Fair Scheduling for Multiple Submissions

Joseph Emeras, Vinicius Pinheiro, Krzysztof Rzadca, Denis Trystram

Les rapports de recherche du LIG

Fair Scheduling for Multiple Submissions

Joseph EMERAS, PhD student, LIG, University of Grenoble (UJF), CEA/CNRS, Grenoble
Vinicius PINHEIRO, PhD student, IME, University of Sao Paulo, Brasil
Krzysztof RZADCA, Assistant professor, MIMUW, University of Warsaw, Poland
Denis TRYSTRAM, Professor, LIG, University of Grenoble (Grenoble-INP), Grenoble

RR-LIG-033
février 2013

http://rr.liglab.fr
ISSN 2105-0422

L I G

Fair Scheduling for Multiple Submissions

Joseph Emeras
Laboratoire d'Informatique de Grenoble
CEA - CNRS, France
joseph.emeras@imag.fr

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

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

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

***Abstract*—Today, most available parallel environments support multiple users executing their jobs in a common shared infrastructure. In such environments, when scheduling individually important jobs from different users, the problem of *fairness* arises. In this work, we propose a fair scheduling algorithm that handles this problem by adopting processor fair-share as a strategy for user fairness. The user's workload follows the Campaign Scheduling model, in which each workload is characterized by the submission of successive sets (or campaigns) of sequential and independent jobs. The algorithm guarantees that the flow time of a campaign is proportional to campaign's size and the total number of users. In contrast to the classic queuing systems, the flow time does not depend on the total system load. Internally, the algorithm maintains a virtual time-sharing schedule in which each user is assigned the same share of processors. The order of completion in the virtual schedule determines the execution order on the real processors. We analyse the performance of the algorithm by simulation on a real parallel machine workload trace.**

Our approach shows that a parallel machine can give a similar type of performance guarantee as a round-robin scheduler, no job preemption been required. Consequently, processors are shared in a proportionally-fair manner among users.

I. Introduction

High performance computing (HPC) systems, like clusters, grids, supercomputers and desktop grids are usually shared by multiple users who compete for the usage of the resources in order to execute their jobs. Most of such systems embrace users (or projects) around a common infrastructure that simplifies resource management and application execution through a centralized scheduler. For instance, the BOINC platform [1] gathers over $580,000$ hosts that deliver over $2,300$ TeraFLOP per day to several projects. In the past, most users were throughput-oriented but popularization of those systems attracted other types of users. Nowadays, response-time users are increasingly common [2]. Workload of response-time users is composed of multiple submissions released sequentially over time [3]–[6]. For such users, the criterion of throughput is not meaningful as they are more interested in the flow time of each submission. How to take advantage of this multi-user submission pattern and how to consider the flow-time objective is a problem that has not been well addressed yet by the HPC community.

In this work, we consider the problem of scheduling campaigns of sequential jobs submitted by multiple users over time in a system composed of identical parallel machines. The campaign scheduling problem was introduced in [7] and analyzed under restrictive assumptions. A user submits campaigns sequentially; each campaign is a set of independent jobs. However, there is a barrier at the end of each campaign. Any job from a campaign cannot start until all the jobs from the previous campaign of the user have been completed. Campaigns generalize the Bag-of-Tasks application model: in Bag-of-Tasks, the application is composed of small jobs, whereas in campaign scheduling the user submits multiple sets of jobs. The campaign scheduling problem models a submission pattern: the user submits a set of jobs, analyses the outcomes and resubmits another set of jobs. In other words, the campaigns from one user must be scheduled one after the other since the submission of a new campaign depends on the outcome of the previous one. As this submission pattern is in fact interactive (submit – analyze – submit – . . .), the objective of each user is to minimize the time each campaign spents in the system (campaign's flow time).

Classic approaches such as FCFS (First-Come-First-Served), backfilling and priority queues are commonly used by actual systems. But these systems are not well-adapted for multi-user environments as they focus exclusively on job characteristics to achieve single objective optimization such as overall makespan, throughput or system utilization [8]–[11]. However, users are selfish: they care about the performance of their jobs, rather than about the whole system. A typical scheduler (such as Maui) would treat between-user fairness as a secondary objective. For instance, users submitting many tasks may be assigned "negative karma", a penalty that reduces the priority of a job.

In this work, we propose a scheduling algorithm that explicitly maintains between-user fairness by guaranteeing the worst-case stretch of each campaign as a function of user's own workload and the number of active users. Fairness is a somewhat fuzzy concept to model theoretically. In scheduling, one of the accepted and used metrics is the stretch—the time the job stays in the system (the flow time) normalized by the job's processing time. The stretch of a job measures how the performance of the job is degraded compared to a system dedicated to this job. Thus, stretch measures the relative responsiveness of the system. Stretch optimization was studied for independent tasks without preemption [12] and for Bag-of-Tasks applications [13].

Using the stretch as a strategy for fairness, we propose *OStrich* — an on-line scheduling algorithm that bounds the

maximum stretch of each campaign by a function of the total workload of this campaign and the previous campaign, the number of active users and the maximum job length (p_{max}). Jobs are scheduled according to a priority list; priorities are determined by campaign's *virtual execution time*. We define this virtual execution time as the time the campaign would take to complete in a divisible load task model and using a fair-share scheduling strategy that assigns an equal share of processors to each user. We validate *OStrich* experimentally by conducting simulations on a workload trace. Compared to a standard scheduler (Maui with FCFS scheduling), *OStrich* results in lower stretches and more consistent results for every 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 III we present a formal model of Campaign Scheduling. This model is an extension of what was defined in [7]. Section IV 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 V. Our solution is assessed through simulations that uses data extracted from a workload execution log in a real cluster. This is presented in Section VI. In Section VII, we show how our solution can also be used for the general scheduling problem with multiple users. Finally, we present our conclusions and future work in Section VIII.

II. State-of-art

The main works related to our research address the problem of scheduling jobs from multiple users who compete for the resources and the use of fair policies to favoring jobs on parallel systems.

Fairness is an important issue on the design of scheduling policies and it has gained growing attention in the last decade. But it is still a fuzzy concept that has been handled in many different ways, varying according to the target problems. In [8], the authors discuss the Maui scheduler, specifically the FCFS-backfill, job prioritization and fairness mechanisms. Historical fair-share usage information is provided, which allows favoring jobs based on historical usage of resources associated with their credentials. Despite the great flexibility offered by the scheduler, which supports fine-tuning of several parameters directly accounted in the fairness configuration, these data are mainly job-attribute based dimensions while our solution adopts a user centric approach.

