Cloud Scheduling with Setup Cost

Yossi Azar [*]
Blavatnik School of Computer Science, Tel-Aviv University.
azar@tau.ac.il

Naama Ben-Aroya [†]
Blavatnik School of Computer Science, Tel-Aviv University.

Nikhil R. Devanur
Microsoft Research, Redmond, WA.
nikdev@microsoft.com

Navendu Jain
Microsoft Research, Redmond, WA.
navendu@microsoft.com

ABSTRACT

In this paper, we investigate the problem of online task scheduling of jobs such as MapReduce jobs, Monte Carlo simulations and generating search index from web documents, on cloud computing infrastructures. We consider the virtualized cloud computing setup comprising machines that host multiple identical virtual machines (VMs) under pay-as-you-go charging, and that booting a VM requires a constant setup time. The cost of job computation depends on the number of VMs activated, and the VMs can be activated and shutdown on demand. We propose a new bi-objective algorithm to minimize the maximum task delay, and the total cost of the computation. We study both the clairvoyant case, where the duration of each task is known upon its arrival, and the more realistic non-clairvoyant case.

Categories and Subject Descriptors

F.1.2 [**Computation by abstract devices**]: Modes of Computation—*Online computation*

General Terms

Algorithms, Theory

Keywords

online algorithms, scheduling, cloud computing, competitive ratio, clairvoyant, non-clairvoyant

1. INTRODUCTION

Scheduling problems involve jobs that must be scheduled on machines subject to certain constraints to optimize a given objective function. The goal is to compute a schedule that specifies when and on which machine each job is to be executed. In online scheduling, the scheduler receives jobs that arrive over time, and generally must schedule the jobs without any knowledge of the future.

Cloud Computing is a new paradigm for provisioning computing instances, i.e., virtual machines (VMs) to execute jobs in an on-demand manner. This paradigm shifts the location of the computing infrastructure from the user site to the network thereby reducing the capital and management costs of hardware and software resources [7]. Public cloud is available in a pay-as-you-go charging model that allows end-users to pay for VMs by the hour e.g., $0.12 per hour. Two key criteria determine the quality of the provided service: (a) the dollar price paid by the end-user for renting VMs and (b) the maximum delay among all given tasks of a job. The goal is to provide a scheduling algorithm that aims to minimize the delay and the production cost of executing a job. We consider arbitrary jobs such as MapReduce jobs, Monte Carlo simulations and generating search index from web documents.

In classical scheduling problems, the number of machines is fixed, and in sequence we have to decide which job to process on which machine. However, the cloud introduces a different model in which we can activate and release machines on demand, and thus control the number of machines being used to process the jobs. This highlights the trade-off between the number of machines used and the delay of processing the jobs. On one hand, if we didn't have to pay for each machine, we could use one machine for each task of the job, and reduce the delay to a minimum. On the other hand, if we want to minimize cost, we could only use a single machine for all tasks of the jobs in a work-conserving model.

In this paper we assume that all computing instances which are available for processing are initially inactive. In order to assign a task to any machine it should be activated first. To activate a machine there is a constant duration of setup until the machine is ready to process. Moreover, when a machine is no longer in use, it should be shut down, again, taking a constant duration for turning off. Both setup and shut down times are included in the production cost of this service, and therefore will be charged to the end-user. As a result, the number of machines allocated for a specific job has a major impact on the total cost.

Our goal is to minimize both the maximum delay (response time) and the total cost. The problem of finding the

[*]Supported in part by the Israel Science Foundation (grant No. 1404/10).

[†]Supported in part by the Google Inter-university center.

SPAA'13, June 23–25, 2013, Montréal, Québec, Canada.

right balance between delay and cost is a bi-objective optimization problem. The solution to this problem is a set of Pareto optimal points. A solution is Pareto optimal if it is impossible to find a solution which improves on one or more of the objectives without worsening any of the others.

Our Results.

The performance of an algorithm will be described by competitive analysis where α is the cost ratio of the algorithm to the optimum cost and δ is the delay ratio (see section 2 for details).

- For the known-duration case (i.e. clairvoyant, task duration known upon its arrival), we present an optimal algorithm (up to a constant factor) with $\alpha = (1+\epsilon)$, and at the same time $\delta = O\left(\frac{1}{\epsilon}\right)$.
- For the unknown-duration (i.e. non-clairvoyant) case, we present an optimal algorithm with $\alpha = (1+\epsilon)$, and at the same time $\delta = O\left(\frac{\log \mu}{\epsilon}\right)$, where μ is the ratio of the longest task duration of any job to the shortest task duration of any job.

