Scheduling Preemptable Tasks on Uniform Processors with Limited Availability for Maximum Lateness Criterion

Maciej Drozdowski, Jacek Błażewicz, Piotr Formanowicz
Institute of Computing Science, Poznań University of Technology, Poznań, Poland
{Maciej.Drozdowski,Jacek.Blazewicz,Piotr.Formanowicz}@cs.put.poznan.pl
Wiesław Kubiak
Faculty of Business Administration, Memorial University of Newfoundland, St. John's, Canada
Günter Schmidt
Department of Information and Technology Management, University of Saarland, Saarbrücken, Germany

Abstract

In this work the problem of scheduling n preemptable tasks with ready times and due-dates on m uniform processors available in q time windows for maximum lateness criterion is studied. The problem is reduced to a sequence of network flow problems. The complexity of algorithm is $O((n+q)^3(\log n + \log q + \log m + \log \max\{b_i\}))$, where b_i is speed of processor i.

1 Introduction

In this work we assume that processors are available only in restricted intervals of time, called time windows. Restricting availability of processing environment seems to be a reasonable assumption. For example, in computer systems real-time tasks are executed in fixed time periods. Thus, free time intervals for lower priority tasks are created. Other reasons for non-availability are maintenance periods or breakdowns.

Now, we formulate the problem. Let $\mathcal{P} = \{P_1, \ldots, P_m\}$ denote a set of *uniform processors*, where 'uniform' means that processors differ only in their speeds. We assume that there are $k \leq m$ different processor types. Processors of the type i have speed $b_i > b_{i+1}$ $(i = 1, \ldots, k-1)$. Processors are available in q different time windows. Let $t_1 < \ldots < t_q$ be the time moments where the availability of processors changes. We will denote by m_i^l the number of type i processors in interval $[t_l, t_{l+1})$. Let $\mathcal{T} = \{T_1, \ldots, T_n\}$ be the set of *tasks*. Task $T_j \in \mathcal{T}$ has processing requirement p_j, while its execution time on P_i is $\frac{p_j}{b_i}$. Tasks are preemptable, i.e. each task can be suspended and restarted later, possibly on a different processor without additional costs. Each task T_j has restricted processing interval determined by its ready time r_j and due-date d_j. The optimality criterion we consider is maximum lateness $L_{max} = \max_{T_j \in \mathcal{T}}\{c_j - d_j\}$, where c_j is the completion time of T_j. Thus, for a particular value of maximum lateness L_{max} task T_j must be completed by $d_j + L_{max}$. We will call value $d_j + L_{max}$, a deadline for T_j.

Our scheduling problem can be denoted as $Q, win|pmtn, r_j|L_{max}$ according to the three-field notation, extended by adding win to indicate that processors are available in time windows. The problem we consider here, and the algorithm proposed extend the results of [1]. The problem is reduced to a sequence of the network flow problems. We describe the construction of the network first. Then, we consider the complexity of this algorithm.

2 The algorithm

The network flow model solves problem $Q, win|pmtn, r_j, d_j|-$, i.e. it finds a feasible schedule provided that one exists for some given values of deadlines. Suppose the test value of the maximum lateness is given and is equal to L_{max}. This defines $K = 2n + q$ events in the task and processor system. The events are of the following type: ready time of some task, disappearing of some task T_j from the system at time $d_j + L_{max}$, a change of processor availability. However, assuming fixed L_{max}, the sequence of the events is also fixed. Let e_l be the time instant at which the lth event takes place, and $\tau_l = e_{l+1} - e_l$ be the length of the lth interval between two consecutive events.

