technische universität
dortmund

fakultät für informatik
informatik 12

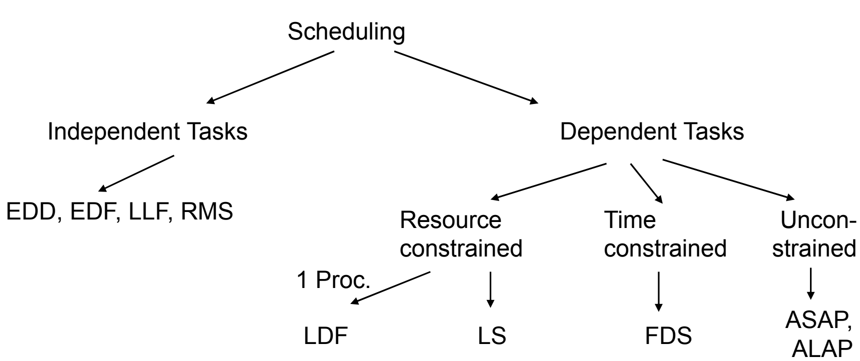
Mapping of Applications to Platforms

Peter Marwedel
TU Dortmund, Informatik 12
Germany

Graphics: © Alexandra Nolte, Gesine Marwedel, 2003

(2010年 12 月 10 日)
Subset of slides selected for EECS 222C.

Classification of Scheduling Problems



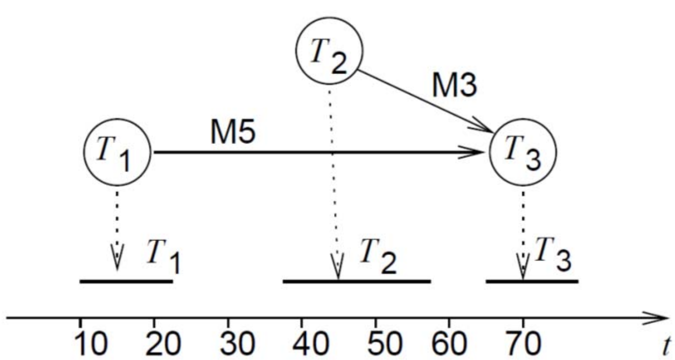
Overview

Scheduling of aperiodic tasks with real-time constraints:
Table with some known algorithms:

	Equal arrival times; non-preemptive	Arbitrary arrival times; preemptive
Independent tasks	EDD (Jackson)	EDF (Horn)
Dependent tasks	LDF (Lawler), ASAP, ALAP, LS, FDS	EDF* (Chetto)

Scheduling with precedence constraints

Task graph and possible schedule:

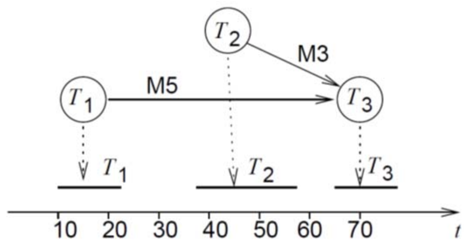


Simultaneous Arrival Times: The Latest Deadline First (LDF) Algorithm

LDF [Lawler, 1973]: reads the task graph and among the tasks with no successors inserts the one with the latest deadline into a queue. It then repeats this process, putting tasks whose successor have all been selected into the queue.

At run-time, the tasks are executed in the **opposite** of the generated total order.

LDF is non-preemptive and is optimal for mono-processors.



If no local deadlines exist, LDF performs just a topological sort.

Asynchronous Arrival Times: Modified EDF Algorithm

This case can be handled with a modified EDF algorithm. The key idea is to transform the problem from a given set of dependent tasks into a set of independent tasks with different timing parameters [Chetto90].
This algorithm is optimal for mono-processor systems.

If preemption is not allowed, the heuristic algorithm developed by Stankovic and Ramamritham can be used.

Static Scheduling with Dependencies: Scheduling in High-Level Synthesis

- HLS-based scheduling
  - ASAP
  - ALAP
  - List scheduling (LS)
  - *Force-directed scheduling* (FDS)

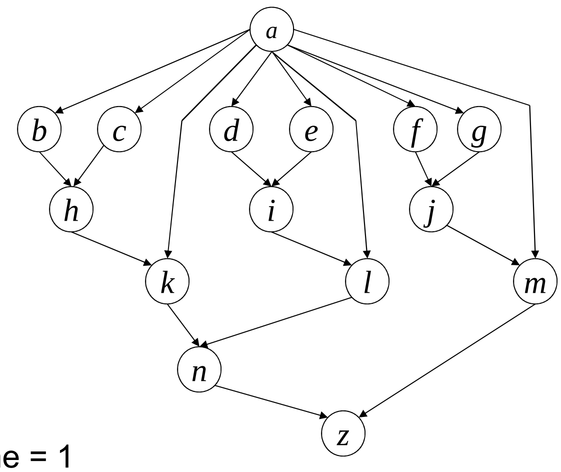
Dependent tasks

The problem of deciding whether or not a schedule exists for a set of dependent tasks and a given deadline is NP-complete in general [Garey/Johnson].