Related work.

Due to the importance of task scheduling and because they are often NP-hard, these kinds of problems have been much studied. Surveys on scheduling algorithms and online scheduling can be found in [9], [15] and [11]. Perhaps the most intuitive measure of Quality of Service (QoS) received by an individual job is the flow time. The flow time F_i of the ith job is the difference between its completion time and its release date. This measurement is equivalent to the delay attribute in our objective function. We next summarize some of the main prior results for minimizing the total and maximum flow time of n tasks on fixed number of m parallel identical machines.

Total flow time: Algorithm SRPT (shortest remaining processing time) has, within a constant factor, the best possible competitive ratio of any online algorithm for minimizing total flow time. It is $\Theta(min(\log \mu, \log n/m))$-competitive (where μ is the ratio between the maximum and the minimum processing time of a job), and this is known to be optimal within a constant factor [10]. SRPT uses both job migrations and pre-emptions to achieve its performance. Awerbuch, Azar, Leonardi, and Regev developed an algorithm without job migration (each job is processed on only one machine) that is $O(min(\log \mu, \log n))$-competitive [2]. Chekuri, Khanna, and Zhu developed a related algorithm without migration that is $O(min(\log \mu, \log n/m))$-competitive. These algorithms utilize a central pool to hold some jobs after their release date. Avrahami and Azar developed an algorithm without migration with immediate dispatch (each job is assigned to a machine upon its release) that is $O(min(\log \mu, \log n))$-competitive [1].

While SRPT and the related algorithms perform well on average, they may starve some jobs in order to serve most jobs well. The best results known for maximum flow time objective function come from Bender, Chakrabarti, and Muthukrishnan [4]. They show that FIFO (first in first out) is $(3 - 2/m)$-competitive, and provide a lower bound of 4/3 for any non-preemptive algorithm for $m \geq 2$.

Non clairvoyant makespan: A general reduction theorem from [16] shows that in any variant of scheduling in online environment with makespan objective, any batch-style c-competitive algorithm can be converted into a $2c$-competitive algorithm in a corresponding variant which in addition allows release times. In [5] it is proved that for a certain class of algorithms the competitive ratio is increased only by additive 1, instead of the factor of 2 in the previous reduction; this class of algorithms includes all algorithms that use a greedy approach similar to List Scheduling. The intuition beyond these reductions is that if the release times are fixed, the optimal algorithm cannot do much before the last release time. In fact, if the online algorithm would know which job is the last one, it could wait until its release, then use the batch-style algorithm once, and achieve the competitive ratio of $c + 1$ easily.

In addition, since the late 1980s, a lot of research was made with bi-criteria objective function. A survey on such multicriteria scheduling can be found in [8]. However, none of these scheduling setups are applicable to the cloud environment.

Cloud scheduling: There has been little theoretical work on online scheduling on computational grids and clouds (where grids consists of a large number of identical processors that are divided into several machines at possibly multiple locations). Moreover, as far as we know, there are no previous results which differ from our model by only one parameter.

In [17], Tchernykh et al. addressed parallel jobs scheduling problem for computational grid systems. They concentrate on two-level hierarchy scheduling: at the first level, a broker allocates computational jobs to parallel computers. At the second level, each computer generates schedules of the parallel jobs assigned to it by its own local scheduler. Selection, allocation strategies, and efficiency of proposed hierarchical scheduling algorithms were discussed. Later, in [18], Tchernykh et al. addressed non-preemptive online scheduling of parallel jobs on two stage grids. They discussed strategies based on various combinations of allocation strategies and local scheduling algorithms. Finally, they proposed and analysed a scheme named adaptive admissible allocation. This includes a competitive analysis for different parameters and constraints. They showed that the algorithm is beneficial under certain conditions and allows for an efficient implementation in real systems. Furthermore, a dynamic and adaptive approach is presented which can cope with different workloads and grid properties.

Schwiegelshohn et al. [12] addressed non-clairvoyant and non-pre-emptive online job scheduling in grids. In their model, jobs have a fixed degree of parallelism, and the goal is to minimize the total makespan. They showed that the performance of Garey and Graham's list scheduling algorithm [6] is significantly worse in grids than in multiprocessors, and presented a grid scheduling algorithm that guarantees a competitive factor of 5.

In [13], Schwiegelshohn studied the case of non-clairvoyant scheduling on massively parallel identical processors. The author pointed out the disadvantages of commonly used metrics like makespan or machine utilization. Instead, he suggested to use the total weighted completion time metric. He showed that this metric exhibits many properties that are similar to the properties of the makespan objective. Later, in [14], he showed that no constant competitive factor exists for the extension of the problem to rigid parallel jobs.

