An Optimal Resource Sharing Protocol for Generalized Multiframe Tasks
— Extended abstract —

Pontus Ekberg, Nan Guan, Martin Stigge, Wang Yi
Uppsala University, Sweden
Email: {pontus.ekberg|nan.guan|martin.stigge|yi}@it.uu.se

Abstract—We consider sharing of non-preemptable resources in real-time task models with flexible job release patterns. Resource sharing is an inherent property of many real-time systems. At the same time, flexible task models are needed to precisely express their workloads. Exact analysis and optimal scheduling of systems with shared resources have been available only for relatively simple task models, such as the sporadic task model.

We propose a new algorithm for scheduling systems with shared resources. The key idea behind the algorithm is to take the tasks’ structures into account when predicting possible resource contention. We show that the algorithm is optimal for scheduling generalized multiframe task sets with shared resources. We also present an efficient feasibility test for such systems, and show that the test is both sufficient and necessary.

I. INTRODUCTION

Processes in real-time systems often compete for shared resources, such as peripheral devices or global data structures that must be accessed in a mutually exclusive manner. To avoid deadlocks and low processor utilization, we need scheduling algorithms that handle the resource sharing.

Well-established solutions to the resource sharing problem exist in the context of sporadic task sets. The popular instantiation of the stack resource policy [1], often called EDF+SRP [2], is even optimal for such workloads. Unfortunately, EDF+SRP is not optimal for more flexible task models, such as the generalized multiframe (GMF) task model [3].

The flexible structure of GMF tasks, in combination with shared resources, is the main source of difficulty in finding an optimal scheduling strategy. To make optimal scheduling decisions at run-time, we must be aware of the tasks’ structures and predict which behaviors each task may display in the near future.

The goal of this work is to show how to analyze and schedule GMF task sets with shared resources. We introduce an efficient technique, which takes the tasks’ structures into account, to predict possible resource contention at run-time and thereby determine the urgency of unlocking currently used resources. Based on this technique we propose a new scheduling algorithm and show that it is well suited for scheduling such workloads. The main contributions include:

- We propose the virtual deadline protocol (VDP) for handling shared resources, and combine it with earliest deadline first (EDF) to form the EDF+VDP scheduling algorithm. We prove that EDF+VDP has the following properties:
  - It is optimal for scheduling GMF task sets with shared resources, in the sense that it successfully schedules all feasible task sets.
  - It is deadlock-free, and it enables efficient implementations because there is at most one preemption per job release and all jobs in the system can share a common run-time stack.
- We derive a sufficient and necessary feasibility test for GMF task sets with shared resources. This test is in the same complexity class as the known feasibility tests for sporadic tasks [4] and GMF task sets without resources [3], i.e., pseudo-polynomial for bounded-utilization task sets.

II. PRELIMINARIES

A. The Generalized Multiframe Task Model

The GMF task model [3] is a generalization of the well-known sporadic [4] task model. Like a sporadic task, a GMF task releases a sequence of jobs. However, the jobs released by a GMF task do not all need to have the same parameters (e.g., execution time and deadline). Instead, a GMF task cycles through a sequence of job types, which specify the parameters of the jobs that are released.

A natural way of representing a GMF task is to use a directed cycle graph, where the vertices represent the job types, and the arcs specify the order in which jobs are released (as well as the minimum delay between consecutive job releases). Formally, a GMF task set \( \tau \) is defined as follows:

- Each task \( T \in \tau \) is a directed cycle graph, with vertices \( V(T) \) and arcs \( A(T) \).
- Each vertex \( v \in V(T) \) is called a job type and is labeled with a pair \( \langle E(v), D(v) \rangle \). For each job that is of type \( v \), \( E(v) \in \mathbb{N}_{>0} \) is an upper bound on its required execution time, and \( D(v) \in \mathbb{N}_{>0} \) is its relative deadline.
- Each arc \( (u, v) \in A(T) \) is labeled with a minimum inter-release separation time \( P(u, v) \in \mathbb{N}_{>0} \).
- One vertex \( v_0 \in V(T) \), denoted \( S(T) \), is called the start vertex of \( T \).
When the system is running, each task $T$ releases a possibly infinite sequence of jobs $[J_0, J_1, J_2, \ldots]$, where each job corresponds to one of $T$’s job types. Intuitively, a job sequence is generated by “walking” through the graph of $T$, starting at vertex $S(T)$. Every time a vertex is visited, a job of the corresponding job type is released. Before the next vertex can be visited, the task must wait for at least the minimum inter-release separation time labeled on the arc leading there.

