Unit-5

Clock Driven Scheduling

→ Clock driven scheduling is applicable to deterministic systems.
→ The clock driven schedulers make the scheduling decisions at a specific time independent of events, such as job release, and completions.

Notations and Assumptions

Assumptions

1. The clock driven scheduling is applicable to deterministic system.
2. There are n periodic tasks in the system. As long as the system stays in a operation mode, n is fixed.
3. All parameters of the periodic tasks are known before execution: each job in T_i is released P_i units of time after the previous job.
4. Each job $J_{i,k}$ is ready for execution at its release time $\phi_{i,k}$.
5. Aperiodic jobs may exist.
   - Aperiodic jobs released at unexpected time instants.
   - The system maintains a single queue for aperiodic jobs. When an aperiodic job is released, it is placed in the queue without need to notify scheduler.
   - When processor is available aperiodic jobs are scheduled.
6. There are no sporadic jobs.

Notations

A periodic task T_i is represented by the 4-tuple notation
$T_i = (\phi_i, P_i, e_i, D_i)$.

where,
ϕ_i = phase, P_i = period, e_i = execution time and D_i = relative deadline
- Default phase of T_i is $\phi = 0$, default relative deadline is the period $D_i = P_i$.
- Omit elements of the tuple that have default values.

Example:
$T_1 = (1, 10, 3, 6)$; here $\phi_1 = 1$, $P_1 = 10$, $e_1 = 3$, $D_1 = 6$
i.e. $J_{1,1}$ released at 1, and deadline : 7
$J_{1,2}$ released at 11, and deadline = 17
- The utilization of this task is 0.3.

$T_2 = (10, 3, 6) \Rightarrow \phi_2 = 0$, $P_2 = 10$, $e_2 = 3$, $D_2 = 6$
$J_{2,1}$ released at 0, deadline 6
$J_{1,2}$ released at 10, deadline 16.

$T_3 = (10, 3) \Rightarrow \phi_3 = 0$, $P_3 = 10$, $e_2 = 3$, $D_2 = 10$
$J_{3,1}$ released at 0, deadline 10
$J_{1,2}$ released at 10, deadline 20

Static, Clock-Driven scheduler

When the parameter of job with hard deadline are known before the system begins the execution then the static schedules of jobs can be developed at offline and processor time allocated to a jobs is equal to its maximum execution time. The scheduler dispatches the jobs according to the static schedule and repeats the jobs in each hyper-periods. The static schedule guarantees that each job completes by its deadline and no job overrun can occur.

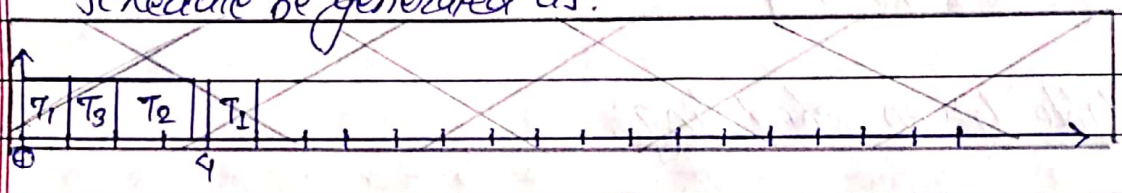
Example:

Four independent periodic tasks: $T_1 = (4, 1)$, $T_2 = (5, 1.8)$, $T_3 = (20, 1)$, $T_4 = (20, 2)$

Total utilization = $1/4 + 1.8/5 + 1/20 + 2/20 = 0.76$

Hyperperiod = LCM (4, 5, 20, 20) = 20

According to this specification, ~~static schedule be constructed as:~~ schedule be generated as:



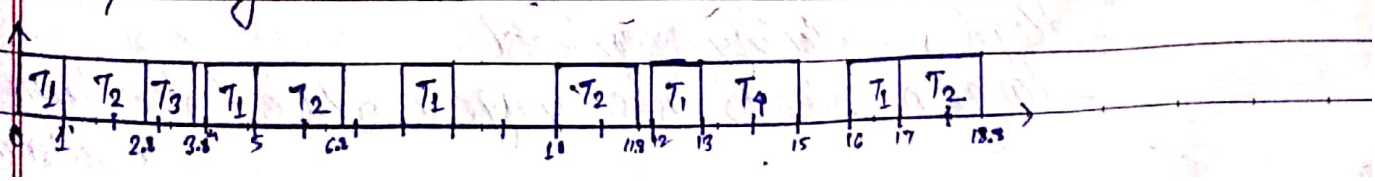
k	t_k	$T(t_k)$
0	0	T_1
1	1	T_2
2	2.8	T_3
3	3.8	I
4	4	T_1
5	5	T_2
6	6.8	I
7	8	T_1
8	9	I
9	10	T_2
10	11.8	I
11	12	T_1
12	13	T_4
13	15	I
14	16	T_1
15	17	T_2

t_k : an instant at which scheduling decision is made.