Outline.

The remainder of this paper is as follows. Section 2 describes the scheduling models, the function for computing the quality of service (QoS), and a brief summary of our results. Section 3 presents the basic observations used throughout this paper. In sections 4 and 5 we present our algorithms, and give proofs for lower bound for the known-duration and unknown-duration cases, respectively.

2. THE MODEL

Input.

The job input consists of multiple tasks that need to be executed. Tasks arrive over time. A task i has an arrival time a_i and (possibly unknown) duration p_i. We denote by p the minimum duration of a task (we assume it is known) and by P the maximum duration. Let $\mu = P/p$.

Model.

Each task runs on a single machine (instance). Each machine can run a single task at a time. Tasks are non-preemptive, i.e., a task has to run continuously without interruptions. Let $e_i = a_i + p_i$ which is the earliest possible completion time of task i. Denote by c_i the actual completion time of task i and $d_i = c_i - e_i$ as the delay that the task encounters. We can activate or shut down machines at any time. In activation of a machine there is T_{setup} time until the machine is available for processing. In shut down there is $T_{shutdown}$ time to turn off the machine. For simplicity we may assume that there is only activation time $T_s = T_{setup} + T_{shutdown}$ and the shut-down is free.

We concentrate on the online problem where no information is known on future arrival of tasks, but that the arrivals of tasks are independent of the scheduling. We consider the known and unknown task duration model (i.e. clairvoyant and non-clairvoyant). In the clairvoyant case the duration of a task is known at its arrival. In the non-clairvoyant model the duration is unknown at its arrival and becomes known only once the task has been completed.

The Algorithm.

At any time t the algorithm needs to decide how many machines $M(t)$ to maintain (and hence to activate or to shut down machines). In addition it should decide for each task when and on which machine to run it. Since pre-emption is not allowed, once a task is started to be processed by a machine it has to run till completion without any interruption.

Goal function.

The goal function consists of two parts: cost and delay. Without loss of generality assume that the cost charged per machine per unit of time is one. Then

$$G = \int_t M(t)dt$$

is the dollar cost of the algorithm, and G_{Opt} is the dollar cost of the optimal algorithm. We would like to find an algorithm for which the dollar cost is close to the optimum while the the maximum delay of any task is small. Let D be the maximum delay of the online algorithm (i.e. $d_i \leq D$ for all tasks i), and D_{Opt} be the maximum delay of the optimal algorithm. Formally the performance of an algorithm will be described by α - the cost ratio of the algorithm to the optimum cost ($\alpha = G/G_{Opt}$), and δ - the delay ratio ($\delta = D/D_{Opt}$). Let $W = \sum p_i$ be the total volume of the tasks. Clearly W is a lower bound for the cost. Actually, our online algorithm will be compared with the volume (as long as the volume is not too small). This corresponds to an optimal algorithm with no restriction on the delay (and hence the delay may be unbounded). On the other hand, the delay given by our algorithm will be compared to an aggressive algorithm with a maximum delay of only T_s which is the minimum possible (assuming at least one task needs to wait for the setup time). Surprisingly, we can provide an online algorithm whose performance compares favorably with the two optimal offline algorithms.

Remark: Let L be the difference between the latest and earliest release times. We assume that the total volume of tasks to process is at least the total duration of the process. Alternatively we may assume that the optimal algorithm is required to maintain at least one open machine at any time, and hence

$$G_{Opt} \geq L. \tag{1}$$

The case in which the volume is extremely small and there is no need to maintain one open machine is not of an interest (as the total work/cost is very small). If we insist in dealing with this case we can simply add an additive constant L. For the remainder of this paper we assume that the cost of the optimum is at least L.

3. BASIC OBSERVATIONS

We first show that it is not hard to have an algorithm with an optimal cost. Recall that W is the total volume of all tasks.

OBSERVATION 3.1. *For any algorithm (in particular the optimal algorithm) $G_{Opt} \geq W + T_s$. By (1) we also have $G_{Opt} \geq L$. Equality may be achieved only when a single machine is activated. Hence maintaining a single machine is optimal with respect to the cost (in a work-conserving system).*

The drawback of such extremely conservative algorithm is that the delays are not bounded. The queue of tasks as well as the delay of each task may go to infinity.

The other extreme possibility is to use a very aggressive algorithm with possibly low efficiency with respect to the cost but with a small delay in the completion of each task. Recall that p is the minimum duration of a task.