Formally, each job $J_i$ in a job sequence is specified by a triple $(r(J_i), e(J_i), d(J_i)) \in \mathbb{R}^3$, where $r(J_i)$ is the job’s absolute release time, $e(J_i)$ its absolute deadline and $d(J_i)$ its execution time requirement. A job sequence is said to be generated by $T$ if and only if there is a path $[v_0, v_1, v_2, \ldots]$ through $T$ such that the following hold for all $i \geq 0$:

1) $v_0 = S(T)$,
2) $r(J_{i+1}) \geq r(J_i) + P(v_i, v_{i+1})$,
3) $e(J_i) \leq E(v_i)$,
4) $d(J_i) = r(J_i) + D(v_i)$.

A job sequence is generated by a task set $\tau$ if and only if it is an interleaving of job sequences generated by the tasks $T \in \tau$.

We assume that the tasks satisfy the $l$-MAD property [3]: $D(u) \leq P(u, v) + D(v)$. This property guarantees that all jobs released by the same task have their (absolute) deadlines ordered in the same order as their release times.

### B. Modeling Shared Resources

In the plain GMF model described above, all jobs are completely independent; there is no way to model contention between jobs for shared resources. We extend the GMF model to include non-preemptable shared resources, which allows us to express which resources may be used by jobs of each job type, and for how long. We refer to the extended model as the GMF-R task model.

When a job is granted access to a resource, we say that it locks the resource, and then holds it for some time before finally unlocking it. If a resource is already held by some job, it cannot be locked again until it has been unlocked by the job holding it. Note that a job may be preempted while holding a resource, but no other job may use that resource until it is unlocked.

Each job type has a worst-case access duration to each resource. After a resource is locked, the job will execute for at most this duration before unlocking it again. We do not assume any a priori knowledge about exactly when a job locks a resource.

Formally, a GMF-R task set is a triple $(\tau, \rho, \alpha)$, such that

- $\tau$ is a GMF task set,
- $\rho$ is a set of resources,
- $\alpha : V(\tau) \times \rho \to \mathbb{N}_{\geq 0} \cup \{\perp\}$ is a function mapping job types and resources to their worst-case access durations,

where $V(\tau) = \bigcup_{T \in \tau} V(T)$ is the set of all job types in $\tau$.

The worst-case access duration of jobs of type $v$ to resource $R \in \rho$ is given by $\alpha(v, R)$. If $\alpha(v, R) = \perp$, then jobs of type $v$ do not use resource $R$.

Otherwise, $\alpha(v, R) \leq E(v)$ is assumed. We let $\alpha^{\max}(T, R)$ denote the maximum access duration to resource $R$ by any job type in task $T$. Fig. 1 shows an example GMF-R task set with two GMF tasks and two shared resources (only $\alpha(v, R) \neq \perp$ are shown).

A single job may use several different resources, possibly at the same time, but resource accesses must be properly nested. That is, if a job locks resource $R_1$ and afterwards locks $R_2$, it must unlock $R_2$ before unlocking $R_1$. A job may also lock the same resource several times during its execution (each time holding it for at most the worst-case access duration). A job must unlock all resources that it holds before finishing execution.

### C. Scheduling Algorithms and Feasibility

For a job sequence to be successfully scheduled, all jobs must finish their execution before their deadlines. That is, each job $J$ in the sequence must be exclusively executed for $e(J)$ time units (not necessarily continuously) between $r(J)$ and $d(J)$. A job is said to be active during the time interval between its release time and the time point where it finishes execution.

A scheduling algorithm decides at each time point which active, non-blocked job (if any) to execute. A scheduling algorithm can know the current system state and how jobs have been released in the past. It does not know what will happen in the future, other than what is specified by the task model.

**Definition II.1** (Feasibility and Optimality). A GMF-R task set $\langle \tau, \rho, \alpha \rangle$ is feasible if and only if there exists some scheduling algorithm that can successfully schedule all job sequences generated by $\tau$, for all legal access patterns to the resources in $\rho$ by jobs in the sequence.

A scheduling algorithm is optimal if and only if it can successfully schedule all feasible task sets.