$T(t_k)$: name of the task whose job should start at t_k or I (idle)

Fig: offline computed schedule table

Corresponding static schedule be constructed as:



Some intervals are not used by periodic task called as slack time. slack time can be used to execute aperiodic jobs.

Types of clock Driven schedules:

- Table driven
- cyclic schedules

Table driven scheduling:

- It stores pre-computed schedule as a table which is called schedule table. Schedule table contains details of which task would run when.
- Every element of the schedule table specifies decision time, and scheduled task.
- Each entry $(t_k, T(t_k))$ in this table gives a decision time t_k, which is an instant at which scheduling decision is made, and $T(t_k)$, which is either the name of the task whose job should start at t_k or I.

$$T(t_k) = \begin{cases} T_i & \text{if } T_i \text{ is to be scheduled at time } t_k \\ I & \text{If no periodic task is scheduled at time } t_k. \end{cases}$$

E.g. → Previous Eg. → table (cont.)

Cyclic schedule

→ Problems with arbitrary table-driven cyclic schedules:
- Scheduling decisions are made at random time instants
- High scheduling overhead
- Large number of tasks → huge schedule table → large memory requirement.

→ A cyclic scheduler repeats a precomputed schedule. The precomputed schedule need to be stored for only one major cycle. The major cycle is divided into one or more minor cycle known as frames. Intervals between 2 consecutive decisions are frames.
E.g.
The major cycle has been divided into four minor cycles (frames

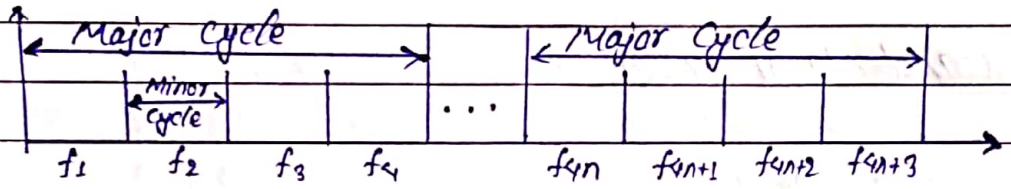


Fig: Major & Minor cycles in a Cyclic scheduler

→ The scheduling points of a cyclic scheduler occur at frame boundaries i.e. a task can begin its execution at the start of a frame. If a task arrives in between a frame then, its execution will begin at the start of the next frame.
→ Every frame has length f; f is called the frame size.
→ There is no preemption within each frame because scheduling decisions are made only at the begining of every frame.
→ The assignment of tasks to frames is stored in a schedule table.

Task Number	Frame Number
T_3	f_1
T_1	f_2
T_3	f_3
T_4	f_4

Fig: An example schedule table for a structured cyclic scheduler

Advantages:
- Scheduler can easily check for overruns and missed deadlines at the end of each frame.
- Can use a periodic clock interrupt, rather than programmable timer.

General structure of cyclic schedules

It consists of following characteristics:
- Frames and major cycles
- Frame size constraints
- Job slices

Frames and Major cycles

→ See previous page.

{A major cycle of a set of tasks is an interval of time on the time line such that in each major cycle, the different tasks reoccur identically}

Frame size constraints

Problem: How to choose frame length?

This constraint explains how to choose frame length in cyclic scheduling. A selected frame size should satisfy the following three constraints.

1. To avoid the preemption of the job, we make frame size f larger than the execution time e_i of each task T_i i.e.

$$f \geq \max(e_1, e_2, \ldots, e_n) \quad \text{——} \quad (i)$$

2. To minimize the number of entries in the cyclic schedule (in order to have the table small), the hyper-period should be an integral multiple of the frame size (f) i.e. f divides the period P_i of at least one task T_i.

$$\exists i : \text{mod}(P_i, f) = 0 \quad \text{——} \quad (ii)$$

3. To allow scheduler to check that jobs complete by their deadline, should be at least one frame between the release time and deadline of every job.

$2f - \gcd(P_i, f) \leq D_i$ for all $i = 1, 2, \ldots n$.

Examples:

1) A cyclic scheduler is to be used to run the following set of periodic task on a uniprocessor. $T_1 = (4, 1)$, $T_2 = (5, 1.8)$, $T_3 = (20, 1)$, $T_4 = (20, 2)$. Find all possible frame size.

Sol

For the given task set, an appropriate frame size is that one that satisfies all the three required constraints. In the following, we determine a suitable frame size f which satisfies all the three required constraints.

Constraint 1:

Let f be an appropriate frame size, then $f \geq \max(e_i)$

$f \geq \max(1, 1.8, 2)$

From this constraint, we get $f \geq 2$.

Constraint 2:

The major cycle / Hyperperiod H for the given task set is given by $H = LCM(4, 5, 20) = 20$.

According to this constraint, H should be an integral multiple of the frame size f i.e. $H \bmod f = 0$.

