l + s_l)$.

[2] One can argue that considering only campaigns $l+1$ and l causes the proof to lose its generality. But, in fact, we could place $l+1$ between any place before l since this does not alter the new position of l as well as the final contradiction finded by the proof. Another argument is that we are considering only the first $l+1$ campaigns from the beginning of the schedule, but the same logic applies if we consider any campaign completion time C_i as the start point. Furthermore, changing the position of $l+1$ with a later campaign, would produce a stretch even bigger for campaign $l+1$ and clearly is not a better solution.

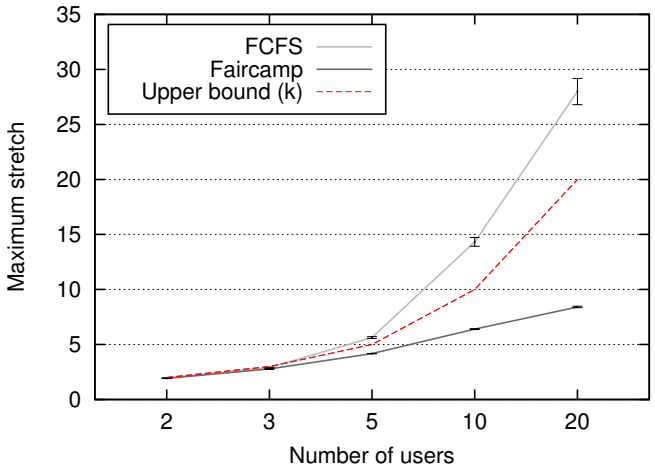

Fig. 4. FCFS vs FAIRCAMP; each point is an average over 10.000 instances; error bars denote 95% confidence intervals

Since $C_l + s_l$ is positive, we proved that $D^v > D'^v$. Now, for D'^u:

- $D^v > D'^u$
- $\rho(C_{l+1})/(C_{l+1} - C_l) > \rho(C_{l+1} - s_l)/(C_l - s_l)$
- $C_{l+1}(C_l - s_l) > (C_{l+1} - s_l)(C_{l+1} - C_l)$
- $C_{l+1}C_l - C_{l+1}s_l > C_{l+1}^2 - C_{l+1}C_l - C_{l+1}s_l + C_l s_l$
- $C_{l+1} < 2C_l - (C_l s_l / C_{l+1})$.

But $C_{l+1} \geq 2C_l$, so $D^v > D'^u$ is false which contradicts the assumption. ■

VII. Simulations

In this section, we present a simulation that demonstrates that FAIRCAMP results in lower stretch (and thus, better performance) than FCFS. The simulator plays the role of a centralized scheduler: it takes instances of user workloads as inputs; and it calculates the max-stretch obtained by each algorithm in an environment composed of $m = 10$ identical processors.

Each instance is composed of 10^4 jobs. For each job we set its length p (uniformly taken from the range $[1, 100]$). The job starts a new campaign with probability of 0.1; otherwise, it belongs to the previous campaign. If the job starts a new campaign, we set the owner of this campaign according to a Zipf distribution with exponent equal to 1.4267 which best models submissions behaviors in large social distributed computing environment [12].

We created 10^3 instances for different number of users (k): 2, 3, 5, 10 and 20. The simulator runs both algorithm with the same instances. The results of the simulation are shown on Figure 4. All results are presented with confidence level of 95%.

First, the max stretch of FAIRCAMP (solid line) is always well below the upper bound k (red dashed line).

Second, the results showed that, in systems with at least 5 users, FAIRCAMP results in significantly lower max stretches than FCFS. With 20 users, the max-stretch of FAIRCAMP is approximately 3.4 times lower. Good results are also achieved with 5 and 10 users, with improvements of 1.35 and 2.24, respectively. With few users, the difference is irrelevant.

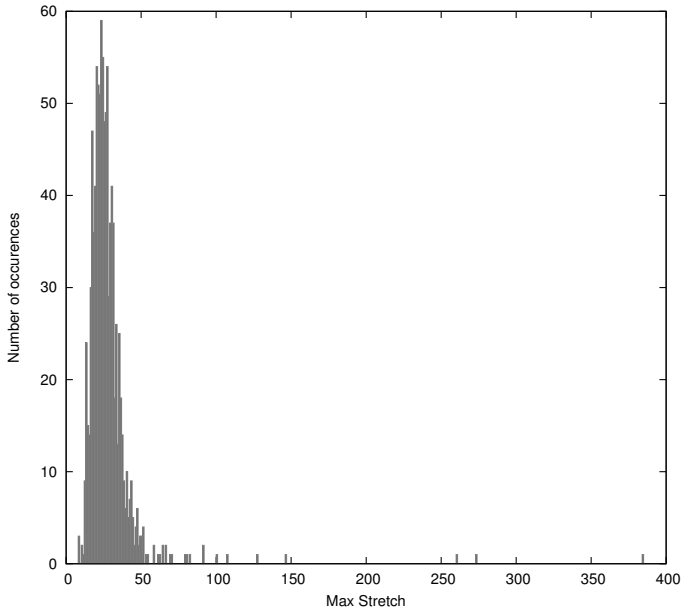

Fig. 5. Max-stretch distribution for FCFS with 20 users

This behavior is motivated by the Zipf distribution that assigns campaigns to their owners. According to Zipf's law, the most frequent user has probability twice the second most frequent user, three times the third most frequent user, etc. The more users, the greater the difference in number of campaigns between the first user and the last one in the frequency rank. So, it is more likely that FCFS generates some schedules that will affect users with few campaigns and, consequently, will result in high values for the max-stretch.