OBSERVATION 3.2. *An algorithm which activates a new machine for each task upon its arrival has a cost ratio of at most $T_s/p + 1$ and a maximum delay of T_s.*

The guarantee on the delay is excellent but the cost may be very large compared to the optimal cost. We are interested in achieving a much better efficiency, that is $1 + \epsilon$ approximation to the cost with reasonable delays. We note that if $T_s = 0$ then the optimal algorithm is trivial.

OBSERVATION 3.3. *If $T_s = 0$ then any algorithm with no idle machine is optimal with respect to the cost G_{Opt}. In particular, the algorithm which activates a new machine for each task upon its arrival is optimal. That is, the cost is the optimal cost (ratio of 1) and the delay of each task is 0 as task i is completed at e_i.*

4. KNOWN DURATION OF TASKS

In this section we first assume that the duration of each task p_i is known upon its arrival. Let $E = \frac{2T_s}{\epsilon}$ (for $0 < \epsilon < 1$). The following algorithm achieves a cost ratio of $(1+\epsilon)$ and a delay ratio of $O\left(\frac{1}{\epsilon}\right)$.

Algorithm Clairvoyant

- Classify a task upon its arrival. It is a long task if $p_i \geq E$ and otherwise short.
- Upon the arrival of each new long task, activate a new machine.
- Accumulate short tasks. Activate a machine to process those tasks at the earliest between
  - case 1: the first time the volume of the accumulated tasks becomes above E. In this case, assign the tasks to a new allocated machine and restart the accumulation.
  - case 2: $(E - T_s)$ time passed from the earliest release time of those tasks. In this case, continue the accumulation until the machine is ready (at time E), assign the tasks to the machine, and then restart the accumulation. If the volume of the tasks exceeds E by the time the machine is ready, stop the accumulation and go to case 1 (these tasks will be classified as case 1).
- Process the tasks on their assigned machine according to their arrival order. Note that the volume assigned to a machine is below $2E$ and each task will start its processing within at most E time after the assigned machine is ready.
- Shut down a machine once it completes tasks assigned to it.

THEOREM 4.1. *The cost ratio of Algorithm Clairvoyant is at most $(1+\epsilon)$. Moreover a long task will have a delay of T_s. The delay of a short task is at most $2E = \frac{4T_s}{\epsilon}$.*

PROOF. Let m be the number of machines opened for long tasks plus the number of machines opened to process tasks from case 1, whose accumulative volume was above E. Let r be the number of machines opened for the tasks from case 2, of which at least E time passed from the first task's release time until the release time of the first task in the next short tasks group. Clearly

$$G_{Opt} \geq W + T_s \geq mE + T_s = \frac{2mT_s}{\epsilon} + T_s$$

and as assumed in (1):

$$G_{Opt} \geq L \geq (r-1)E = \frac{2(r-1)T_s}{\epsilon}.$$

The total setup time for all machines is $(r+m)T_s$. Hence the cost ratio is at most

$$\begin{aligned}\frac{W+(r+m)T_s}{G_{Opt}} &= \frac{W+T_s}{G_{Opt}} + \frac{(r-1)T_s}{G_{Opt}} + \frac{mT_s}{G_{Opt}} \\ &\leq \frac{W+T_s}{W+T_s} + \frac{(r-1)T_s}{2(r-1)T_s/\epsilon} + \frac{mT_s}{2mT_s/\epsilon} \\ &= 1 + \frac{\epsilon}{2} + \frac{\epsilon}{2} = 1+\epsilon\end{aligned}$$

as needed. Next we discuss the delays of the tasks. Once a long task arrives it will start running after the setup time and hence be delayed by at most T_s. A short task will get a machine after at most $E - T_s$ time, then the machine has T_s setup time and finally it will start running in at most additional E time. Hence its total delay is $2E$ as claimed. □

We can further save cost and reduce the delay of Algorithm Clairvoyant although it does not improve the performance in the worst case. Specifically,

- Once a machine completed its assigned task it can process any waiting task. Shut down a machine once it becomes idle and there are no waiting tasks.

Next we show that if we insist on a total cost of at most $(1+\epsilon)W$, then the loss in the delay ratio is required.

LEMMA 4.2. *If an online algorithm is limited to a cost of $(1+\epsilon)W$, then its delay is at least $\frac{T_s}{\epsilon}$, and the delay ratio is $\frac{1}{\epsilon}$. This is true even if all of the tasks are of the same duration.*

PROOF. For $0 < \epsilon \leq \frac{1}{2}$: let $p = P = T_s\left(\frac{1}{\epsilon} - 1\right)$ (all the tasks have the same duration). We construct an instance of 4 tasks for which the maximum delay of an online algorithm would have to be at least $\frac{T_s}{\epsilon}$. We denote by W_t the total volume released by time t.