So, possible choices for f are 2, 4, 5, 10 and 20.

Hence $f \in \{2, 4, 5, 10, 20\}$

Constraint 3:

To satisfy this constraint, we need to check whether

A ~~frame~~ selected frame size f satisfies the inequality:

$2f - gcd(P_i, f) \leq D_i$ for each $i = 1, 2, 3, \ldots$

So;

$2f - gcd(4, f) \leq 4$ (T_1)

$2f - gcd(5, f) \leq 5$ (T_2)

$2f - gcd(20, f) \leq 20$ (T_3, T_4)

For $f = 2$ & Task T_1

$2*2 - gcd(4, 2) \leq 4$

$\Rightarrow 4 - 2 \leq 4$

$\Rightarrow 2 \leq 4$ (T)

For $f = 2$ & Task T_2

$2*2 - gcd(5, 2) \leq 5$

$\Rightarrow 4 - 1 \leq 5$

$\Rightarrow 3 \leq 5$ (T)

For $f = 2$ & Task T_3 & T_4

$2*2 - gcd(20, 2) \leq 20$

$\Rightarrow 4 - 2 \leq 20$

$\Rightarrow 2 \leq 20$ (T)

So, frame size 2 satisfies all the 3 constraints. Hence it is a feasible frame size.

For $f = 4$ & Task T_1

$2*4 - gcd(4, 4) \leq 4$

$\Rightarrow 8 - 4 \leq 4$

$\Rightarrow 4 \leq 4$ (T)

For $f = 4$ & Task T_2

$2*4 - gcd(5, 4) \leq 5$

$\Rightarrow 8 - 1 \leq 5$

$\Rightarrow 7 \leq 5$ (F)

So, frame size 4 is not a feasible frame size.

~~For f = 4 & Task T_3 & T_4~~

~~$2*4 - gcd(20, 4) \leq 20$~~

~~$\Rightarrow 8 -$~~

for $f = 5$ & Task T_1

$2*5 - gcd(4, 5) \leq 4$

$\Rightarrow 10 - 1 \leq 4$

$\Rightarrow 9 \leq 4$ (F) So, frame size 5 is not a feasible frame size.

For $f = 10$ & Task T_1

$2 * 10 - \gcd(4, 10) \leq 4$

$\Rightarrow 20 - 2 \leq 4$

$\Rightarrow 18 \leq 4$ (F)

So, frame size 10 is not a feasible frame size.

For $f = 20$ & Task T_1

$2 \times 20 - \gcd(4, 20) \leq 4$

$\Rightarrow 40 - 4 \leq 4$

$\Rightarrow 36 \leq 4$ (F)

So, frame size 20 is not a feasible frame size.

∴ Appropriate frame size $f = 2$.

2) $T_1 = (6, 1)$, $T_2 = (10, 2)$ & $T_3 = (18, 2)$

Constraint 1:

$f \geq \max(e_i)$

$f \geq \max(1, 2, 2)$

$f \geq 2$

Constraint 2:

$H = LCM(6, 10, 18) = 90$

$\therefore f \in \{2, 3, 5, 6, 9, 10, 18\}$

Constraint 3:

$2f - \gcd(P_i, f) \leq D_i$ for all $1 \leq i \leq n$

$f = 2$

$2 * 2 - \gcd(6, 2) = 4 - 2 = 2 \leq 6$ (T)

$2 * 2 - \gcd(10, 2) = 4 - 2 = 2 \leq 10$ (T)

$2*2 - gcd(18,2) = 4-2 = 2 \leq 18$ (T)

For $f=3$

$2*3 - gcd(6,3) = 6-3 = 3 \leq 6$ (T)

$2*3 - gcd(10,3) = 6-1 = 5 \leq 10$ (T)

$2*3 - gcd(18,3) = 6-3 = 3 \leq 18$ (T)

For $f=5$

$2*5 - gcd(6,5) = 10-1 = 9 \not\leq 6$ (F)

For $f=6$

$2*6 - gcd(6,6) = 12-6 = 6 \leq 6$ (T)

$2*6 - gcd(10,6) = 12-2 = 10 \leq 10$ (T)

$2*6 - gcd(18,6) = 12-6 = 6 \leq 18$ (T)

For $f=9$

$2*9 - gcd(6,9) = 18-3 = 15 \leq 6$ (F)

For $f=10$

$2*10 - gcd(6,10) = 20-2 = 18 \leq 6$ (F)

For $f=18$

$2*18 - gcd(6,18) = 36-6 = 30 \leq 6$ (F)

Here, frame size 2, 3 & 6 satisfies all 3 constraints.

∴ Appropriate frame size = 2, 3 & 6.

Job slices

Sometimes, a system cannot meet all three frame size constraints simultaneously. At this situation we can often solve by partitioning a job with large execution time into slices (sub-jobs) with shorter execution times/ deadlines then these slices are called job slices. This gives the effect of preempting the large jobs, so allow other jobs to run.

