A Note on On-line Load Balancing for Related Machines

Piotr Berman * **Marek Karpinski** †

Abstract

We describe an on-line algorithm for scheduling permanent jobs on related machines. It achieves the competitive ratio of $3 + \sqrt{8} \approx 5.828$ for the deterministic version, and $3.31/\ln 2.155 \approx 4.311$ for its randomized variant, improving the previous competitive ratios of 8 and $2e \approx 5.436$.

*Dept. of Computer Science & Eng., Pensylvania State University, University Park, PA16802 Email:berman@cse.psu.edu

†Dept. of Computer Science, University of Bonn, 53117, Bonn. This research was partially supported by the DFG Grant KA 673/4-1 Email:marek@cs.uni-bonn.de

1 Introduction

We are given a set of machines that differ in speed but are related in the following sence: a job of size p requires time p/v on a machine with speed v. While we cannot compare structurally different machines using with a single speed parameter, it is a reasonable approach when the machines are similar; in other cases it may be a good approximation.

Our task is to allocate a sequence of jobs to the machines in an on-line fashion, while minimizing the maximum load of the machines. This problem was solved with a competitive ratio 8 by Aspnes *et al.* [1]. Later, it was noticed by Motwani [2] that by randomizing properly the key parameter of the original algorithm the expected competitive ratio can be reduced to $2e$.

Adapting their notation, we have n machines with speeds $v_1, \ldots, v_n$ (for later convenience, we assume that the sequence of speeds is nondecreasing) and a stream of m jobs with sizes $p_1, \ldots, p_m$. A schedule s assigns to each job j the machine $s(j)$ that will execute it. We define the load of a machine i and the load of entire schedule s as follows:

$$load(s,i) = \frac{1}{v_i} \sum_{s(j)=i} p_j, \qquad Load(s) = \max_i \; load(s,i)$$

It is easy to observe that finding an optimum schedule s^* is NP-hard offline, and impossible on-line. We want to minimize the competitive ratio of our algorithm, i.e. the ratio $Load(s)/Load(s^*)$ where s is the schedule resulting from our on-line algorithm, and s^* is an optimum schedule.

We describe an on-line scheduling algorithm with competitive ratio $3 + \sqrt{8} \approx 5.828$ for the deterministic version, and $3.31/\ln 2.155 \approx 4.311$ for its randomized variant.

2 Preliminaries

The idea of the improvement is the following. There exists a simple algorithm that achieves competitive ratio 2 if we know exactly the optimum load Λ: we simply assign each job to the slowest machine that would not increase its load above 2Λ. Because we do not know Λ, we make a safely small initial guess and later double it whenever we cannot schedule a job within the current load treshold.

Our innovation is to double (or rather, increase by a fixed factor r) the guess as soon as we can prove that it is too small, without waiting for the time when we cannot schedule the subsequent job. Intuitively, we want to avoid wasting the precious capacity of the fast machines with puny jobs that could be well served by the slow machines. Therefore we start from describing our method of estimating the necessary load.

Let $V = \{0, v_0, \ldots, v_n\}$. For $v \in V$ we define $Cap(v)$ as the sum of speeds of these

machines that have speed larger than v. (*Cap* stands for capacity, note that $Cap\ (0)$ is the sum of speeds of all the machines and $Cap(v_n) = 0$.) For a set of jobs J and a load treshold Λ we define $OnlyFor(v, \Lambda, J)$ as the sum of sizes of these jobs that have $p_j/v > \Lambda$. (*OnlyFor* stands for the work that can be performed only by the machines with speed larger than v if the load cannot exceed Λ.) The following observation is immediate:

Observation 1. For a set of jobs J, there exists a schedule s with $Load(s) \leq \Lambda$ only if $OnlyFor(v, \Lambda, J) \leq \Lambda Cap(v)$ for every $v \in V$.

If the set of jobs J satisfies the above condition, we say that Λ is *appropriate* for $v_1, \ldots, v_n$ and J.