- At time 0, two tasks arrive ($W_0 = \frac{2(1-\epsilon)T_s}{\epsilon}$).

  The online algorithm can not activate more than one machine. If it activates two machines, its cost will be

  $$2T_s + W_0 > 2(1-\epsilon)T_s + W_0 = \epsilon W_0 + W_0 = (1+\epsilon)W_0.$$

  Hence, its cost will exceed the limitation. Therefore the online algorithm activates only one machine. At time T_s, this machine will start to process the first task, and at time $T_s + p$ it will process the second.
- At time $T_s + p$, the next two tasks arrive.

Note that the maximum delay of this scheduling is at least the delay of task 2, which is:

$$T_s + p = T_s + \frac{(1-\epsilon)T_s}{\epsilon} = \frac{T_s}{\epsilon}.$$

The offline algorithm, on the other hand, knows that $W = W_{T_s+p} = \frac{4(1-\epsilon)T_s}{\epsilon} \geq \frac{2T_s}{\epsilon}$ from the beginning of the process. Hence, it can open two machines on the arrival of tasks 1 and 2. The total cost is

$$2T_s + W \leq \epsilon W + W = (1+\epsilon)W.$$

Clearly, the delay of the first two tasks is T_s. The next two tasks can be scheduled immediately after their arrival, at time $T_s + p$, exactly after the completion of the first two tasks (one task for each machine). Therefore, the maximum delay of the offline algorithm is T_s, and the competitive ratio is at least $\delta = \frac{1}{\epsilon}$. □

One may think that if we allow a total cost which is much higher than the total work then the delay ratio would become 1. We prove that this is not true. Specifically, if the cost is α times the total work, then the delay ratio is at least $1 + \frac{1}{2\alpha}$.

LEMMA 4.3. *If an online algorithm is limited to a cost of αW ($\alpha > 1$), its delay ratio is at least $1 + \frac{1}{2\alpha}$.*

PROOF. Let $p = P = \frac{T_s}{2(\alpha-1)}$ (all the tasks have the same duration). We construct an instance of 4 tasks for which the maximum delay of an online algorithm would have to be greater than the delay of an offline algorithm. We denote by W_t the total volume at time t.

- At time 0, two tasks arrive $\left(W_0 = \frac{T_s}{\alpha-1}\right)$.

  The online algorithm can not activate more than one machine. If it activates two machines, its cost will be

  $$2T_s+W_0 = (\alpha-1)\frac{2T_s}{\alpha-1}+W_0 > (\alpha-1)W_0+W_0 = \alpha W_0.$$

  Hence, its cost will exceed the limitation. Therefore the online algorithm activates only one machine. At time T_s, this machine will start to process the first task, and at time $T_s + p$ it will process the second.

- At time $T_s + p$, the next two tasks arrive.

The maximum delay of this scheduling is at least the delay of task 2, which is:

$$T_s + p = T_s + \frac{T_s}{2(\alpha-1)} \geq T_s\left(1+\frac{1}{2\alpha}\right).$$

The offline algorithm knows that $W = \frac{2T_s}{\alpha-1}$ from the beginning of the process. Hence, it can open two machines on the arrival of tasks 1 and 2, with total cost of

$$2T_s + W = (\alpha-1)\frac{2T_s}{\alpha-1} + W = (\alpha-1)W + W = \alpha W.$$

and a delay of T_s. The next two tasks can be scheduled immediately after their arrival, at time T_s+p, exactly after the completion of the first two tasks (one task for each machine). Therefore, the maximum delay of the offline algorithm is T_s, and the competitive ratio is at least $1 + \frac{1}{2\alpha}$. □

5. UNKNOWN DURATION OF TASKS

The algorithm Clairvoyant from section 4 depends on estimates of how long each task will run. However, user estimates of task duration compared to actual runtime are often inaccurate or overestimated [3]. In this section we focus on the non-clairvoyant case, where the task duration is known only on its completion.

Now we present our algorithm for the unknown duration case. We divide time into epochs and deal with the tasks in each epoch separately.

Algorithm Non-Clairvoyant

- Divide the time into epochs of $F = 4T_s/\epsilon$. Let B_0 be the set of tasks given initially (time 0) and for $k \geq 1$ let

  $$B_k = \{i|(k-1)F < a_i \leq kF\}.$$

  All tasks B_k are handled at time kF separately from tasks of $B_{k'}$ for $k' \neq k$.