In [10] and [9], metrics are proposed for expressing the degree of unfairness in various systems. Both works evaluates 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.

The Multi-Users Scheduling Problem (MUSP) was first studied on a single processor with two users by Agnetis et al. [14] and on multiple processors by Saule and Trystram [15].

Agnetis et al. [14] provided a $< \overline{1}, \overline{1} >$-approximation for the problem of two users competing to perform their respective jobs as soon as possible on a common processing resource. They analyzed several scenarios depending on the objective function adopted by each agent and on the structure of the processing system. They are interested in Constrained Optimization Problems where one objective is fixed as a constraint while the second objective is optimized.

The authors show that when both users are interested in the makespan, the problem can be solved in polynomial time. If they are both interested in the sum of completion times, the problem becomes binary NP-hard and they provide a pseudo-polynomial dynamic program to solve it. With mixed objectives, if one user is interested in the weighted sum of completion times, the problem is binary NP-hard. Other cases are polynomial.

Saule and Trystram [15] 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 notation is a natural extension of the classic notation of approximation ratios where the u-th number of the vector corresponds to the approximation ratio of the objective

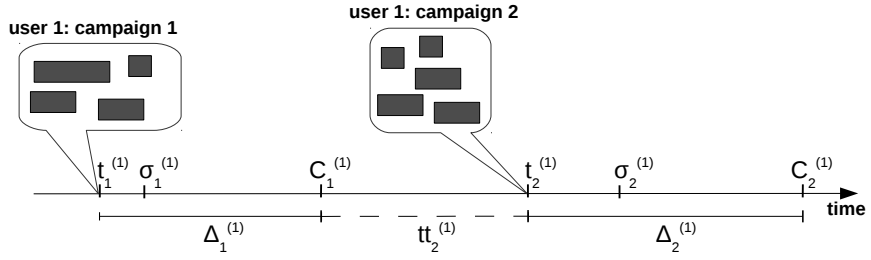

Fig. 1. Campaign Scheduling notations (two submissions from 1 user)

of the u-th user.

What is interesting regarding our work is that the problem of MUSP with all the users interested in C_{max} is equivalent to an instance of Campaign Scheduling where all users submit only one campaign at the same time, in the beginning of the schedule (i.e. ready jobs). Obviously, the performance guarantees obtained from $MUSP(k : C_{max})$ can be applied in the same way for this particular instance of Campaign Scheduling.

The Campaign Scheduling model was introduced in our earlier work [7] where we considered a partially on-line model with continuous streams of campaigns: for each user, immediately after a campaign was completed, the next one was supposed to be ready. This paper considers an on-line version of this problem. Additionally, we allow a "think-time" between successive campaigns, thus the number of active users (those who have jobs to compute) varies dynamically during the system's life-time. This work is related to the DEQ algorithm [16] which considers virtual schedule and preemptive jobs. However our main contribution lies in the introduction of a real schedule. It enables us to be more efficient than DEQ (e.g. in DEQ, if there are 2 users, it is Pareto-efficient to finish all jobs of the first user before starting the second one, instead of executing them in parallel).

III. Model and Problem Definition

The model consists of k users (indexed by u) sharing the computational resources on a parallel platform composed of m identical processors (indexed by q). The resources are managed by a centralized scheduler. Figure 1 illustrates some of the used notation.

A user workflow is composed of $\gamma^{(u)}$ successive campaigns. Each user campaign $i \in [1, \gamma^{(u)}]$ is submitted at a time denoted by $t_i^{(u)}$ and is composed of a set of independent and non-preemptive sequential jobs. The submission time $t_i^{(u)}$ of a campaign is not fixed *a priori* as it depends on the termination of the previous campaign (but it is a part of the instance of a problem). A campaign is defined as the set of jobs $J_i^{(u)}$ released in one submission. $n_i^{(u)}$ denotes the number of jobs of u 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 this job in the campaign. $p_{i,j}^{(u)}$ denotes the job's length; the length is known when the job is submitted (the clairvoyant model). The job start time is denoted by $\sigma_{i,j}^{(u)}$; its completion time is denoted by $C_{i,j}^{(u)}$.

For a campaign $J_i^{(u)}$ its start time $\sigma_i^{(u)}$ is the time the first job starts, $\sigma_i^{(u)} \triangleq \min_j \sigma_{i,j}^{(u)}$; 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 total workload within campaign i is: $W_i^{(u)} = \sum_j p_{i,j}^{(u)}$.

[1]There is a particular case where the jobs can be infinitely divided into smaller pieces (i.e. fine grained). For instance, this is the case of BOINC divisible loads [1]. 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 campaign's flow time $\Delta_i^{(u)}$ is equal to the time jobs of the campaign stay in the system, $\Delta_i^{(u)} \triangleq C_i^{(u)} - t_i^{(u)}$. The campaign's stretch $D_i^{(u)}$ is equal to the campaign flow time divided by its surface $W_i^{(u)}$, $D_i^{(u)} = \frac{\Delta_i^{(u)}}{W_i^{(u)}}$.

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 $tt_{i+1}^{(u)} = t_{i+1}^{(u)} - C_i^{(u)}$.

The considered model is on-line, meaning that the campaign $J_i^{(u)}$'s submission time and its workload is known only after the campaign is submitted (i.e. at time $t_i^{(u)}$). However, as soon as a campaign is submitted, its jobs lengths are known (clairvoyant model).

The scheduling goal 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 indivual users. Moreover, considering stretch of each campaign (rather than the overall stretch) guarantees that the system is responsive: 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 ($\gamma = 1$), it is equivalent to the minimization of the makespan on identical parallel processors ($P||C_{\max}$).

IV. Algorithm

A. Principle

We propose in this section a new scheduling algorithm called *OStrich* (from "per-User guaranteed Stretch"). 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 executes the following two steps as soon as a processor becomes available:

- Select from the list the next job of the campaign with the highest priority.
- Schedule this job on the available processor.

Any scheduling policy can be used to determine the execution order of jobs within a single campaign; for instance LPT [17] (or, more appropriately, MLPT [18]) or Shortest Processing Time (SPT) [19].