Before we formulate and analyze our algorithm, we will show how to use the notions of *Cap* and *OnlyFor* to analyze the already mentioned algorithm that keeps the load under 2Λ if load Λ is possible off-line. We reformulate it to make it more similar to the new algorithm. Machine i has *capacity* $c_i = \Lambda v_i$ equal to the amount of work it can perform under Λ load, and the *safety margin* m_i to assure that we will be able to accomodate the jobs in the on-line fashion. In this algorithm the capacity and the safety margin are given the same value, in the new one they will be different.

```
(* initialize *)
for i ← 1 to n do
    m_i ← c_i ← Λv_i
j ← 0
(* online processing *)
repeat
    read(p)
    j ← j + 1
    for i ← 1 to n do
        if c_i + m_i > p then
            s(j) ← i
            c_i ← c_i − p
            exit for
forever
```

This algorithm shares the following property with the new one: the jobs are offered first to the machine 1 (the slowest), then to machine 2 etc. Given a stream of jobs J, we can define J_{i-1} as the stream of jobs that are passed over by machine $i-1$ or that reach machine i (for $1 \leq i < n$ this two conditions are equivalent, for $i = 0$ only the latter and for $i = n$ only the former applies). The correctness of the algorithm is equivalent to the fact that the stream J_n is empty—it consists of the jobs passed over by all the machines. From

the correctness the load guarantee follows easily, because the sum of sizes of jobs assigned to machine i is less than the initial capacity plus the safety margin, i.e. $\Lambda v_i + \Lambda v_i$, and so the load is less than $2\Lambda v_i / v_i = 2\Lambda$.

Observation 2. If there exists a schedule s^* with $Load(s^*) = \Lambda$, then for every $i = 0, \ldots, n$, Λ is appropriate for $v_{i+1}, \ldots, v_n$ and the stream of jobs J_i.

Proof. By induction on i. For $i = 0$ the claim follows from Observation 1. For the inductive step, we use notation $Cp_j(v)$ for $Cap(j)$ defined in respect to the sequence of machine speeds $v_j, \ldots, v_n$, and V_j to denote the set $\{0, v_j, \ldots, v_n\}$. Our inductive assumption says that

$$OnlyFor(v, \Lambda, J_{i-1}) \leq \Lambda Cap_i(v) \text{ for } v \in V_i$$

and have to show that

$$OnlyFor(v, \Lambda, J_i) \leq \Lambda Cap_{i+1}(v) \text{ for } v \in V_{i+1}$$

. Observe that for any $v \in V_{i+1} - \{0\}$, $OnlyFor(v, \Lambda, J_i) \leq OnlyFor(v, \Lambda, J_{i-1})$ and $Cap_{i+1}(v) = Cap_i(v)$. Thus it suffices to show that $OnlyFor(0, \Lambda, J_i \leq \Lambda Cap_{i+1}(0)$.

First observe that $Cap_{i+1}(0) = Cap_i(0) - v_i \geq Cap_i(v_i)$. We consider two cases according to the final value of c_i in the execution of the algorithm. If it is positive, then machine i accepted all jobs with size at most $m_i = \Lambda v_i$ from the stream J_{i-1}, hence $OnlyFor(0, \Lambda, J_i$, which is the sum of job sizes in J_i, is at most $OnlyFor(v_i, \Lambda, J_{i-1}$, which in turn is less or equal to $\Lambda Cap_i(v_i)$. Because $Cap_i(v_1) \leq Cap_{i+1}(0)$, the claim follows.

To finish the proof, we consider the case when the final value of c_i is negative or 0. Then total size of the jobs accepted by machine i is at least Λv_i, the initial value of c_i, hence $OnlyFor(0, \Lambda, J_i) \leq OnlyFor(0, \Lambda, J_{i-1}) - \Lambda v_i$, while $Cap_{i+1}(0) = Cap_i(0) - v_i$. Because one of our assumption is $OnlyFor(0, \Lambda, J_{i-1}) \leq Cap_i(0)$, the claim follows. □

Observation 2 implies that if a schedule with load Λ exists, then then Λ is appropriate for the empty sequence of the machines $v_{n+1}, \ldots, v_n$ and J_n. Thus the stream J_n of unscheduled jobs is empty, which means that the algorithm is correct.