- Sometimes, it is necessary to partition jobs into more slices than required by the frame size constraints, to yield a feasible schedule.

Example:

Consider a system with $T_1 = (4, 1)$, $T_2 = (5, 2, 7)$, $T_3 = (20, 5)$

Constraint 1 → $f \geq 5$

Constraint 2 → $f \in \{2, 4, 5, 10, 20\}$

Constraint 3 → $2f - \gcd(P_i, f) \leq 4, \leq 7, \leq 20$

From constraint 1, we must have $f \geq 5$ but to satisfy constraint 3 we must have $f \leq 4$.

- solve by splitting T_3 into $T_{3,1} = (20, 1)$, $T_{3,2} = (20, 3)$ and $T_{3,3} = (20, 1)$
  - other possible splits exist: pick based on application domain knowledge.
- The resultant system has five tasks for which we can choose the frame size $f = 4$. The figure shows a cyclic schedule for these tasks.

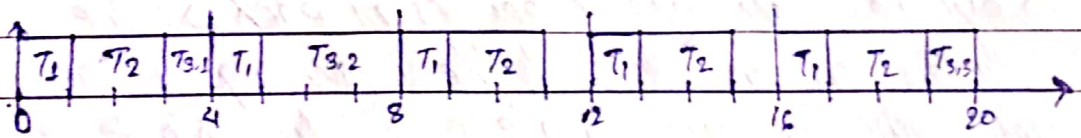


Fig: A preemptive cyclic schedule of $T_1 = (4, 1)$, $T_2 = (5, 2, 7)$ & $T_3 = (20, 5)$

→ To construct a cyclic schedule, we need to make three kind of design decisions:

- choose a frame size based on constraints
- Partition jobs into slices
- Place slices in frame

Cyclic Executive

Cyclic executive is the modification of table-driven cyclic scheduler to be frame based. It makes scheduling decisions only at the begining of each frame and deterministically interleaves the execution of periodic tasks. It allows aperiodic and sporadic jobs to use the time that are not used by periodic tasks.

→ Schedule is computed offline, and stored in the table. The stored table that gives the precomputed cyclic schedule has F entries.

Where. F is the number of frames per major cycle

i.e. $F = H/f$

- Each entry (say the k^{th}) lists the names of the job slices that are scheduled to execute in frame k. The entry is denoted by $L(k)$ and is called a scheduling block or simply a block. The current block refers to the list of periodic job slices that are scheduled in the current frame.

→ The cyclic executive executes at each of the clock interrupts, which occur at the begining of frames:
- Determines the appropriate scheduling block for this frame
- Executes the jobs in the scheduling block in order.
- starts job at head of the aperiodic job queue running for remainder of frame.

→ less overhead than pure table driven cyclic scheduler, since only interrupted on frame boundries, rather than on each job.

Improving the Average Response Time of Aperiodic Jobs

Aperiodic jobs are scheduled in the background after all jobs with hard deadlines scheduled in each frame have completed the slices of periodic task in each frame are completed, it causes average response time is long.

→ Average response time for aperiodic jobs can be improved by scheduling hard-real time jobs as late as possible without missing the deadline.

* Slack stealing → in deadline driven system → Way to schedule aperiodic jobs with clock driven scheduler e.g.

A natural way to improve the response time of aperiodic jobs is by executing the aperiodic jobs ahead of the periodic jobs whenever possible. This approach, called slack stealing.

→ slack time refers to the maximum time that a job can be delayed and still meet its deadline.

For the slack-stealing scheme to work, every periodic job slice must be schedule in a frame that ends no later than its deadline. When aperiodic job executes ahead of slice of periodic task then it consumes the slack in the frame. It reduces the response time of aperiodic jobs but requires accurate timer.

Let f be the frame size, and the total amount of time allocated to all the slices/periodic jobs scheduled in frame k be x_k. The slack time available in the frame is equal to $f - x_k$ at the beginning of the frame.

After y units of slack time are used by aperiodic jobs, the available slack is reduced to $f - x_k - y$. The cyclic executive can let aperiodic jobs execute in frame k as long as there is slack, i.e. the available slack $f - x_k - y$ in the frame is larger than 0.

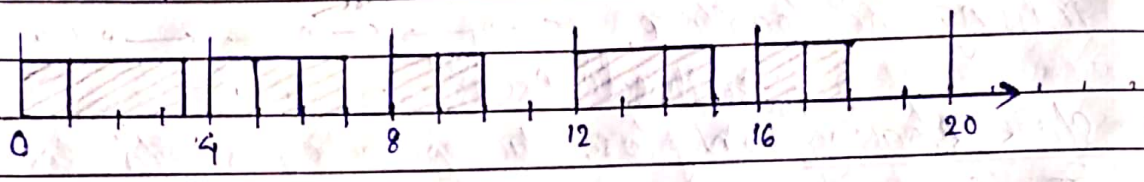
→ At the beginning of each frame, the cyclic executive sets the timer to the value of the initial slack in the frame.
- Whenever an aperiodic job executes slack is reduced.
- when the timer expires, indicating that there is no more slack, the cyclic executive preempts the executing aperiodic job & lets the execution of the next periodic job.