The virtual fair-sharing schedule is maintained by dividing the resources between the active users at each given moment. The resources 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, which

is simply 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 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 resources, added by its virtual start time. More formally:

$$\tilde{C}_i^{(u)}(t) = \tilde{W}_i^{(u)}/(m/\tilde{k}(t)) + \tilde{\sigma}_i^{(u)} = \tilde{k}(t)\tilde{W}_i^{(u)}/m + \tilde{\sigma}_i^{(u)}$$

Note that the share of a user is defined as the number of machines m divided by the number of active users at moment t, denoted by $\tilde{k}(t)$. This is the number of users with unfinished campaigns at time t, according to the virtual schedule. Formally, $k(t)$ 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 ($\tilde{\sigma}_i^{(u)} \geq t_i^{(u)}$), but also not sooner than the virtual completion time of the previous campaign (as the campaign in the real schedule can be completed earlier than in the virtual schedule):

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

The second condition guarantees that at each time moment, at most one campaign of each user is executing in the virtual schedule; thus the number of allocated processors depends on the number of active users, and not the workload.

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 the virtual completion times of other active campaigns.

Suppose that a new campaign i is submitted at time $t_i^{(u)}$. The value of $\tilde{k}(t_i^{(u)})$ is updated to reflect the number of active users at $t_i^{(u)}$. Thus, every time $\tilde{k}$ changes, the virtual completion time of the campaigns must follow this change as well. Besides, at each event e occuring at time t_e, the 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 ocurrence t_{e-1} and multiplying it by campaign's share of resources on that time interval. Considering all the events passed after the campaign's submission, the workload is:

$$\tilde{W}_i^{(u)} = \sum p_i^{(u)} - \sum_e (\tilde{k}(t_{e-1}).(t_e - t_{e-1})/m),$$

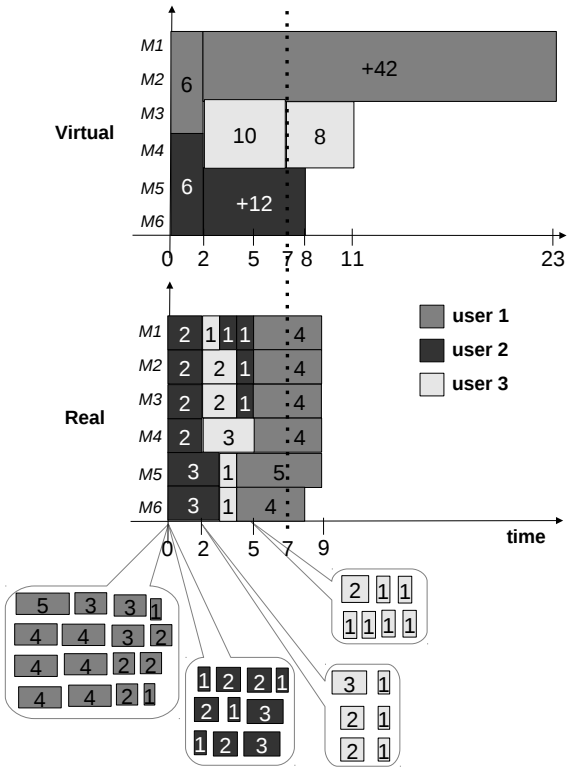

Fig. 2. Virtual and real schedule generated by the *OStrich* algorithm with 3 users

B. Example

The figure 2 shows an example with the real and the virtual schedule generated by the *OStrich* algorithm in a system with 6 identical machines 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 machines equally between them and 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 his virtual completion time is the smallest (thus, he has execution priority).

The situation changes at $t = 2$ when user 3 submits her 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 machines 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 machines 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 first campaign is $\tilde{C}_1^{(3)} = 7$. As $\sigma_2^{(3)} < \tilde{C}_1^{(3)}$, $\tilde{\sigma}_2^{(3)} = \tilde{C}_1^{(3)}$ and her virtual completion time is set to $\tilde{C}_2^{(3)} = 11$. The user 3 retakes the priority but note that even if her first campaign was finished in the real schedule at $t = 5$, her next campaign must wait until $\tilde{\sigma}_2^{(3)} = 7$ to be taken into account. This mechanism keeps the real schedule in accordance with the virtual (and fair) schedule: a user is not able to take a greater share of the resources than what is assigned in the virtual schedule.

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

C. Implementation

OStrich 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 resources are available.

The algorithm is implemented by two loop procedures that run in parallel. These procedures share a event list and a priority queue with the campaigns that are ready to execute. The first procedure receives the submissions, creates the corresponding events and puts them into the list. The second procedure consumes the events from the list and updates the schedule. Bellow, we write a simplified description of these procedures.

```
1: events = new_ordered_list()
2: campaigns = new_queue()
3: k = new_set()
4: while true do
5:    wait(next)
6:    sub = new_event(next, SUBMISSION)
7:    insert(events, sub)
8:    enqueue(campaigns, next)
9: end while
```

The first 3 lines are the global variables shared by both procedures: an ordered list of events, a priority queue of campaigns and a set to hold the active users. The line 5 blocks the procedure, waiting for the next campaign submission. When this happens, a new event of type submission is created (line 6) and inserted into the list of events (line 7). The new campaigns is enqueued and is ready for execution (line 8).

In this procedure, the line 2 removes the next event from the list. This list is ordered by the event times. The number of active users is updated according to the new event (line 3). If the event is a submission, the corresponding virtual completion time event is created (lines 4, 5 and 6). This can trigger the update of the existing virtual completion times (line 8. Finally, the schedule is updated according to the priorities defined by the virtual completion times (lines 9 and 10).

```
1: while true do
2:    e = remove(events)
3:    update_k(e, k)
4:    if type(e) = SUBMISSION then
5:       v = new_event(e.next, VCOMPLETION)
6:       insert(events, v)
7:    end if
8:    update_vcompletions(events)
9:    priorities = vcompletions(events)
10:   update_schedule(machines, campaigns, priorities)
11: end while
```

V. Theoretical Analysis

The goal of this section is to study the worst case of the maximal stretch within a campaign. The idea of the proof is to bound the completion time of the last job of a campaign using a global surface argument compared to the virtual schedule.

In this section we will denote by V the virtual schedule; and by R the real schedule. To simplify the formulation of proofs, we will also 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.