3 The new algorithm

The next algorithm is similar, but it proceeds in phases, each phase having a different value of Λ. While it is correct for any value of the parameter $r > 1$, we will later find the optimum r's (they are different in the deterministic and randomized versions).

(* initialize *)
$\Lambda \leftarrow$ something very small

```
for i ← 1 to n do
    m_i ← c_i ← 0
j ← 0, J ← empty string
(* online processing *)
repeat
    read(p)
    j ← j + 1, p_j ← p, append J with p_j
    if Λ is not appropriate for v_1, ..., v_n and J then
        (* start a new phase *)
        multiply Λ by r until it is appropriate
        for i ← 1 to n do
            m_i ← Λv_i,    c_i ← c_i + m_i
    (* schedule p_j *) for i ← 1 to n do
        if c_i + m_i > p_j then
            s(j) ← i
            c_i ← c_i − p_j
            exit for
forever
```

We need to prove that the algorithm is correct, i.e. we never go through the last **for** loop without finding a machine with sufficient remaining capacity. We will say that executing this loop schedules p_j (even though, for the sake of argument, we admit the case that after executing this loop $s(j)$ remains undefined).

Let Λ_0 be the value of Λ when the first job was scheduled. We view the execution as consisting of phases numbered from 0 to k, where l-th phase operates with $\Lambda = \Lambda_l = \Lambda_0 r^l$. Let J^l be the stream of jobs received in phase l. Using the same convention as in the analysis of the previous algorithm, we define J^l_{i-1} to be the stream of jobs that in phase l machine i received or machine $i-1$ passed over. Now the correctness will mean that the stream J^l_n are empty for every phase l.

Because the initial estimate for Λ*maybetoolow*, machines may receive more work than in the previous algorithm. This is due to the fact that in the initial phases the machines from the beginning of the sequence needlessly refuse to pick jobs that they would gladly accept later, thus increasing the load of the end of the sequence. Nevertheless, as we shall show, this increase is limited.

As a preliminary, we need to analyze the consequences of the test that triggers a new phase as soon as Λ is not appropriate for the stream of jobs received so far. First of all, this implies that every Λ_i is appropriate for the stream $J_1 \cdots J^l$, and in particular, for the

substream J^l. Therefore

$$\mathit{OnlyFor}(v, \Lambda_l, J^l) \leq \Lambda_l \mathit{Cap}(v) \text{ for every phase } l \text{ and every } v \in V. \quad (\#)$$

This allows to prove, by induction on i, the following

Observation 3. For every $i = 0, \ldots, n$ and every phase l

$$\sum_{t=0}^{l} \mathit{OnlyFor}(0, \Lambda_t, J_i^t) \;\leq\; \left(\sum_{t=0}^{l} \Lambda_t\right)\left(\sum_{j=i+1}^{n} v_j\right)$$

For $i = 0$ this follows simply from the fact that for every phase $t \leq l$

$$\mathit{OnlyFor}(0, \Lambda_t, J_0^t) = \mathit{OnlyFor}(0, \Lambda_t, J^t) \leq \Lambda_t \mathit{Cap}(0) = \Lambda_t \left(\sum_{j=1}^{n} v_j\right).$$

For $l = 0$ the follows from Observation 2, as the phase 0 is identical to the first algorithm with $\Lambda = \Lambda_0$.