(os exam unit-3)

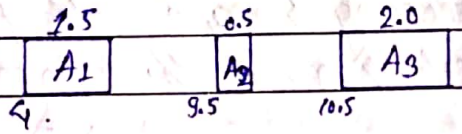
For example:

Consider a system with
- Three periodic tasks $T_1(4,1)$, $T_2(5,2)$ & $T_3(8,10,1)$
- Three aperiodic tasks $A_1(4,1.5)$, $A_2(9.5,0.5)$ & $A_3(10.5,2)$

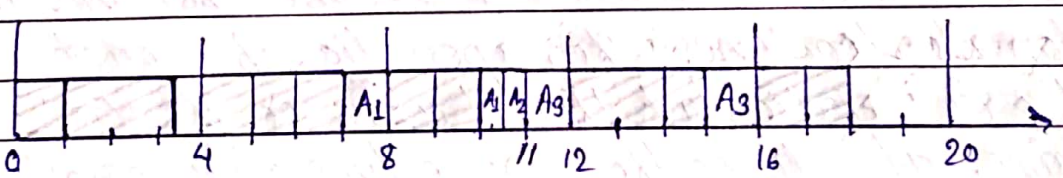
Hence major cycle in the cyclic schedule of the periodic tasks is:



3 aperiodic jobs:



When aperiodic job executes in use of the cyclic executive (no slack stealing)

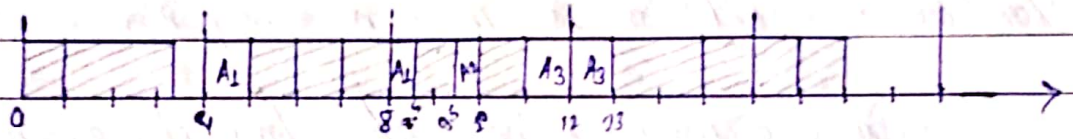


- Job A_1: released at $t=4$, starts at $t=7$, completes at $t=10.5$;
  response time $= 10.5-4 = 6.5$
- Job A_2: released at $t=9.5$, starts at $t=10.5$, completes at $t=11$;
  response time $= 1.5$
- Job A_3: released at $t=10.5$, starts at $t=11$, completes at $t=16$
  response time $= 5.5$

$\therefore$ Average response time $= 4.5$

If the cyclic executive does slack stealing such that
- At time between 4 & 8 there is the slack of 1 such that A1 can be executed at time 4 & preempted at 5.
- At time bet^n 8 to 12 there exists the slack of 2 such that A1 can resumed upto 8.5 and A2 can executed between 9.5 to 10. Similarly A3 can executed between 11 to 12 & preempted.
- At time 12 & 16 there exists the slack of 1 and A3 can resumed on 12 and executed at 13.

Now the schedule be generated as



∴ Average response time = 2.5

This shows that the average response time is improved if slack stealing approach is used.

Scheduling of sporadic Jobs

Sporadic jobs have hard deadlines, and their release and execution times are not known a priori. Hence, a clock-driven scheduler cannot guarantee a priori that sporadic jobs complete in time.

Scheduling of sporadic jobs is done in two steps:

1. Acceptance test
2. EDF scheduling of accepted sporadic jobs.

Acceptance Test

During an acceptance test, the scheduler checks whether

the newly released sporadic job can be feasibly scheduled with all the jobs in the system at the time. Here all jobs means either a periodic job or other sporadic jobs previously scheduled but not yet completed.

If there is a sufficient slack time in the frames before the new job's deadline, the scheduler accepts and schedules the new sporadic job. Otherwise, the scheduler rejects the new sporadic job.

→ If more than one sporadic jobs arrives at once then these should be queued for acceptance in EDF order.

Suppose an acceptance test is done at the begining of frame t on a sporadic job $S(d, e)$. The deadline d of S is in frame $l+1$. The frame l ends before d but frame $l+1$ ends after d and $l \geq t$. The job must be scheduled in the l^{th} or earlier frames. The job can complete in time only if the current (total) amount of slack time $\sigma_c(t, l)$ in frames t, $t+1$, ..., l is equal to or greater than its execution time e. The scheduler should reject S if $e > \sigma_c(t, l)$.

↳ The scheduler accepts the new job $S(d, e)$ only if $e \leq \sigma_c(t, l)$ and no sporadic jobs in system are adversely affected.

EDF scheduling of the Accepted Jobs

The EDF algorithm is used to schedule accepted sporadic jobs. For this purpose, the scheduler maintains a queue of accepted sporadic jobs in non-decreasing order of their deadlines and inserts each newly accepted sporadic job into this queue in this order.