A. Worst-Case Bound

As V can assign a job an arbitrary fraction of processors (from ϵ to m), a schedule in V is a series of compact rectangles (separated by idle times when there are no jobs in the system). R must execute each job on a single processor; thus R is not as compact as V. 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 a surface 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 surface in R (counted from the earliest to the latest job start time) exceeds the total idle surface 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 surface in R starts to exceed $mp_{\max}$. Thus, at least one processor is free at time t and there is no ready job to be executed. As V has no idle times, at time t the surface executed by V exceeds the surface executed by R by more than $mp_{\max}$. Thus, the surface exceeding $mp_{\max}$ is ready to be executed at R. As a single job has a surface of at most $p_{\max}$, a surface 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.

If there is idle time in V, each idle period can be modelled as a set of special 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 $\{J_I\}$ contribute the same amount $\sum p_I$ of idle surface to V and to R). If R executes $\{J_I\}$ partially (i.e. as $\{J'_I\}$, with $0 \leq p'_I \leq p_I$) the contribution of these jobs to the idle surface 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 his/her campaign $J_j^{(v)}$ which has a lower surface than what remains of $J_i^{(u)}$, and thus gains 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 ($\nexists J' : J \in J_j^{(v)} \wedge \sigma_{Jk} < \sigma_{J'} < \sigma_{Jl}$).

The following lemma bounds the completion time of the last piece of the campaign. After a campaign completes in the virtual schedule, it cannot be delayed by any other newly-submitted campaign; thus it has the highest priority and is executed in one piece (i.e., the last piece). The lemma upper-bounds the virtual surface having higher priority by the surface of the campaign, as in the worst case k users submit campaigns of equal surface, 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:

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

where $t_i^{(u)} + k\frac{W_{i-1}^{(u)}}{m}$ expresses the time the campaign waits until the virtual completion time of the previous campaign $J_{i-1}^{(u)}$ of the same user; $(k-1)\frac{W_i^{(u)}}{m}$ bounds the time needed to execute other users' campaigns that can have higher priority; $\frac{W_i^{(u)}}{m} + p_{\max}^{(u)}$ 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.

Proof: A campaign starts in the virtual schedule $\tilde{\sigma}_i^{(u)}$ no sooner than the virtual completion time of the previous campaign $\tilde{\sigma}_i^{(u)} = \max(t_i^{(u)}, \tilde{C}_{i-1}^{(u)})$. As $\tilde{C}_{i-1}^{(u)} = \tilde{\sigma}_{i-1}^{(u)} + k\frac{W_{i-1}^{(u)}}{m}$; and $t_i^{(u)} \geq \sigma_{i-1}^{(u)}$ (the next campaign cannot be released before the previous one starts), $\tilde{\sigma}_i^{(u)} \leq t_i^{(u)} + k\frac{W_{i-1}^{(u)}}{m}$.

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

We denote by S the surface of jobs executed in R after the time moment $\tilde{\sigma}_i^{(u)}$ (when the campaign starts in the virtual schedule), and until $\tilde{\sigma}_{i,q}^{(u)}$. We claim that $S \leq mp_{\max} + (k-1)W_i^u + W_i'^u$ where $W_i'^u$ is the surface of jobs from campaign $J_i^{(u)}$ executed until $\sigma_{i,q}^u$. The Figure 3 facilitates the visualization of these notations, including the surface S (shaded area).

To prove the claim, we analyze job J executed in R in the period $[\tilde{\sigma}_i^u, \sigma_{i,q}^u)$. First, J is not executed in V after $\sigma_{i,q}^u$. If J is in V after $\sigma_{i,q}^u$, 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 $\tilde{\sigma}_i^u$, it means that R is "late" in terms of executed surface: but the total surface of such "late" jobs it at most $mp_{\max}$ (from Lemma 1).

Thus, if J has a corresponding surface in the virtual schedule exectued in the period $[\tilde{\sigma}_i^u, \sigma_{i,q}^u)$, the surface S of the jobs started in the real schedule in this period is equal to the surface of the virtual schedule between $[\tilde{\sigma}_i^u, \sigma_{i,q}^u)$ (plus the lateness $mp_{\max}$). Recall that from time $\sigma_{i,q}^u$ 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, $\sigma_{i,q}^u$ 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, $\sigma_{i,q}^u \leq p_{\max} + \tilde{\sigma}_i^{(u)} + k\frac{W_i^{(u)}}{m}$.

Starting from $\sigma_{i,q}^u$, the remaining jobs of $J_i^{(u)}$ start and complete. $J_i^{(u)}$ can claim all processors at the latest $p_{\max}$ after $\sigma_{i,q}^u$. Then, by using classic lower bounds, it takes $W_i^{(u)}/m + p_{\max}^{(u)}$ to complete the campaign. ■

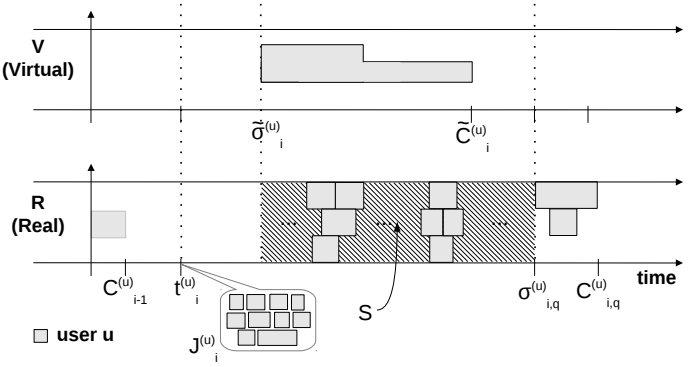

Fig. 3. Analysis of *OStrich*: bound for the campaign stretch

The following theorem states that in *OStrich* the stretch of a campaign depends only on the number of active users and the relative surface of two consecutive campaigns. In contrast, the stretch does not dependent on the current total load of the system; thus heavily-loaded users do not influence the less-loaded ones.

Theorem 1: The stretch of a campaign is proportional to the number of active users k and the relative surface of consecutive campaigns. $D_i^{(u)} \in O(k(1 + \frac{W_{i-1}^{(u)}}{W_i^{(u)}}))$.