---

1Note that $\alpha(v, R) = 0$ is useful to express that jobs of type $v$ can be forbidden to execute while some other job holds $R$, but do not hold $R$ themselves. This can be used to model non-preemptable sections in jobs.
III. The Virtual Deadline Protocol

The virtual deadline protocol (VDP) is a resource sharing protocol, designed to extend the earliest deadline first (EDF) scheduling algorithm to handle shared resources. We call the resulting scheduling algorithm EDF+VDP. We will show that EDF+VDP is an optimal scheduling algorithm for GMF-R task sets.

EDF+VDP uses what we call virtual deadlines to schedule jobs. It schedules jobs in a similar way to EDF, but uses virtual deadlines instead of absolute deadlines for scheduling decisions. That is, at each time point, EDF+VDP chooses to run the job with the earliest virtual deadline. It is then up to the VDP part of EDF+VDP to assign virtual deadlines to jobs in a way that guarantees the desired properties. It does this by potentially lowering the virtual deadlines (and thereby increasing the priorities) of jobs that are currently holding resources. The virtual deadline of a job therefore represents not only the urgency of the job itself, but also the urgency of releasing the resources that the job is currently holding. To assign virtual deadlines in an optimal way, we must be able to determine how urgent it is that a certain resource becomes unlocked. We capture this urgency by introducing the concept of a resource deadline, which is described in the following section.

A. Resource Deadlines

VDP relies on the idea that we can predict, at any time, exactly the earliest future time point where some not-yet-released job can have a deadline. In particular, we are interested in the deadlines of future jobs that may need some resource \( R \). The earliest possible such deadline we call the resource deadline of \( R \).

Definition III.1 (Resource deadline). The resource deadline \( \Delta(R, t) \) of resource \( R \) at time point \( t \) is exactly the earliest time point when some job that is not yet released at \( t \) and that may need \( R \) can potentially have a deadline, without violating the semantics of the task model.

In other words, if a task set \( \tau, \rho, \alpha \) has released a sequence of jobs \([J, J', \ldots, J'']\) up to time point \( t \), then \( \Delta(R, t) \) is the smallest value such that the following is satisfied, for some potential future job \( J''' \):

1) \( J''' \) may use \( R \),
2) \( \Delta(R, t) = d(J''') \),
3) \( \nu(J''') \geq t \),
4) some \([J, J', \ldots, J'', J''']\) is generated by \( \tau \).

Note that no future job that uses \( R \) actually has to get a deadline at \( \Delta(R, t) \), as long as it is possible, given the task model and the system state at time \( t \). We will show how resource deadlines can be efficiently computed at run-time.

Example III.2. Consider the task system in Fig. 1 and the following run-time scenario, illustrated in Fig. 2. At time 15, we want to know the resource deadline \( \Delta(R_1, 15) \). The latest job released by task \( T_1 \) was of type \( v_3 \) at time 11, and the latest job released by task \( T_2 \) was of type \( v_1 \) at time 2.

We can see that the next job of \( T_1 \) that may need \( R_1 \) is of type \( v_0 \). The earliest possible deadline of the next job of type \( v_0 \) is at \( 11 + 5 + 12 + 9 = 37 \). Similarly, the next job of \( T_2 \) that may need \( R_1 \) is of type \( v_2 \), and can have a deadline earliest at time \( 2 + 20 + 35 = 57 \). The earliest possible time when some future job that needs \( R_1 \) may have a deadline is therefore at time 37, and \( \Delta(R_1, 15) = 37 \).

B. The EDF+VDP Scheduling Algorithm

In EDF+VDP we use virtual deadlines to represent the urgency of executing jobs. The urgency of executing a job depends not only on the job itself (i.e., its absolute deadline), but also on whether the resources that it holds might be needed by some other job. We introduced resource deadlines to capture this latter aspect of the urgency.

By combining these aspects of urgency, we can now present the complete EDF+VDP scheduling algorithm, which is defined by the following four rules:

1) When a job \( J \) is released, it gets a virtual deadline \( \nu(J) \) equal to its absolute deadline:
   \[ \nu(J) \leftarrow d(J). \]
2) When a job \( J \) locks a resource \( R \) at time \( t \), \( \nu(J) \) is updated based on the resource deadline \( \Delta(R, t) \):
   \[ \nu(J) \leftarrow \min(\nu(J), \Delta(R, t)). \]
3) When a job unlocks a resource, it regains the virtual deadline that it had immediately before locking that resource.
4) At each time point, EDF+VDP executes the active job \( J \) with the earliest virtual deadline \( \nu(J) \). If several jobs share this earliest virtual deadline, then those jobs are executed in first-come, first-served order.

REFERENCES