The network $G(V, A)$ has a source node S_1, and terminal node S_2. Between these two, two layers of nodes are inserted. The first layer consists of n nodes T_j representing the tasks. The second layer has Kk nodes w_{il} $i = 1, \ldots, k, l = 1, \ldots, K$, representing different ranges of processor speeds in the interval $[e_l, e_{l+1}]$. The source node S_1 is connected by an arc of capacity p_j with the node representing task T_j. The capacities of these arcs guarantee that no task receives more processing than required. Nodes representing tasks which can be executed in the interval $[e_l, e_{l+1}]$ are connected with nodes w_{il} by edges with capacity $(b_i - b_{i+1})\tau_l$ $(i = 1, \ldots, k, b_{k+1} = 0)$. These arcs guarantee that no task is processed on any processor longer than possible in the interval of length τ_l. Interval-speed range nodes w_{il} are connected with the terminus S_2 by edges of capacity $(b_i - b_{i+1})\tau_l \sum_{z=1}^{i} m_z^l$. The constraints imposed by the capacities of the arcs heading to and leaving nodes w_{il} can be understood as equivalent to the processing capacity constraints imposed in [2] to solve problem $Q|pmtn|C_{max}$. The maximum flow can be found in $O((n + q)^3)$ time. When the edges joining the source with the task nodes are saturated, then a feasible schedule exists. The schedule can be constructed on an interval by interval basis. A partial schedule in the interval can be built using the algorithm proposed in [2].

Now, let us analyze the algorithm finding the optimum value of the maximum lateness L_{max}^*. With changing the value of L_{max}, the sequence of events in the system changes when for some pair of tasks T_i, T_j $r_i = d_j + L_{max}$. The sequence also changes with L_{max} when for some task T_j its deadline $d_j + L_{max}$ passes from one interval of processor availability to another. Hence, there are $O(n(n + q))$ intervals of L_{max} where the sequence of events is constant. The network flow algorithm must be called $O(\log n + \log q)$ times in a binary search fashion to determine the interval containing L_{max}^*.

Now we analyze number of the additional calls to the network flow algorithm needed to find L^*_{max}. After fixing the sequence of the events, the structure of network G remains fixed, only the values of arc capacities change. Suppose that we already fixed the sequence of events, and L^1_{max} is the biggest value of the maximum lateness considered in the previous binary search, for which a feasible solution does not exist. For L^1_{max} the maximum flow in G is $\phi_1 < \phi = \sum_{j=1}^{n} p_j$, where ϕ is the desired value of the flow. Now, we can find the cut i.e. the set of arcs with minimum capacity which is bounding the maximum flow.

With increasing of L^1_{max} by δ, the processing capacity of all arcs (w_{il}, S_2) $(i = 1, \ldots, k)$ for some interval $[e_l, e_{l+1}]$ increases by $\delta \sum_{i=1}^{k} (m_i^l b_i)$. Hence, the maximum possible increase of the capacity in the cut is $\delta \sum_{l=1}^{K} \sum_{i=1}^{k} (m_i^l b_i)$. On the other hand, task T_j which processing requirement is not satisfied can forward additional flow to all the speed-ranges of interval l via all arcs (T_j, w_{zl}), consuming at most $\delta b'_{l1}$ units of the flow, where b'_{l1} is the speed of the fastest processor in interval l. Hence, the minimum increase of the cut capacity is $\delta \min_{1 \le l \le K}\{b_{l1}\}$. The actual increase is $\mu^1 \delta$ where μ^1 is an integer multiplier satisfying $\min_{1 \le l \le K}\{b'_{l1}\} \le \mu^1 \le \sum_{l=1}^{K} \sum_{i=1}^{k} (m_i^l b_i)$, and reflecting which tasks can be executed in which interval, which intervals increase their capacity when L_{max} increases, and which of these combinations have arcs in the cut. We set $\delta = \frac{\phi - \phi_1}{\mu^1}$, $L^2_{max} := L^1_{max} + \delta$. The problem is to use the exact value of the multiplier, present in network G for the optimum L^*_{max}. By binary search over $\sum_{l=1}^{K} \sum_{i=1}^{k} (m_i^l b_i)$ values of multipliers one can find the right extension at which L^*_{max} is attained. Thus, the number of additional calls to the network flow algorithm is $O(\log n + \log q + \log m + \log \max\{b_i\})$, and total algorithm complexity is $O((n+q)^3(\log n + \log q + \log m + \log \max\{b_i\}))$.

References

[1] A.Federgruen, H.Groenevelt, Preemptive scheduling of uniform processors by ordinary network flow techniques, *Mgmt Sci.* **32**, 1986, 341-349.

[2] T.Gonzalez, S.Sahni, Preemptive scheduling of uniform processor systems, *Journal of the ACM* **25**, 1978, 92-101.