Proof: The result follows directly from Lemma 2. For campaign $J_i^{(u)}$, the stretch $D_i^{(u)}$ is equal to the campaign flow time $\Delta_i^{(u)} = C_i^{(u)} - t_i^{(u)}$ divided by its surface $W_i^{(u)}$, $D_i^{(u)} = \frac{C_i^{(u)} - t_i^{(u)}}{W_i^{(u)}}$. By Lemma 2,

$$C_{i,q} \leq t_i^{(u)} + k\frac{W_{i-1}^{(u)}}{m} + 2p_{\max} + (k-1)\frac{W_i^{(u)}}{m} + \frac{W_i^{(u)}}{m} + p_{\max}^{(u)}.$$

Thus, $D_i^{(u)} \leq \frac{k}{m}(1 + \frac{W_{i-1}^{(u)}}{W_i^{(u)}}) + 3\frac{p_{\max}}{W_i^{(u)}}$. For a given supercomputer, m is constant; similarly, the maximum size of a job $p_{\max}$ can be treated as constant, as typically it is set as a limit by system administrators. Thus, $D_i^{(u)} \in O(k(1 + \frac{W_{i-1}^{(u)}}{W_i^{(u)}}))$. ■

B. Tightness

This section analyses the tightness of the $O(k(1 + \frac{W_{i-1}^{(u)}}{W_i^{(u)}}))$ bound proposed in Theorem 2. We start with a negative result that says that in heavily-loaded systems, campaigns have to be executed sequentially, thus at least one of them will have a stretch in $O(k)$.

Proposition 1: No scheduling algorithm can achieve better stretch than $O(k)$.

Proof: Consider an instance with k users, each submitting at $t = 0$ a campaign with m jobs of unit size. The campaigns have to be executed sequentially, so there is at least one user whose campaign completes at time k. ■

The following proposition shows an instance in which the bound $O(k(1 + \frac{W_{i-1}^{(u)}}{W_i^{(u)}}))$ (Theorem 2) is asymptotically tight. The instance is composed of a series of long campaigns followed by a series of short campaigns. A user who had his/her long campaign executed at the beginning, must wait with the short campaign not only the time needed to complete all other long campaigns, but also possibly other short campaigns.

Proposition 2: The bound $D_i^{(u)} \in O(k(1 + \frac{W_{i-1}^{(u)}}{W_i^{(u)}}))$ is asymptotically tight.

Proof: Consider an instance with m processors and k users having two campaigns each. At $t = 0$, each user submits a campaign with m jobs of size $p_{\max}$. The second campaign of a user is submitted immediately after the completion of the first campaign, with m jobs of size $p = 1$. As campaigns $\{J_1^{(u)}\}$ have the same priority; and campaigns $\{J_2^{(u)}\}$ have the same priority, *OStrich* can produce any schedule that executes first all the campaigns $\{J_1^{(u)}\}$, then all campaigns $\{J_2^{(u)}\}$. Thus, an admissible schedule is $(J_1^{(1)}, J_1^{(2)}, \ldots, J_1^{(k)}, J_2^{(k)}, J_2^{(k-1)}, \ldots, J_2^{(1)})$. The completion time of $J_2^{(1)}$ is $C_2^{(1)} = kp_{\max} + k$, thus the stretch $D_2^{(1)} = (k-1)p_{\max} + k = 1 + (k-1)(1 + \frac{W_1^{(1)}}{W_2^{(1)}})$. ■

VI. Simulations

In this section, we analyze the performance of *OStrich* on the trace from the LPC-EGEE cluster [20]. We first describe how to detect campaigns in standard workloads (that do not provide information about dependencies between jobs). Then, we describe the settings of the simulations. Finally, we compare the performance of *OStrich* to the standard scheduler (Maui with FCFS and backfilling) used on the cluster.

A. Detecting campaigns in real workloads

In recent works about workload modeling [3]–[5], Feitelson *et al.* consider that the actions of a user (i.e. his/her job submission behavior and frequency) is influenced by the actions of the other users and by the scheduler decisions. This *user feedback* has an impact on the observed workload. This is justified by the fact that the user is aware of the current system behavior and his/her future actions directly depend on the system's actual performance. To incorporate this feedback into the model, Feitelson proposes the postulate of dependencies between jobs, which relies upon the idea of what is called the *think-time*, i.e. the delay between the ending of a job and the submission of a new one. This delay represents the time taken by the user to analyze the result of an executed job and to prepare the submission of the next. This is the same concept as the think-time we use in the Campaign Scheduling model.

In [4], the think-time is used to detect user sessions and batches in a workload trace. A user session is a period of continuous work where a same user submits jobs. The continuity of a session is defined by a time threshold. Interruption of the user activities above this threshold configures a session break. Within a session, jobs that are overlapping will be grouped in what is called a *batch*. Note that, the notion of *campaign* is similar to the notion of batch: a set of jobs, submitted asynchronously by a user and that will run independently from the others.

[4] proposes different methods to detect users' batches in a workload log. The *LAST* algorithm was originally proposed in [6]. In this approach, two jobs that are submitted one after the other belong to the same batch if the last one was submitted before the end of the first one. The *ARRIVAL* algorithm is based on the inter-arrival times of the jobs, i.e. it does not take into account their runtimes. The *MAX* algorithm states that a job belongs to a batch if it overlaps at least one of the jobs of the batch. In this work, the *MAX* algorithm is adopted since it produces batches that correspond to the concept of campaign described in the model.

B. Simulations Conditions

To run the simulations, we choose the LPC-EGEE[2] trace from the Parallel Workload Archive [20] (cleaned version). This trace comes from a cluster that is part of the EGEE[3] project. This cluster has the particularity of being composed of multiple *Bag-of-Tasks* (i.e. serial jobs), which fits well

[2]http://www.cs.huji.ac.il/labs/parallel/workload/l_lpc/index.html
[3]http://public.eu-egee.org/intro/

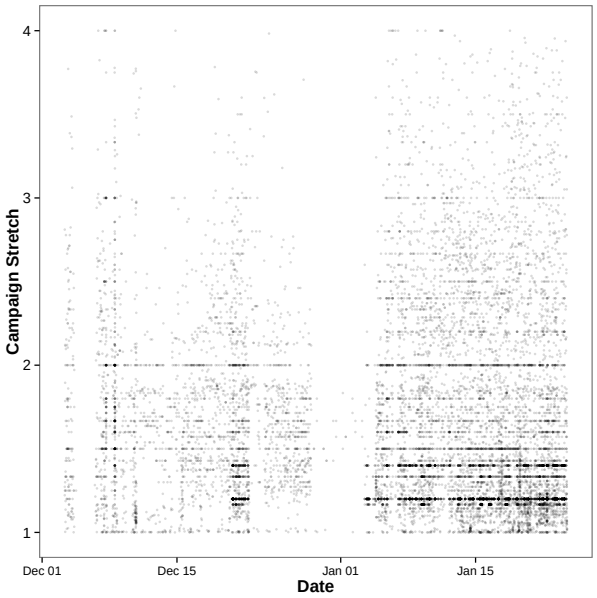

Fig. 4. Campaign stretches along time (original log: Maui, FCFS)