- Whenever all the slices of periodic tasks in each frame

are completed, the cyclic executive lets the jobs in the sporadic job queue execute in the order they appear in the queue.

Consider the following example:

- Frame size is 4. The shaded boxes show where periodic tasks are scheduled.
- S_1 - S_4 are sporadic tasks with parameters (d_i, e_i)

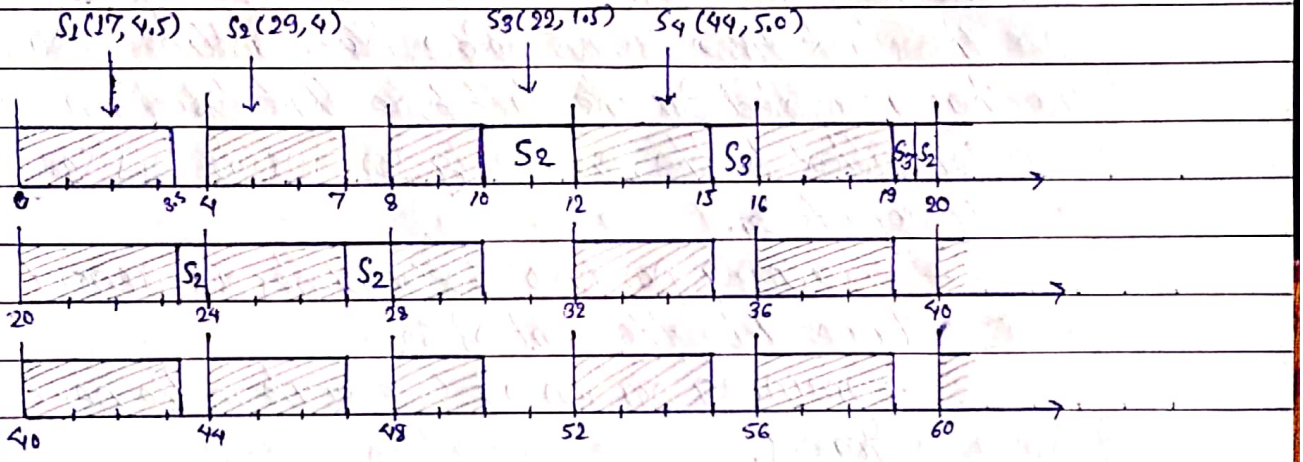


fig: Example of scheduling sporadic jobs.

→ S_1 is released in time 3.

- S_1 must be scheduled in frames 2, 3 & 4.
- The acceptance test is done at time 4, i.e. at the begining of frame 2. And total amount of slack time is 4 which is less than execution time - job is rejected.

→ S_2 is released in time 5.

- Must be scheduled in frames 3 to 7.
- Acceptance test - at the begining of frame 3. The slack time is 5.5 which is greater than execution time. Therefore scheduler accepts this jobs & the first part of S_2 with $e=2$ executes in frame 3 (current frame).

→ S_3 & is released in time 21.

- Must be scheduled in frames 4 & 5.

- The acceptance test is done at the begining of frame 4. The slack time is 2 (enough for S_3 & the part of S_2) - Job is accepted, where S_3 executes in current frame, followed by second part of S_2 because the deadline of S_2 is later than S_3.

→ S_4 is released in time 14.

- The acceptance test is done at the begining of frame 5. The slack time is 4.5 before the deadline of S_4, after it has accounted for the slack time that has already been committed to the remaining portions of S_2 & S_3. Therefore, it rejects S_4.
  - Remaining portion of S_3 completes in current frame followed by the part of S_2
  - Remaining portion of S_2 execute in the next two frames.

Practical considerations

Handling overruns

How to handle frame overruns?

→ Jobs are scheduled based on maximum execution time, but failures might cause overrun.

→ A robust system will handle this by either:

1) killing the job and starting an error recovery task; or
2) Preempting the job and scheduling the remainder as an aperiodic job.

→ Depends on usefulness of late results, dependencies bet^n jobs etc.

Mode changes

How to do mode changes?

→ The ~~parame~~ number n of periodic tasks in the system and their parameters remain constant as long as the system stays in the same (operation) mode.
→ During a mode change, the system is reconfigured.
→ Some periodic tasks are deleted from the system because they will not execute in the new mode.
→ Periodic tasks that execute in the new mode but not in the old mode are created and added to the system.
→ The periodic tasks that execute in both modes continue to execute in a timely fashion.
→ When the mode change completes, the new set of periodic tasks are scheduled and executed.
→ The mode change job has either a soft or hard deadline.

Aperiodic mode change:
- Since the deadlines of the remaining aperiodic jobs are soft, their execution can be delayed until after the mode changes.
- It can lengthen the response time of the mode change.

Sporadic mode change:
- A sporadic mode change has to be completed by a hard deadline.