- Let $n_k = |B_k|$ be the number of tasks arrived in epoch k.

- Let $m_k = \lceil n_k p/F \rceil$ and activate m_k machines.

- Process tasks on machines in arbitrary order for T_s+F time (this also includes setup times of newly activated machines) . If after $T_s + F$ time there are still waiting tasks then activate additional m_k machines set $m_k \leftarrow 2m_k$ and repeat this step (note that tasks that are already running will continue to run with no interruption).

- Shut down a machine once it becomes idle and there are no waiting tasks.

Recall that μ is the ratio of the longest to the shortest task duration.

LEMMA 5.1. *The cost ratio of Algorithm Non-Clairvoyant is* $(1+\epsilon)$. *Each task is delayed by at most* $O\left(\frac{T_s \log \mu}{\epsilon}\right)$.

PROOF. First we deal with the cost. Let W_k be the volume of tasks arrived in the kth epoch. Assume that doubling the number of machines happens $r_k \geq 0$ times until all these tasks are processed. Hence the final number of machines was $m_k 2^{r_k}$. The total setup times of the machines is $S_k = m_k 2^{r_k} T_s$. We first assume that $r_k \geq 1$. Since the tasks were not completed by $m_k 2^{r_k-1}$ machines it means that

$$W_k \geq m_k F(2^{r_k-1} + 2^{r_k-2} + \ldots + 2^0) \geq m_k F 2^{r_k-1} .$$

Hence

$$\frac{S_k}{W_k} = \frac{m_k 2^{r_k} T_s}{W_k} \leq \frac{m_k 2^{r_k} T_s}{m_k F 2^{r_k-1}} = \frac{2T_s}{4T_s/\epsilon} = \frac{\epsilon}{2} .$$

We are left with the case that $r_k = 0$ i.e.,all tasks were completed at the first round. Assume first that $n_k p/F > 1$ and hence $m_k \geq 2$. In this case $m_k = \lceil n_k p/F \rceil \leq 2n_k p/F$. Then

$$\frac{S_k}{W_k} = \frac{m_k T_s}{W_k} \leq \frac{m_k T_s}{n_k p} \leq \frac{2T_s n_k p/F}{n_k p} = \frac{2T_s}{F} = \frac{2T_s}{4T_s/\epsilon} = \frac{\epsilon}{2} .$$

The only remaining case is when $m_k = 1$ and $r_k = 0$ i.e., all tasks were completed by the single machine. In this case $S_k = T_s$.

Let K_1 be the set of k for which $m_k = 1$ and $r_k = 0$ and K_2 the set of all the other k's. Note that for every $k \in K_2$:

$$S_k \leq \frac{\epsilon}{2} W_k.$$

Hence

$$S = \sum_k S_k = \sum_{k\in K_1} S_k + \sum_{k\in K_2} S_k \leq \sum_{k\in K_1} T_s + \sum_{k\in K_2} \frac{\epsilon}{2} W_k.$$

Clearly, $\sum_{k\in K_2} W_k \leq W$. Recall that L is the difference between the latest and earliest release times. Since we divided that time to epochs of F we have $|K_1| \leq \lceil L/F \rceil$. We may assume that $L > F$ (otherwise we have only one epoch and one machine so it is easy) and hence $|K_1| \leq 2L/F$. Applying that, we get:

$$S \leq \sum_{k\in K_1} T_s + \sum_{k\in K_2} \frac{\epsilon}{2} W_k \leq \frac{2L}{F} T_s + \frac{\epsilon}{2} W.$$

Assigning the value of F, the assumptions of (1), and the fact that $G_{Opt} \geq W$:

$$S \leq \frac{2L}{F} T_s + \frac{\epsilon}{2} W \leq \frac{\epsilon}{2} G_{Opt} + \frac{\epsilon}{2} G_{Opt} = \epsilon G_{Opt}.$$

We conclude that the cost ratio is at most

$$\frac{G}{G_{Opt}} = \frac{W+S}{G_{Opt}} = \frac{W}{G_{Opt}} + \frac{S}{G_{Opt}} \leq \frac{W}{W} + \frac{\epsilon G_{Opt}}{G_{Opt}} = 1+\epsilon$$

as needed.

