Lecture 12: Identical Parallel Machines ($P||C_{max}$)

June 11th, 2009

In $(P||C_{max})$ we have m machines and n jobs. Job j takes time p_j on each machine. This problem, as we saw in the last lecture, is NP-hard even when $m = 2$. Thus, we look at approximation algorithms. Throughout, we will let OPT denote the makespan of the optimal schedule. Recall, an factor $\rho > 1$ approximation algorithm returns a schedule S with $C^S_{max} \leq \rho \cdot OPT$.

We will use the following lower bounds on OPT. Note that,

$$OPT \geq p_j \quad \forall j \tag{1}$$

$$OPT \geq (\sum p_j)/m \tag{2}$$

1 List Scheduling: A factor 2 approximation algorithm

List scheduling is the following simple algorithm: Consider the jobs in any order. Schedule an unscheduled job in this order on the first available machine. Note that we do not require to know the processing times of all the jobs to schedule a particular job. Thus, this can be used as an *online* algorithm.

Theorem 1.1. *Let S be the schedule obtained from List-Scheduling. $C^S_{max} \leq (2 - 1/m) \cdot OPT$.*

Proof. Let ℓ be the last job to finish in S. Let t^S_ℓ be the time at which this job started. We know that at time t^S_l, all the machines were busy. Since no machine is idle, we have that the total processing times of all jobs apart from ℓ must be at least mt^S_l. That is,

$$t^S_l \leq (\sum p_j - p_l)/m \leq OPT - p_l/m$$

Also,

$$C^S_{max} = t^S_l + p_l \leq OPT + p_l(1 - 1/m) \leq (2 - 1/m) \cdot OPT$$

□

Let us see that the algorithm is tight for $m = 2$ machines. Consider the list of three jobs with processing times $\{1, 1, 2\}$. The list scheduling algorithm will process job 1 on machine 1, job 2 on machine 2, and job 3 on machine 1 again, say. Thus $C^S_{max} = 3$, while $OPT = 2$ with jobs 1 and 2 processed on machine 1, and job 3 on machine 2.

In the next section we see an improvement.

2 Longest Processing Time Rule (LPT)

The example of the previous section shows a drawback of list scheduling – the longest jobs should be processed earlier. In fact that precisely is the LPT rule: Sort the jobs in decreasing order of their processing times and run the list scheduling algorithm with this list.

Theorem 2.1. *LPT returns a schedule S such that $C^S_{max} \leq (4/3 - 1/3m)OPT$.*

Proof. Suppose, as in the previous proof, l is the last job to finish. We can assume that l is actually the last job. The reason is if not, then consider the jobs from $\{1, \ldots, l\}$ (assuming $p_1 \geq p_2 \geq \ldots$). LPT will return the same schedule and thus will have the same makespan as with $\{1, \ldots, n\}$ jobs. The optimum only decreases. Thus, if we show that the makespan on LPT on the first l jobs is within $4/3$ of the optimum of the first l jobs, we will be done. Thus, we will assume that $p_l = p_{min}$.

Lemma 2.2. *If $p_{min} > OPT/3$, then $C^S_{max} = OPT$.*

Proof. We actually show that if no optimum schedule processes more than two jobs per machine, then LPT will return the optimal schedule. This will imply the lemma. (Why?)

To see this, for every machine i, let i_1 and i_2 be the two jobs that the optimum schedules on it (i_2 being empty if only one job is scheduled) and suppose $p_{i_1} \geq p_{i_2}$. Also, order the machines from 1 to m such that $p_{i_1} \geq p_{i'_1}$ if $i < i'$. That is the largest job goes on the first machine, the second largest on the second machine, and so on. We claim that we may assume $p_{i_2} \leq p_{i'_2}$ for all $i < i'$. If not, a simple interchange argument will not decrease C_{max}.

We now claim that LPT returns the same schedule. Note that if LPT also puts at most two jobs on a machine then it precisely returns the above schedule. Suppose LPT processes three jobs on a machine. Let j be the first job which is put on a machine with two jobs already. Note that there at most $2m$ jobs. Call a job a *loner* if it is the only job processed by a machine. Note that if LPT processes three jobs on a machine there must be a loner job in LPT which is not a loner in the OPT schedule. Furthermore, when the job j is being processed by LPT, the loner must have already been processed. Therefore, the loner has processing time at least $2p_{min} > 2OPT/3$ (for otherwise j would be processed with the loner). Since it is not a loner in OPT, the job which goes with it in the optimum schedule must have processing time $< OPT/3$ contradicting the premise. □

Now we are almost done. Since the job which finishes last, l has $p_l = p_{min}$, if $p_l > OPT/3$, we get $C^S_{max} = OPT$. Otherwise, from the inequality in the previous proof

$$C^S_{max} \leq OPT + p_l(1 - 1/m) \leq (4/3 - 1/3m)OPT$$

□

Let us see that the algorithm is tight for $m = 2$ machines. Consider the list of five jobs with processing times $\{3, 3, 2, 2, 2\}$. LPT will return a schedule with $C^S_{max} = 7$, while $OPT = 6$ with jobs 1 and 2 processed on machine 1, and rest of the jobs on machine 2.