Figure 5 depicts the distribution of max-stretch for FCFS with 20 users. The overwhelming majority of instances produces max-stretch values between 10 and 50 (which is higher than approx. 8 produced by FAIRCAMP). Nonetheless, in a few instances, the max-stretches are extremely high, some of them reaching the hundreds (7 of them can be seen in the Figure 5). This happens because, despite its clear predominance around the average, FCFS is not bounded since it does not take stretch into account, like FAIRCAMP.

VIII. Conclusion

The popularization of parallel systems leveraged by scientific clouds and grids promotes the emergence of new user profiles and applications, whose needs impose new challenges for the HPC community, particularly for researches on scheduling theory.

In this work, we define the Campaign Scheduling problem. In Campaign Scheduling, each user submits campaigns of many independent jobs; yet all the jobs from a campaign must be completed before the first job from the next campaign starts.

We believe that Campaign Scheduling models an emergent pattern of execution of increasing number of HPC workloads.

We demonstrated that the problem is NP-hard. We proposed an approximation algorithm called FAIRCAMP. The algorithm achieves fairness among users by guaranteeing that no user is worse-off than in a time-shared machine. The guarantee is based on deadlines that bound the max-stretch of user workloads. At the same time FAIRCAMP gives performance guarantees from the absolute optimal solutions. We proved that for each user FAIRCAMP is $k\rho$-competitive for the flow time comparing with a dedicated, single-user system (where k is the number of users, and ρ is the approximation ratio of the auxiliary scheduling algorithm). We also proved that FAIRCAMP is ρ-approximate for the max-stretch comparing with a shared system with an optimal scheduler.

We verified the average performance of FAIRCAMP by simulation. From $k = 5$ users, FAIRCAMP schedules result in max stretches significantly lower than FCFS; with $k = 20$ users, FAIRCAMP schedules have the maximum stretch 3.4 times lower.

Our future work tackles both theory and practice of Campaign Scheduling. We plan to extend the theoretical model by considering delays between subsequent campaigns of a single user (the delays model the time needed for, e.g., analysis and interpretation of the obtained results). We also plan to derive the quantitative characteristics of users' campaigns from HPC systems' workloads. We intend to construct probabilistic profiles of users: a profile will define campaign's average workload and a submission frequency. We expect that the distribution of workloads between users will follow a power-law distribution. Classic systems (with the average job performance as the main goal) mix high- and low-volume users; thus large number of low-volume users are affected by a few high-volume ones. This phenomenon stressed the need for scheduling algorithms that use fairness as its main criterion.

ACKNOWLEDGMENT

Krzysztof Rzadca thanks Jaroslaw Zola for the inspiration for the campaign scheduling model.

Krzysztof Rzadca is partly supported by Foundation for Polish Science "Homing Plus" Programme co-financed by the European Regional Development Fund (Innovative Economy Operational Programme 2007-2013).

REFERENCES

[1] S. Emmott and S. Rison, "Towards 2020 science," Working Group Reserach. Microsoft Research Cambridge, Tech. Rep., 2006.

[2] D. P. Anderson, "Boinc: A system for public-resource computing and storage," in *5th IEEE/ACM International Workshop on Grid Computing*, 2004, pp. 4–10.

[3] B. Donassolo, A. Legrand, and C. Geyer, "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*, ser. CCGRID '11, 2011, pp. 144–153.

[4] E. Saule and D. Trystram, "Multi-users scheduling in parallel systems," in *Proc. of IEEE International Parallel and Distributed Processing Symposium 2009*, Washington, DC, USA, may 2009, pp. 1–9.

[5] R. Graham, E. Lawler, J. K. Lenstra, and A. R. Kan, in *Optimization and approximation in deterministic sequencing and scheduling theory: A survey*, ser. Annals of Discrete Mathematics. Elsevier, 1979, vol. 5, pp. 287–326.

[6] A. Agnetis, P. B. Mirchandani, D. Pacciarelli, and A. Pacifici, "Scheduling problems with two competing agents," *Operations Research*, vol. 52, no. 2, pp. 229–242, 2004.

[7] M. Voorneveld, "Characterization of Pareto dominance," *Operations Research Letters*, vol. 31, pp. 7–11, january 2003.

[8] D. S. Hochbaum and D. B. Shmoys, "Using dual approximation algorithms for scheduling problems theoretical and practical results," *J. ACM*, vol. 34, pp. 144–162, January 1987. [Online]. Available: http://doi.acm.org/10.1145/7531.7535

[9] C.-Y. Lee, "Parallel machines scheduling with nonsimultaneous machine available time," *Discrete Appl. Math.*, vol. 30, pp. 53–61, January 1991. [Online]. Available: http://dx.doi.org/10.1016/0166-218X(91)90013-M

[10] H. Kellerer, "Algorithms for multiprocessor scheduling with machine release times," *IIE Transactions*, vol. 30, pp. 991–999, 1998, 10.1023/A:1007526827236. [Online]. Available: http://dx.doi.org/10.1023/A:1007526827236

[11] L. Guo-Hui, "The exact bound of Lee's MLPT," *Discrete Applied Mathematics*, vol. 85, no. 3, pp. 251 – 254, 1998. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0166218X9700139X

[12] A. Iosup, C. Dumitrescu, D. Epema, H. Li, and L. Wolters, "How are real grids used? the analysis of four grid traces and its implications," in *Proceedings of the 7th IEEE/ACM International Conference on Grid Computing*, ser. GRID '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 262–269. [Online]. Available: http://dx.doi.org/10.1109/ICGRID.2006.311024