Multiprocessor scheduling

How to schedule task on multiprocessor system?

→ It is straightforward to schedule tasks on several processors whenever the workload parameters are known a priori and there is a global clock.
→ We can construct a global schedule which specifies on what processor each job execute & when the job executes.

$D_i >= P_i$

Algorithm for constructing static schedules: The Network flow Algorithm

→ A system of independent preemptable periodic tasks whose relative deadline are ~~equal to or~~ not less than their respective periods is schedulable iff the total utilization of the tasks is ≤ 1.

→ Some tasks may have relative deadlines shorter than their periods but a feasible schedule may not exist due to frame size constraint even when $U \leq 1$. Therefore an iterative algorithm is used to find the feasible ~~schedule~~ cyclic schedule if one exists. The algorithm is called the iterative network flow algorithm or the INF algorithm.

It assumes that the task can be preempted at any time and are independent.

The INF algorithm is performed in 2 steps:

step 1: Find all the possible frame sizes of the system that meet the frame size constraints 2 and 3 but not necessarily constraint 1.

step 2: Construct the network flow graph with largest possible frame size.

The INF algorithm iteratively tries to find a feasible cyclic schedule of the system for a possible frame size at a time, starting with the largest value.

Network-Flow Graph

The network flow graph of preemptive scheduling jobs is used by INF. To use this algorithm

- We ignore the tasks to which the job belong.

- We name the jobs to be scheduled in a major cycle of F frame $J_1, J_2, \ldots, J_N$.

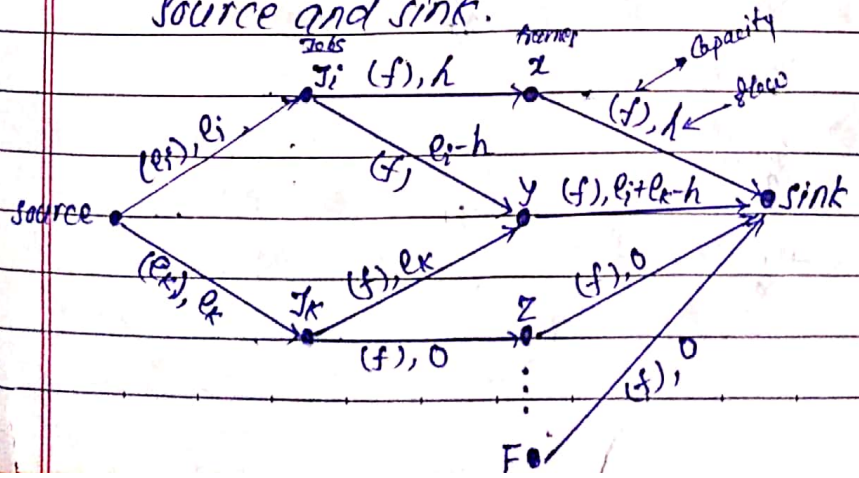
The constraints on which the jobs can be scheduled are represented by the network flow graph of the system.

The graph contains the following vertices and edges:

1. A job vertex J_i, $i = 1, 2, \ldots N$ represents each job.
2. A frame vertex j, $j = 1, \ldots, F$ represents each frame in the major cycle.
3. There are two special vertices named source and sink.
4. There is a directed edge (J_i, j) from a job vertex J_i to a frame vertex j if the job J_i can be scheduled in the frame j, and the capacity of the edge is the frame size f i.e. frame j satisfies release-time and deadline constraints of J_i.
5. There is a directed edge from the source vertex to every job vertex J_i, and the capacity of this edge is the execution time e_i of the job.
6. There is a directed edge from every frame vertex to the sink, and the capacity of this edge is f.

→ A flow of an edge is a nonnegative number that satisfies the following constraints:

- It is no greater than the capacity of the edge.
- The sum of flows of all the edges into every vertex is equal to the sum of the flows of all the edges out of the vertex except source and sink.



The graph indicates that job J_i can be scheduled in frame x & y and the job J_k can be scheduled in frame y and z.