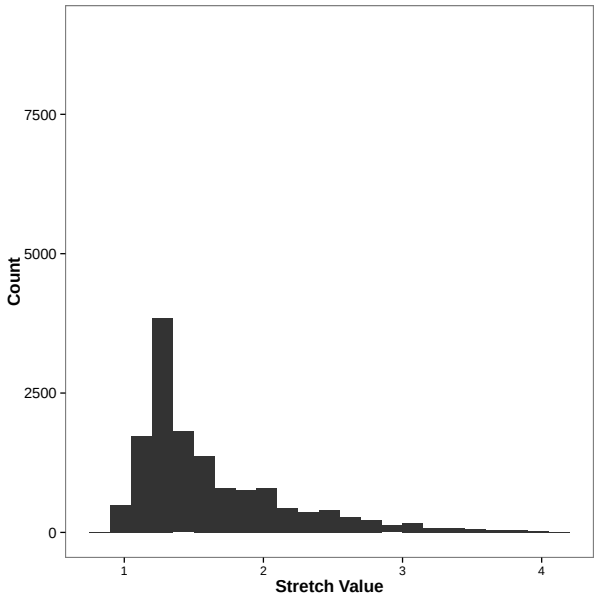

Fig. 5. Campaign stretch values distribution (original log: Maui, FCFS)

the concept of campaign. The system is composed of 140 identical machines. The scheduler used to produce the log was Maui [8] with its default EASY configuration, that is, FCFS with backfilling. However, since all the jobs are sequential, there is actually no backfilling: whenever a CPU is available, this is enough for executing the job at the head of the wait-queue, and thus no other job is allowed to skip it.

The trace is long (10 months); it contains several cuts (due to electrical problems, management system reconfiguration, cooling failures...) and machines' failures. We selected the longest period without failures as the input of our algorithm (from December 6 2004 to January 24 2005). The workload covered in this period is issued from 33 users. The performance of *OStrich* is compared to the one extracted from the log.

C. Results

The results are plotted in figures 7, 8, 4, 5, 6 and 9.

Figures 7 and 8 show system utilization, or the percentage of allocated resources over time. *OStrich* achieves more homogeneous utilization while the original log presents a more erratic behavior. With *OStrich* the system achieves its maximum utilization several times. Regarding the periods where the system load is above 10%, *OStrich* achieves system utilization at least 20% greater than the one obtained from the original log. Not surprisingly, periods of user inactivity are also more frequent in *OStrich*. From our point of view, these periods of machine inactivity could be potentially exploited in at least two ways: energy saving and execution of best effort jobs.

The figure 4 shows the stretch values obtained by each campaign at their completion times in the original log. As we can observe, the stretches are widely spread. To better

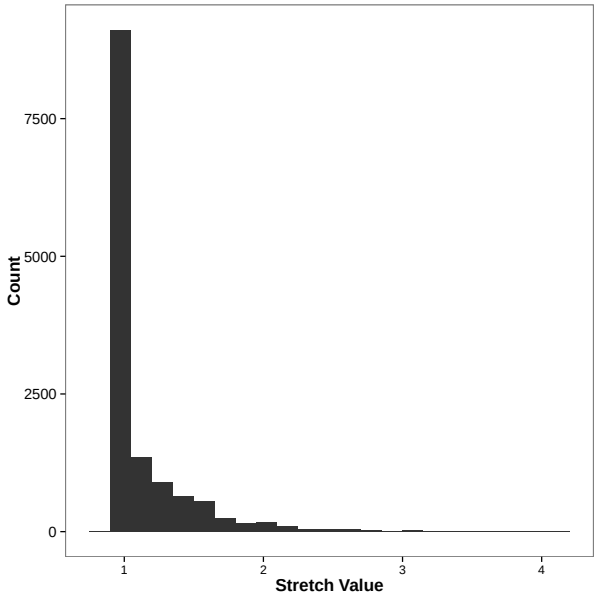

Fig. 6. Campaign stretch values distribution (*OStrich*)

understand the distribution of these stretches we plot figure 5 for the original log then figure 6 for the workload replayed in the simulation with *OStrich*. In these figures, some outliers, with very high values, have been removed to clarify the graphs.[4] To be as close as possible to the original log in our re-scheduling we chose to keep these campaigns in the replay but

[4]These high values come from the fact that there is no restriction for the maximum length of a job (i.e. p_{max}). In consequence, few users monopolize the platform at some periods in the trace, filling the machine with very long jobs and hence causing big stretches for campaigns that are composed of short jobs.

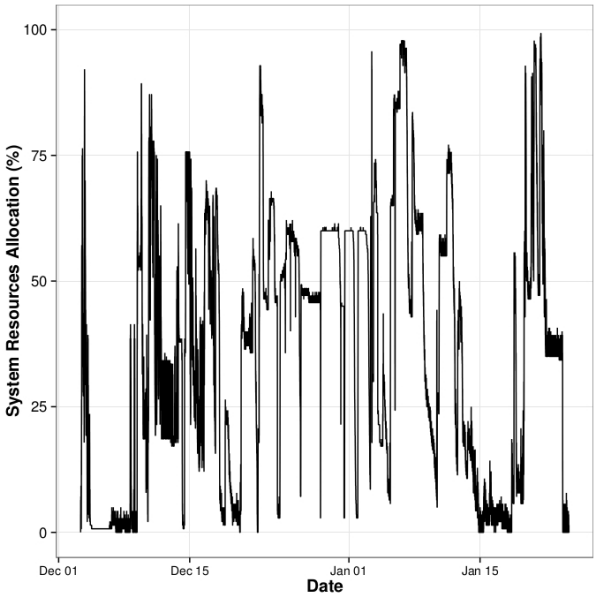

Fig. 7. System resources utilization (original log: Maui, FCFS)