Next we consider the delay. By volume consideration the number of machines in epoch k is at most $\lceil n_k P/F \rceil = \lceil n_k p\mu/F \rceil$, i.e., roughly μ times the initial number of machines for this epoch. Hence the doubling can happen at most $\log(\mu)+1$ times. Hence the delay is at most $F + (F + T_s)(\log\mu + 2) = O\left(\frac{T_s \log\mu}{\epsilon}\right)$.

□

LEMMA 5.2. *If an online non-clairvoyant algorithm is limited to a cost of $(1+\epsilon)W$, then its delay ratio is $\Omega\left(\frac{\log\mu}{\epsilon}\right)$.*

PROOF. Let $p = \frac{T_s}{\epsilon(2\mu-1)}$, and hence $P = p\mu$. We construct an instance of $n_0(\log\mu+1)$ tasks for which the maximum delay of an online algorithm is $\Omega\left(\frac{T_s\log\mu}{\epsilon}\right)$ and the delay of an offline algorithm is T_s. We divide the tasks to $(\log\mu+1)$ groups (each of size n_0). Tasks of group i (for $0 \le i \le \log\mu$) will be of length $p_i = p2^i$ and will arrive at $a_i = \sum_{k=0}^{i-1} p_k$ ($a_0 = 0$).

A machine is alive at time t if it was activated before t and shut down after t. We denote by $m(t)$ the number of the live machines at time t. Clearly, the total cost of the online algorithm is at least $m(t)T_s + W$, for any t. Moreover, it is limited, by assumption, to $(1+\epsilon)W$, and therefore

$$m(t) \le \frac{\epsilon}{T_s} W. \tag{2}$$

For any given time t^*, the online algorithm processed volume which is at most $\int_0^{t^*} m(t)\,dt$. At time t^* the online algorithm only knows the duration of the completed tasks. It may be possible that the duration of each of the remaining tasks is p. Therefore it may be possible that W is at most

$$pn_0(\log\mu+1) + \int_0^{t^*} m(t)\,dt. \tag{3}$$

Assigning (3) in (2), we get that the maximum number of live machines is

$$m(t^*) \le \frac{\epsilon}{T_s}\left(n_0 p(\log\mu+1) + \int_0^{t^*} m(t)\,dt\right). \tag{4}$$

Hence,

$$\frac{m(t^*)}{\left(n_0 p(\log\mu+1) + \int_0^{t^*} m(t)\,dt\right)} \le \frac{\epsilon}{T_s}.$$

By integrating both sides over t^*

$$\ln\left(\frac{n_0 p(\log\mu+1) + \int_0^{t^*} m(t)\,dt}{n_0 p(\log\mu+1)}\right) \le \frac{\epsilon t^*}{T_s},$$

which yields

$$\frac{n_0 p(\log\mu+1) + \int_0^{t^*} m(t)\,dt}{n_0 p(\log\mu+1)} \le e^{\frac{\epsilon t^*}{T_s}},$$

and,

$$\frac{\epsilon}{T_s}\left(n_0 p(\log\mu+1) + \int_0^{t^*} m(t)\,dt\right) \le \frac{\epsilon}{T_s} n_0 p(\log\mu+1) e^{\frac{\epsilon t^*}{T_s}}. \tag{5}$$

By transitivity of (4) and (5),

$$m(t) \le \frac{\epsilon}{T_s} n_0 p(\log\mu+1) e^{\frac{\epsilon t}{T_s}}.$$

By definition, $W = \sum_{i=0}^{\log\mu} n_0 p 2^i = n_0 p(2\mu-1)$. Let C denote the completion time of the online algorithm. Hence we have the following:

$$\begin{aligned} n_0 p(2\mu-1) &= W \le \int_0^C m(t)\,dt \\ &\le \int_0^C \frac{\epsilon}{T_s} n_0 p(\log\mu+1) e^{\frac{\epsilon t}{T_s}}\,dt \\ &= n_0 p(\log\mu+1)\left(e^{\frac{\epsilon C}{T_s}} - 1\right). \end{aligned}$$

Hence,

$$\frac{2\mu-1}{\log\mu+1} \le e^{\frac{\epsilon C}{T_s}} - 1$$

and therefore

$$\frac{T_s}{\epsilon}\ln\left(\frac{2\mu-1}{\log\mu+1}+1\right) \le C.$$

The maximum delay of the online algorithm is at least the delay of the last completed task. Recall that $e_i = a_i + p_i$ is the earliest possible completion time of tasks of group i, and that $d_i = c_i - e_i$ is the delay that the task encounter. For each i

$$e_i \le \sum_{k=0}^{\log\mu-1} p_k + p_{\log\mu} = p(2^{\log\mu+1}-1) \le p(2\mu-1) = \frac{T_s}{\epsilon}.$$

