Provably good multiprocessor scheduling with resource sharing

Björn Andersson · Arvind Easwaran

Abstract We present a $12(1+3R/(4m))$ competitive algorithm for scheduling implicit-deadline sporadic tasks on a platform comprising m processors, where a task may request one of R shared resources.

Keywords Multiprocessor scheduling with resource sharing · Competitive ratio for multiprocessor resource sharing

1 Introduction

Consider scheduling of n sporadically arriving tasks on m identical processors. A task generates a sequence of jobs whose arrival times cannot be controlled by the scheduling algorithm and are *a priori* unknown. The time between two successive jobs by the same task τ_i is at least T_i. Every job by τ_i requires at most C_i units of execution over the next T_i time units after its arrival. If a task executes for L time units on a processor p having speed s, then the task performs $s \cdot L$ units of execution. A processor executes at most one job at a time and no job may execute on multiple processors simultaneously.

There is a set of R shared resources that tasks need in addition to the m processors. We assume that each task may *request* at most one shared resource from R, and further each job of that task may request the resource at most once during its execution; resource requests cannot be nested. The resource is *granted* to the job some time instant at or after the request, and then at some future time instant the job voluntarily *releases* the resource. We let $C_{i,k}$ denote the maximum amount of execution of a job of τ_i holding resource R_k. Jobs can access resources only in a mutually exclusive manner, and further, a job whose request is not granted cannot execute; it must wait. We however do not make any assumption on the placement of the resource request within a job's execution.

Our goal is to schedule all jobs to meet deadlines. In some scheduling problems, the concept of utilization bound has been used to characterize the performance of scheduling algorithms. But for characterizing scheduling algorithms for tasks that share resources, the utilization bound is an inappropriate metric because there exists a task set for which the utilization approaches zero and a deadline is missed regardless of algorithm used; this occurs when R contains just a single resource and all jobs require this single resource during their entire execution. But the resource-augmentation framework (Phillips et al. 1997) can be used in this context to characterize the performance: We say that an algorithm A has competitive ratio CPT_A if, for every real-time task set for which it is possible to meet deadlines, it holds that if the speed of each processor is multiplied by CPT_A then A will meet deadlines as well.

Low competitive ratio is preferred; ideally it should be one. A scheduling algorithm with a finite competitive ratio is desirable as well because it can ensure a designer that deadlines will be met by using faster processors. Consequently, the real-time systems community has embraced the development of scheduling algorithms with finite competitive ratio (Baruah et al. 2009). Unfortunately, the community has not yet developed a multiprocessor scheduling algorithm with finite competitive ratio for tasks that share resources, although many resource-sharing schemes (Block et al. 2007; Rajkumar et al. 1988; López et al. 2004; Gai et al. 2001; Easwaran and Andersson 2009) have been proposed.

In this paper, we present a new multiprocessor scheduling algorithm for tasks which share resources and prove that it has a finite competitive ratio of $12(1 + 3R/(4m))$. We call this algorithm gEDF-vpr, which stands for "**g**lobal **E**arliest **D**eadline **F**irst with **v**irtual **p**rocessor based **r**esource sharing".

2 Framework for gEDF-vpr algorithm

The main idea is to use m processors of speed 1 to emulate $2m + R$ virtual processors using generalized processor sharing[1] (Parekh and Gallager 1993). Specifically the following virtual processors are used: (1) m type-1 virtual processors of speed $2m/(4m + 3R)$ each, (2) R type-2 virtual processors of speed $3m/(4m + 3R)$ each, and (3) m type-3 virtual processors of speed $2m/(4m + 3R)$ each.

A task τ_i with unfinished execution is at every instant assigned to exactly one phase; see Fig. 1. When a job of τ_i arrives, τ_i is assigned phase-1. $T_i/3$ time units later

[1]Under generalized processor sharing, it is assumed that a virtual processor has access to some fraction of a physical processor at all times, and hence processing capacity is not wasted in this emulation.

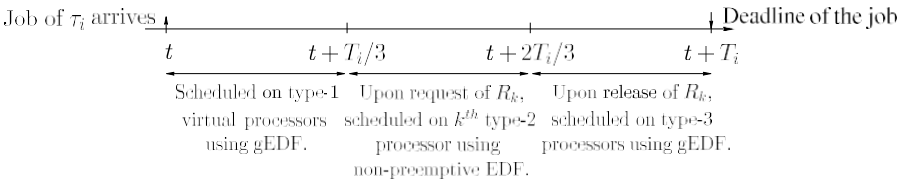

Fig. 1 Different execution phases of a job are scheduled on different sets of virtual processors

it is assigned phase-2, and an additional $T_i/3$ time units later it is assigned phase-3. Additionally, each phase is given a relative deadline of $T_i/3$ time units. A task performing no resource request is only in phase-1 and executes only on type-1 virtual processors. For a task τ_i which makes resource requests, the following apply: (1) It is ready for execution in phase-1 only if it has not yet made a resource request, (2) It is ready for execution in phase-2 only if it has not yet been granted the shared resource it requested,[2] and (3) It is ready for execution in phase-3 only if it has unfinished execution after releasing the shared resource.