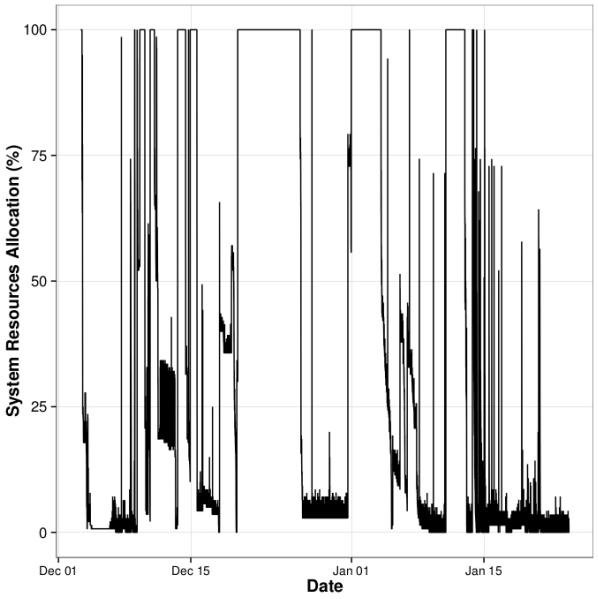

Fig. 8. System resources utilization (*OStrich*)

remove from the figures the few very large stretches (stretches of more than 1000) due to this particular behavior. In both figures, the amount of campaigns removed is less than 4%. The results show stretch values ranging from 1 to 4 but with *OStrich* they are much more concentrated near 1, while in the original log the values are more spread with a higher density between 1.3 and 1.5. For the original log, the mean stretch is 1.61, for *OStrich* it is 1.12 on the dataset used to plot figures 5 and 6 (without the outliers). So, for a great majority of campaigns (64%), the stretch obtained by *OStrich* was exactly of 1; 90% of the campaigns have a stretch of less than 1.4,

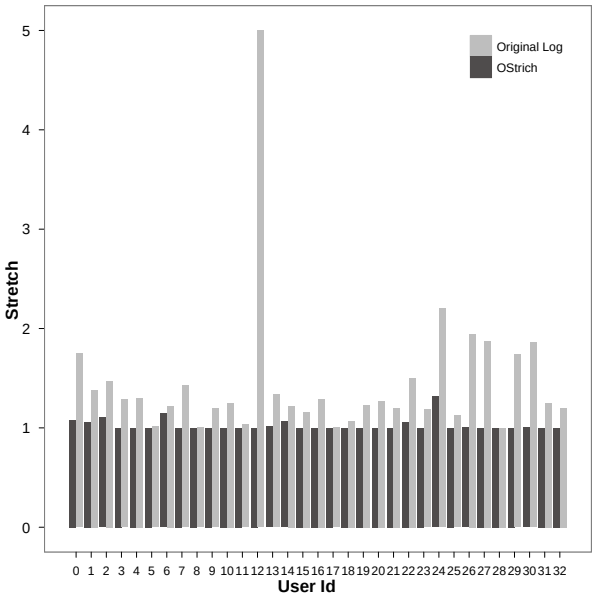

Fig. 9. Median of campaign stretch: comparison between *OStrich* and Maui (FCFS)

and 99% of the campaigns have a stretch of less than 2.15.

In order to verify whether all the users have similar performance, in Figure 9 we extract the median stretch value obtained by each user on the complete dataset (including the outliers). The median was used instead of the average to avoid outlier's interference. In addition to emphasizing the superiority of *OStrich* regarding the stretch values, this graph shows a more fair distribution among the 33 different users active during the period of the selected trace, between the trace replayed in *OStrich* and the original log.

VII. Extension: General (Non-Campaign) On-line Scheduling Problem with OStrich

Up to this point, we considered the problem of scheduling a specific type of workload in which each user submitted campaigns of jobs: all the jobs from a campaign $J_i^{(u)}$ are ready at the same time $t_i^{(u)}$; and the subsequent campaign $J_{i+1}^{(u)}$ is not ready until the previous campaign completes ($t_{i+1}^{(u)} \geq C_i^{(u)}$). For this problem, *OStrich* guarantees stretch proportional to the number of active users. In this section, we show how *OStrich* can be used for the general on-line scheduling problem with multiple users.

In this section, we will denote as $\iota_l^{(u)}$ the l-th job of user u (we do not use the previous notation of $J_{i,j}^{(u)}$ as it assigns a job to a campaign). $r_l^{(u)}$ denotes job's release date. Similarly to the campaign scheduling problem, we consider an on-line, clairvoyant model: job $\iota_l^{(u)}$ is not known until it is released; job's length $p_l^{(u)}$ is known at the time the job is released.

An instance of the general on-line scheduling problem can be transformed to the campaign scheduling problem. The transformation is analogous to building batches; yet, unlike the classic approach [21], we don't mix jobs of different

users in one batch. In the following, we will use the term "batch" to denote a set of jobs; these batches will be treated as campaigns by the scheduling algorithm. For user u, the first batch $J_1^{(u)}$ is constructed by one or more jobs released at the same time $r_1^{(u)}$, i.e., $J_1^{(u)} = \{\iota_l^{(u)} : r_l^{(u)} = r_1^{(u)}\}$ and $t_1^{(u)} = r_1^{(u)}$. Then, if a previous batch has not yet completed in the virtual schedule, the following jobs are joined into a batch: $J_i^{(u)} = \{\iota_l^{(u)} : t_{i-1}^{(u)} \leq r_l^{(u)} \leq \tilde{C}_{i-1}^{(u)}\}$; the batch is released immediately after the previous batch completes, $t_i^{(u)} = \tilde{C}_{i-1}^{(u)}$. If a user does not have an executing batch in the virtual schedule, similarly to the first batch, the batch gathers all the jobs submitted at the same moment and is released immediately.

The mapping allows for a similar nature of a worst-case bound as for the campaign scheduling problem. However, instead of bounding the stretch of a campaign, we will bound the job's $\iota_l^{(u)}$ flow time $\delta_l^{(u)}$. As previously, the bound depends only on the number of active users k and the workload of the user—and not the total load of the system.

Theorem 2: The maximum flow time of a job is proportional to the number of active users k and the total surface of two consecutive batches, $F_l^{(u)} \leq 3p_{\max} + \frac{k}{m}(W_{i-1}^{(u)} + W_i^{(u)})$.