Example

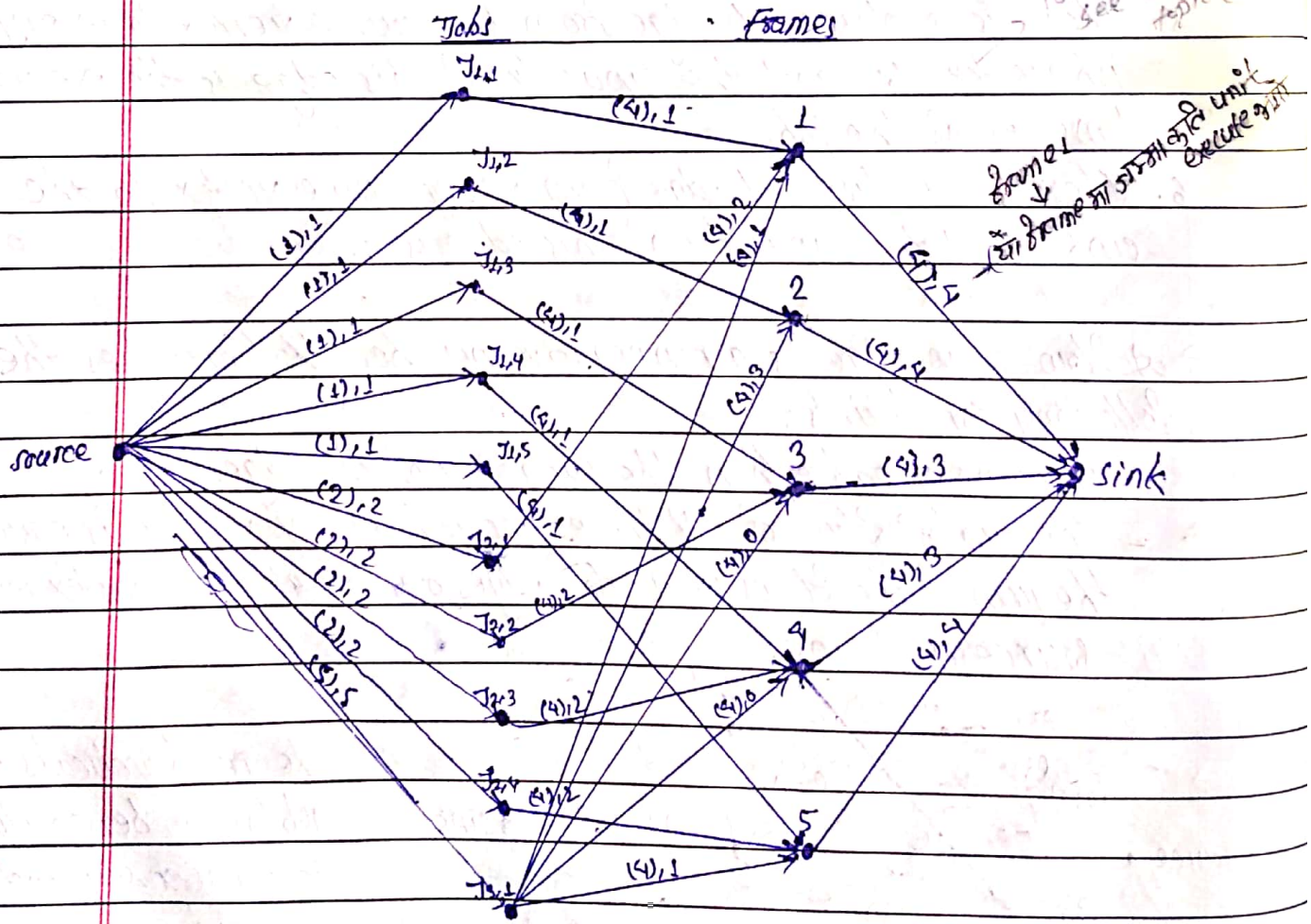
$T_1(4,1)$, $T_2 = (5, 2, 7)$ and $T_3 = (20, 5)$

$H = 20$, $U = 0.9$

The possible frame sizes are 4 and 2. (Frame sizes 2 and 4 meet constraints 2 and 3 but not 1).

- Try first frame size 4: In H there are:
  - 5 frames,
  - 5 jobs of T_1, 4 jobs of T_2, 1 of T_3

Each job is scheduled in a frame that:
- begins no sooner than its release time,
- ends no later than its deadline.



with $f = 4$

maximum flow is $18 = \sum e_i$

$\Rightarrow$ feasible schedule.

→ The flows of the feasible schedule indicate that T_3 is to be partitioned in 3 slices and give their size.

Pros and Cons of clock driven scheduling

Advantages:

1. Conceptual simplicity
   - It defines the ability to consider complex dependencies, communication delays, and resource contention among jobs when constructing the static schedule; guaranteeing absence of deadlocks and unpredictable delays.
   - The entire schedule is captured in a static table.
   - Different operating modes can be represented by different table.
   - No concurrency control or synchronization is required.
   - If completion time jitter requirement exists, can be captured in the schedule.
2. When workload is mostly periodic and the schedule is cyclic, timing constraints can be checked and enforced at each frame boundary.
3. The choice of frame size can minimize context switching and communication overheads.
4. Clock driven scheduling is relatively easy to validate, test and certify.

Date
Page

Disadvantages:

1. Inflexible:
   - Pre-compilation of knowledge into scheduling tables means that if anything changes materially, have to redo the table generation.
   - Best suited for systems which are rarely modified once built.
2. other disadvantages:
   - Release time of all jobs must be fixed.
   - All possible combination of periodic tasks that can execute at the same time must be known a priori, so that the combined schedule can be pre-computed.
   - The treatment of aperiodic jobs is very primitive.

Jayanta Poudel.