Therefore, the maximum delay D is at least

$$D \ge \frac{T_s}{\epsilon}\ln\left(\frac{2\mu-1}{\log\mu+1}+1\right) - \frac{T_s}{\epsilon} = \Omega\left(\frac{T_s\log\mu}{\epsilon}\right).$$

The offline algorithm uses n_0 machines. Its total cost is

$$n_0 T_s + W = n_0 T_s \frac{\epsilon(2\mu-1)}{\epsilon(2\mu-1)} + W = n_0 p\epsilon(2\mu-1) + W = (\epsilon+1)W.$$

Hence, the cost is bounded as needed. Since the offline algorithm knows the real value of W, it uses all n_0 machines from the beginning of the scheduling. Each machine schedules exactly one task of each group. At time T_s each machine starts to process a task from group 0, at time $T_s + a_i$ it completes task from group $i-1$, and starts to process task from group i. For each task in group i: $e_i = a_i + p_i$, and $c_i = T_s + a_i + p_i$. Therefore, each tasks is delayed by T_s, and the delay ratio is $\Omega\left(\frac{\log\mu}{\epsilon}\right)$ for all online algorithms with cost limited by $(1+\epsilon)W$. □

6. REFERENCES

[1] Nir Avrahami and Yossi Azar. Minimizing total flow time and total completion time with immediate dispatching. In *Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, pages 11–18, 2003.

[2] Baruch Awerbuch, Yossi Azar, Stefano Leonardi, and Oded Regev. Minimizing the flow time without migration. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 198–205, 1999.

[3] Cynthia Bailey Lee, Yael Schwartzman, Jennifer Hardy, and Allan Snavely. Are user runtime estimates inherently inaccurate? In *Job Scheduling Strategies for Parallel Processing*, volume 3277, pages 253–263. 2005.

[4] Michael A. Bender, Soumen Chakrabarti, and S. Muthukrishnan. Flow and stretch metrics for scheduling continuous job streams. In *Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*, pages 270–279, 1998.

[5] Anja Feldmann, Bruce Maggs, Jiri Sgall, Daniel D. Sleator, and Andrew Tomkins. Competitive analysis of call admission algorithms that allow delay, 1995.

[6] M. R. Garey and R. L. Graham. Bounds for multiprocessor scheduling with resource constraints. *SIAM Journal on Computing*, 4(2):187–200, 1975.

[7] Brian Hayes. Cloud computing. *Communications of the ACM*, 51(7):9–11, July 2008.

[8] Han Hoogeveen. Multicriteria scheduling. *European Journal of Operational Research*, 167(3):592 – 623, 2005.

[9] David Karger, Cliff Stein, and Joel Wein. Scheduling algorithms. *Algorithms and Theory of Computation Handbook*, 1997.

[10] Stefano Leonardi and Danny Raz. Approximating total flow time on parallel machines. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 110–119. ACM, 1997.

[11] Kirk Pruhs, Jiri Sgall, and Eric Torng. Online scheduling. pages 115–124, 2003.

[12] U. Schwiegelshohn, A. Tchernykh, and R. Yahyapour. Online scheduling in grids. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1 –10, april 2008.

[13] Uwe Schwiegelshohn. An owner-centric metric for the evaluation of online job schedules. *Proceedings of the 2009 multidisciplinary international conference on scheduling: theory and applications*, pages 557–569, 2009.

[14] Uwe Schwiegelshohn. A system-centric metric for the evaluation of online job schedules. *Journal of Scheduling*, 14:571–581, 2011.

[15] Jiri Sgall. On-line scheduling. In *Developments from a June 1996 seminar on Online algorithms: the state of the art*, pages 196–231, 1998.

[16] David B. Shmoys, Joel Wein, and David P. Williamson. Scheduling parallel machines on-line. In *Proceedings of the 32nd annual symposium on Foundations of computer science*, pages 131–140, 1991.

[17] Andrei Tchernykh, Juan Ram?rez, Arutyun Avetisyan, Nikolai Kuzjurin, Dmitri Grushin, and Sergey Zhuk. Two level job-scheduling strategies for a computational grid. In *Parallel Processing and Applied Mathematics*, volume 3911, pages 774–781. 2006.

[18] Andrei Tchernykh, Uwe Schwiegelshohn, Ramin Yahyapour, and Nikolai Kuzjurin. On-line hierarchical job scheduling on grids with admissible allocation. *Journal of Scheduling*, 13:545–552, 2010.