Algorithm gEDF-vpr schedules (i) all phase-1 tasks onto type-1 processors using gEDF, (ii) a phase-2 task that requests resource R_k onto the kth type-2 processor using non-preemptive EDF, and (iii) all phase-3 tasks onto type-3 processors using gEDF. Thus, the use of non-preemptive scheduling on type-2 processors ensures that shared-resources are accessed in a mutually exclusive manner.

3 Competitive ratio of gEDF-vpr algorithm

In this section, we first present results on the competitive ratio of gEDF (multiprocessor) and non-preemptive EDF (single processor),[3] and then derive the competitive ratio for gEDF-vpr.

3.1 Fundamental results on competitive ratio

We let *sched(A,τ, m,s)* denote a predicate meaning that task set τ meets all deadlines when scheduled by algorithm A on m processors of speed s. The following lemma re-states the gEDF result.

Lemma 1 (Theorem 2.2 in Phillips et al. 1997) *sched(feasible,τ, m, 1)* $\Rightarrow$ *sched(gEDF,τ, m, 2), where "feasible" implies that there exists some schedule which meets all the deadlines; this schedule may use inserted idle time and it may be generated using future arrival times and it may be different from EDF.*

The following lemma presents a finite competitive ratio for non-preemptive EDF scheduling on a single processor, assuming, as in this paper, tasks have implicit deadlines. That is, T_i for task τ_i denotes not only the minimum time between successive job releases but also the relative deadline of jobs.

---

[2] Once the request is granted, execution occurs non-preemptively on the virtual processor in this phase.

[3] Although the gEDF result is known, we present the first competitive ratio for non-preemptive EDF.

Lemma 2 *sched(non-preemptive-feasible,τ, 1, 1) ⇒ sched(non-preemptive EDF, τ, 1, 3), where "non-preemptive-feasible" implies that there exists some non-preemptive schedule which meets all the deadlines; this schedule may use inserted idle time and it may be generated using future arrival times and it may be different from EDF.*

Proof Suppose the lemma is incorrect. Let us consider the schedule where τ failed under non-preemptive EDF; τ is assumed to be non-preemptive-feasible. Let t_1 denote the time when a deadline miss occurred, and let t_0 denote the earliest time, before t_1, when the processor transitioned from idle to busy. Let $t_= t_1 - t_0$. Further, let J_f denote a job of task τ_k which failed to meet a deadline at t_1.

Lower-priority jobs which arrive after the arrival of J_f cannot block J_f. Therefore any such job which blocked J_f must have arrived before J_f arrived. Further, since this job has lower priority than J_f, its deadline must be at t_1 or later (EDF scheduling). Hence, any task τ_i which blocks J_f with lower-priority jobs, must satisfy the condition $T_i \geq T_k$. Therefore, J_f can experience lower-priority blocking for at most $\max_{T_i \geq T_k} C_i/3$ time units. Here the execution is divided by 3 because processors under non-preemptive EDF are assumed to be 3 times as fast as the original ones. Since J_f missed a deadline

$$\sum_{j=1}^{n} \left\lfloor \frac{t'}{T_j} \right\rfloor \cdot C_j/3 + \max_{T_i \geq T_k} C_i/3 > t' \quad \text{and} \quad t' \geq T_k \tag{1}$$

Now for τ to be non-preemptive feasible, it holds that

$$\sum_{j=1}^{n} \left\lfloor \frac{t'}{T_j} \right\rfloor \cdot C_j \leq t' \quad \text{and} \quad \forall \tau_u, \tau_v : C_u \leq 2(T_v - C_v) \tag{2}$$

In the above equations, terms $\sum_{j=1}^{n} \lfloor \frac{t'}{T_j} \rfloor \cdot C_j/3$ and $\sum_{j=1}^{n} \lfloor \frac{t'}{T_j} \rfloor \cdot C_j$ denote the maximum higher-priority workloads that must be completed in a time interval of length t'. Also, the expression $\tau_u, \tau_v : C_u \leq 2(T_v - C_v)$ is necessary for non-preemptive feasibility as can be seen by considering the periodic arrival pattern of tasks τ_u and τ_v and realizing that τ_u needs a (non-existing) contiguous time window of length greater than $2(T_v -$ yields

$$\sum_{j=1}^{n} \left\lfloor \frac{t'}{T_j} \right\rfloor \cdot C_j > t' + 2 \cdot t' - \max_{T_i \geq T_k} C_i \tag{3}$$

Applying the two expressions in (2) and $t \geq T_k$ on (3) yields: $t > t \quad 2+ T_k \quad 2- \cdot (T_k - C_k)$. Subtracting t on both sides and rewriting yields $0 > 2 \quad C_k$. This is a contradiction and it proves the lemma. D

3.2 Competitive ratio of gEDF-vpr

We let $T D_1(\tau)$ denote a function which takes task set τ as a parameter and outputs a task set which differs from τ only in that for each task $\tau_i \in T D_1(\tau)$, the parameter

T_i is one third of the parameter T_i of the corresponding task in τ and we also set $C_{i,k} = 0$ for every resource and task.