Proof: Job $\iota_l^{(u)}$ is assigned to a batch $J_i^{(u)}$ which is released, at the latest, when the previous batch completes, $t_i^{(u)} = \tilde{C}_{i-1}^{(u)}$. As the job was not assigned to the previous batch, its release date $r_l^{(u)} > t_{i-1}^{(u)} = \tilde{\sigma}_{i-1}^{(u)}$ (the last equation is true as there is at most one active batch of u, thus it can be started immediately). Thus, a job waits until it is started in the virtual schedule from $\tilde{\sigma}_{i-1}^{(u)}$ till $\tilde{C}_{i-1}^{(u)}$, i.e., $W_{i-1}^{(u)}k/m$.

Batches are treated by *OStrich* like campaigns, thus, by Lemma 2, the last job of batch $J_i^{(u)}$ completes in the real schedule by $C_{i,q} \leq \tilde{\sigma}_i^{(u)} + 2p_{\max} + (k-1)\frac{W_i^{(u)}}{m} + \frac{W_i^{(u)}}{m} + p_{\max}^{(u)}$. Thus, $C_{i,q} \leq r_l^{(u)} + k\frac{W_{i-1}^{(u)}}{m} + k\frac{W_i^{(u)}}{m} + 3p_{\max}$. ∎

VIII. Concluding remarks

We have presented in this work a new scheduling algorithm for the on-line scheduling problem with multiple submissions issued by many users. *OStrich* algorithm has been designed to handle the problem of fairness between users by defining execution 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. Furthermore, we demonstrated that the same algorithm can be used for the general on-line scheduling problem; for each user, the maimum flow time of a job again depends only on the number of active users and the user's workload, and not the total load of the system.

The performance of our algorithm is assessed by running simulations based on actual workloads. The results show that *OStrich* achieves lower stretches than the FCFS based Maui scheduler for sequential jobs; moreover distribution of stretches among users is also more equal. Thus, *OStrich* delivers a good compromise between fairness and user performance.

References

[1] 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.

[2] 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. Washington, DC, USA: IEEE Computer Society, 2011, pp. 144–153. [Online]. Available: http://dx.doi.org/10.1109/CCGrid.2011.34

[3] D. Feitelson, *Workload Modeling for Computer Systems Performance Evaluation*, 2005. [Online]. Available: http://www.cs.huji.ac.il/~feit/wlmod/wlmod.pdf

[4] N. Zakay and D. G. Feitelson, "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. [Online]. Available: http://www.cs.huji.ac.il/~feit/parsched/jsspp12/p12-zakay.pdf

[5] D. Mehrzadi and D. G. Feitelson, "On extracting session data from activity logs," in *Proc. of the 5th Intl. Syst. & Storage Conference*, The Hebrew University, Israel, 2012. [Online]. Available: http://www.cs.huji.ac.il/~feit/papers/Ses12SYSTOR.pdf

[6] E. Shmueli and D. Feitelson, "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, pp. 167 – 178.

[7] V. Pinheiro, K. Rzadca, and D. Trystram, "Campaign scheduling," in *IEEE International Conference on High Performance Computing (HiPC), Proceedings*, 2012, accepted for publication.

[8] D. B. Jackson, Q. Snell, and M. J. Clement, "Core algorithms of the maui scheduler," in *Revised Papers from the 7th International Workshop on Job Scheduling Strategies for Parallel Processing*, ser. JSSPP '01. London, UK, UK: Springer-Verlag, 2001, pp. 87–102. [Online]. Available: http://dl.acm.org/citation.cfm?id=646382.689682

[9] D. Raz, H. Levy, and B. Avi-Itzhak, "A resource-allocation queueing fairness measure," *SIGMETRICS Perform. Eval. Rev.*, vol. 32, no. 1, pp. 130–141, Jun. 2004. [Online]. Available: http://doi.acm.org/10.1145/1012888.1005704

[10] G. Sabin, G. Kochhar, and P. Sadayappan, "Job fairness in non-preemptive job scheduling," in *Proceedings of the 2004 International Conference on Parallel Processing*, ser. ICPP '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 186–194. [Online]. Available: http://dx.doi.org/10.1109/ICPP.2004.41

[11] S.-H. Chiang and M. K. Vernon, "Production job scheduling for parallel shared memory systems," in *Proceedings of the 15th International Parallel & Distributed Processing Symposium*, ser. IPDPS '01. Washington, DC, USA: IEEE Computer Society, 2001, pp. 47–. [Online]. Available: http://dl.acm.org/citation.cfm?id=645609.662318

[12] M. A. Bender, S. Muthukrishnan, and R. Rajaraman, "Improved algorithms for stretch scheduling," in *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, ser. SODA '02. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2002, pp. 762–771. [Online]. Available: http://dl.acm.org/citation.cfm?id=545381.545482

[13] A. Legrand, A. Su, and F. Vivien, "Minimizing the stretch when scheduling flows of biological requests," in *Proceedings of the eighteenth annual ACM symposium on Parallelism in algorithms and architectures*, ser. SPAA '06. New York, NY, USA: ACM, 2006, pp. 103–112. [Online]. Available: http://doi.acm.org/10.1145/1148109.1148124

[14] 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.

[15] 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. [Online]. Available: http://bmi.osu.edu/~esaule/public-website/paper/ipdps09-ST.pdf

[16] X. Deng, N. Gu, T. Brecht, and K. Lu, "Preemptive scheduling of parallel jobs on multiprocessors," in *Proceedings of the seventh annual ACM-SIAM symposium on Discrete algorithms*, ser. SODA '96. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1996, pp. 159–167. [Online]. Available: http://dl.acm.org/citation.cfm?id=313852.313912

[17] R. L. Graham, "Bounds on multiprocessing timing anomalies," *SIAM JOURNAL ON APPLIED MATHEMATICS*, vol. 17, no. 2, pp. 416–429, 1969.

[18] 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

[19] J. Bruno, J. E. G. Coffman, and R. Sethi, "Scheduling independent tasks to reduce mean finishing time," *Commun. ACM*, vol. 17, no. 7, pp. 382–387, Jul. 1974. [Online]. Available: http://doi.acm.org/10.1145/361011.361064

[20] D. G. Feitelson, "Parallel workload archive," http://www.cs.huji.ac.il/labs/parallel/workload/, last accessed on 28 Sep, 2012.

[21] D. Shmoys, J. Wein, and D. Williamson, "Scheduling parallel machines on-line," *SIAM Journal on Computing*, vol. 24, no. 6, pp. 1313–1331, 1995.


L I G