Therefore we may assume that the claim is true for $i-1$ and $l-1$. We consider two cases, according to the value of c_i at the end of phase l. Assume first that this value is positive. Subtract formally from both sides of the claim for i and l the respective sides of the claim for i and $l-1$; this way we see that it suffices to show that

$$\mathit{OnlyFor}(0, \Lambda_l, J_i^l) \;\leq\; \Lambda_t \left(\sum_{j=i+1}^{n} v_j\right)$$

Because the final value of c_i is positive, in phase l machine i accepted all jobs from the stream J_{i-1}^l that had size bounded by $\Lambda_l v_i$, and therefore the stream J_i^l consists only of the jobs that must be executed on machines faster than v_i. Thus the sum of sizes of all jobs in this stream, $\mathit{OnlyFor}(0, \Lambda_l, J_i^l)$, equals to $\mathit{OnlyFor}(v_i, \Lambda_l, J^l)$, which by (#) is at most $\Lambda_l \mathit{Cap}(v_i)$. Lastly, $\mathit{Cap}(v_i) \leq \sum_{j=i+1}^{n} v_j$.

Now assume that the final value of c_i in phase l equals some $c \leq 0$. This time subtract from both sides of the claim the respective sides of the claim for $i-1$ and l, this way we can see that it suffices to show that

$$\sum_{t=0}^{l} (\mathit{OnlyFor}(0, \Lambda_t, J_i^t) - \mathit{OnlyFor}(0, \Lambda_t, J_{i-1}^t)) \;\leq\; \left(\sum_{t=0}^{l} \Lambda_t\right) v_i \quad (\#\#)$$

On the left hand side this inequality has the difference between the sum of jobs sizes that reach machine i and the sum of the job sizes that are passed over by machine i to the subsequent machines (during the phases from 0 to l). In other words, this is the sum of sizes of the jobs accepted by machine i during these phases. This sum, say s, is related in the following manner to c:

$$0 \geq c = \left(\sum_{t=0}^{l} \Lambda_t v_i\right) - s \;\text{ which implies }\; s \geq \left(\sum_{t=0}^{l} \Lambda_t\right) v_i \;\equiv\; (\#\#). \quad \square$$

To analyze the competitive ratio, we may assume that $Load(s^*) = 1$. Then the penultimate value of Λ must be smaller than 1 and the final one smaller than r. Consider a machine with speed 1. The work accepted by a machine is smaller than the sum of all Λ's up to that time (additions to the capacity) plus the last Λ given for the safety margin. Together it is $(r + 1 + r^{-1} + \ldots) + r = r(1/(1 - r^{-1}) + 1) = r(2r - 1)/(r - 1)$. To find the best value of r, we find zeros of the derivative of this expression, namely of $(2r^2 - 4r + 1)/(r - 1)^2$, and solve the resulting quadratic equation. The solution is $r = 1 + \sqrt{1/2}$ and the resulting competitive ratio is $3 + \sqrt{8} \approx 5.8284$.

One can observe that the worst case occurs when our penultimate value of Λ is very close to 1 (i.e. to the perfect load factor). We will choose the initial value of Λ to be of the form r^{-N+x} where N is a suitably large integer and x is chosen, uniformly at random, from some interval $< -y, 1 - y >$ (we shifted the interval $< 0, 1 >$ to componsate for the scaling that made $Load(s^*) = 1$). Therefore we can replace the factor r with the average value of the last Λ. For negative x this value is r^{x+1}, for positive it is r^x. The average is

$$\int_{-y}^{0} r^{1+x} dx + \int_{0}^{1-y} r^x dx = \int_{1-y}^{1} r^x dx + \int_{0}^{1-y} r^x dx = \int_{0}^{1} r^x dx = \frac{r-1}{\ln r}$$

Therefore the average competitive ratio is

$$\frac{r-1}{\ln r}\frac{2r-1}{r-1} = \frac{2r-1}{\ln r}$$

The equation for the minimum value is kind of ugly, but nevertheless the minimum is achieved for r close to 2.155, and approximately equals 4.311.

References

[1] Aspnes, J., Y. Azar, A. Fiat, S. Plotkin and O. Waarts, *On-line load balancing with applications to machine scheduling and virtual circuit routing*, Proc. 25th ACM STOC, 623-631, San Diego, 1993.

[2] Motwani, R., personal communication through P. Indyk.