Strategies:

1. Add resources, so that scheduling becomes easier
2. Split problem into static and dynamic part so that only a minimum of decisions need to be taken at run-time.
3. Use scheduling algorithms from high-level synthesis

Task graph



Assumption:
execution time = 1
for all tasks

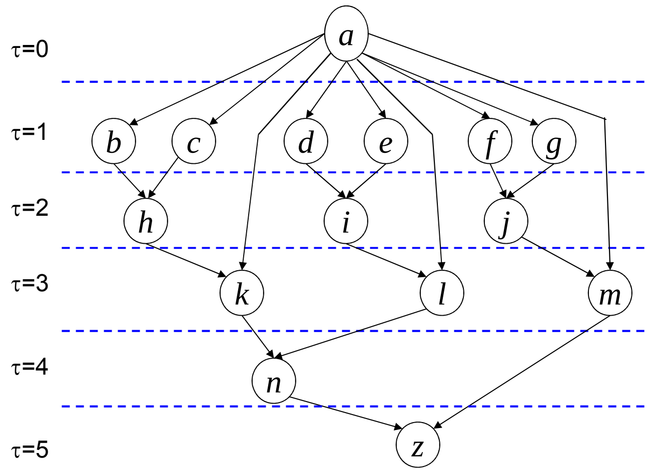
As soon as possible (ASAP) scheduling

ASAP: All tasks are scheduled as early as possible

Loop over (integer) time steps:

- Compute the set of unscheduled tasks for which all predecessors have finished their computation
- Schedule these tasks to start at the current time step.

As soon as possible (ASAP) scheduling: Example



As-late-as-possible (ALAP) scheduling

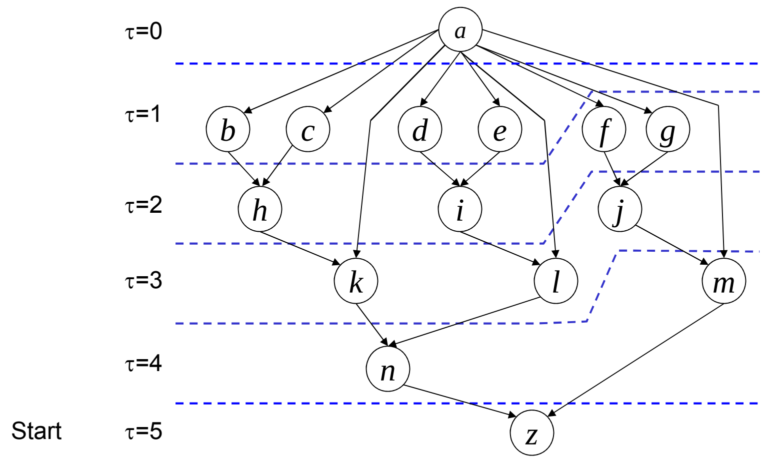
ALAP: All tasks are scheduled as late as possible

Start at last time step*:

Schedule tasks with no successors and tasks for which all successors have already been scheduled.

* Generate a list, starting at its end

As-late-as-possible (ALAP) scheduling: Example



(Resource constrained) List Scheduling

List scheduling: extension of ALAP/ASAP method

Preparation:

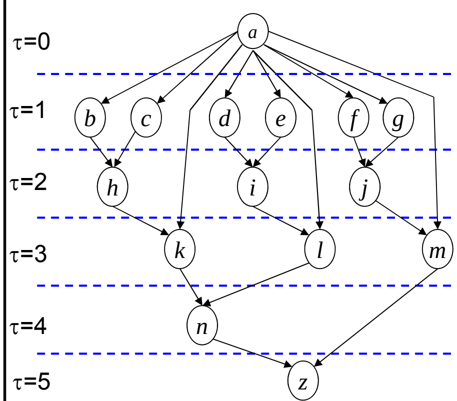
- Topological sort of task graph $G=(V,E)$
- Computation of priority of each task:

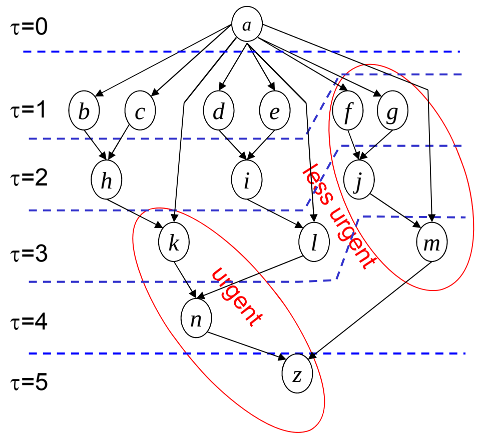
  Possible priorities u:

  - Number of successors
  - Longest path
  - **Mobility** = τ (ALAP schedule)- τ (ASAP schedule)

Mobility as a priority function

Mobility is not very precise





List Scheduling Algorithm

List($G(V,E)$, B, u){
i :=0;
repeat {
Compute set of candidate tasks A_i ;
Compute set of not terminated tasks G_i ;
Select $S_i \subseteq A_i$ of maximum priority r such that
$|S_i| + |G_i| \leq B$ (***resource constraint***)
foreach ($v_j \in S_i$): $\tau(v_j)$:=i; (*set start time*)
$i := i + 1$;
}
until (all nodes are scheduled);
return (τ);
}

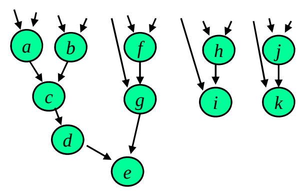
} may be repeated for different task/ processor classes

Complexity: $O(|V|)$

Example

Modified example based on J. Teich

Assuming B =2, unit execution time and u : path length



$u(a)= u(b)=4$
$u(c)= u(f)=3$
$u(d)= u(g)= u(h)= u(j)=2$
$u(e)= u(i)= u(k)=1$
$\forall\ i : G_i =0$