For each resource $R_k \in R$, we let $TD_{2,k}(\tau)$ denote a function which takes task set τ as a parameter and outputs a task set constructed as follows:

1. $\forall \tau_i \in TD_{2,k}(\tau)$, the parameter T_i is one third of the parameter T_i of the corresponding task in τ.
2. $\forall \tau_i \in TD_{2,k}(\tau)$, $C_{i,j} = 0$ for all $j \neq k$.
3. $\forall \tau_i \in TD_{2,k}(\tau)$, $C_{i,k}$ is equal to $C_{i,k}$ of the corresponding task in τ.
4. $\forall \tau_i \in TD_{2,k}(\tau)$, C_i is equal to $C_{i,k}$ of the corresponding task in τ.

That is, task τ_i in $TD_{2,k}(\tau)$ accesses resource R_k throughout its execution and for the same amount of time as τ_i in τ accesses R_k. It is easy to see that $TD_1(\tau)$ models tasks in phases 1 and 3, whereas $TD_{2,*}(\tau)$ models tasks in phase 2.

In these definitions, we can intuitively understand the meaning of "TD" as "one ThirD". The following theorem proves the competitive ratio of gEDF-vpr.

Theorem 1 *Competitive ratio of gEDF-vpr is* $12(1 + 3R/(4m))$.

Proof Since processors that are 3 times as fast as the original ones make it feasible to meet deadlines that are one third of the original deadlines, we have by definition

$$\exists A : sched(A, \tau, m, 1) \Rightarrow \exists A : sched(A, TD_1(\tau), m, 3) \tag{4}$$

For each resource R_k, since R_k is accessed in a mutually exclusive manner, all the task executions in τ that use R_k must be sequential. Then, a single dedicated processor is sufficient to guarantee feasibility of all such task executions. The following equation is a consequence of this observation.

$$\forall R_k \in R, (\exists A : sched(A, \tau, m, 1) \Rightarrow \exists A : sched(A, TD_{2,k}(\tau), 1, 3)) \tag{5}$$

Now, using Lemma 1 for type-1 and type-3 virtual processors (one instance of (4) for each type), and Lemma 2 for type-2 virtual processors (R instances of (5)), we get

$$\text{type 1: } (\exists A : sched(A, \tau, m, 1)) \Rightarrow (sched(\text{gEDF}, TD_1(\tau), m, 6)), \tag{6}$$

$$\text{type 2: } \forall R_k \in R, (\exists A : sched(A, \tau, m, 1)) \Rightarrow (sched(\text{non-preemptive EDF}, TD_{2,k}(\tau), 1, 9)), \tag{7}$$

$$\text{type 3: } (\exists A : sched(A, \tau, m, 1)) \Rightarrow (sched(\text{gEDF}, TD_1(\tau), m, 6)) \tag{8}$$

Multiplying the processor speeds of (6)–(8) by $m/(12m + 9R)$ and using the emulation with virtual processors as mentioned in Sect. 2 gives

$$(\exists A : sched(A, \tau, m, m/(12m + 9R)) \Rightarrow (sched(\text{gEDF-vpr}, \tau, m, 1)) \tag{9}$$

This proves the theorem. D

4 Conclusions

This paper presented the first provably good multiprocessor scheduling algorithm for tasks that share resources. As stated in the introduction however, this initial result required some simplifying assumptions on the task model. It would be interesting to relax those assumptions; in particular, restrictions on deadlines and shared-resource access pattern of tasks.

References

Baruah S, Bonifaci V, Marchetti-Spaccamela A, Stiller S (2009) Implementation of a speedup-optimal global EDF schedulability test. In: Proceedings of Euromicro conference on real-time systems, pp 259–268

Block A, Leontyev H, Brandenburg BB, Anderson JH (2007) A flexible real-time locking protocol for multiprocessors. In: Proceedings of real-time and embedded computing systems and applications conference, pp 47–56

Easwaran A, Andersson B (2009) Resource sharing in global fixed-priority preemptive multiprocessor scheduling. In: Proceedings of IEEE real-time systems symposium, pp 377–386

Gai P, Lipari G, Di Natale M (2001) Minimizing memory utilization of real-time task sets in single and multi-processor systems-on-a-chip. In: Proceedings of IEEE real-time systems symposium, pp 73–83

López JM, Díaz JL, García FD (2004) Utilization bounds for EDF scheduling on real-time multiprocessor systems. J Real-Time Syst 28(1):39–68

Parekh A, Gallager R (1993) A generalized processor sharing approach to flow control—the single node case. IEEE/ACM Trans Netw 1(3):344–357

Phillips CA, Stein C, Torng E, Wein J (1997) Optimal time-critical scheduling via resource augmentation. In: Proceedings of the ACM symposium on theory of computing, pp. 140–149

Rajkumar R, Sha L, Lehoczky JP (1988) Real-time synchronization protocols for multiprocessors. In: Proceedings of IEEE real-time systems symposium, pp 259–269

.