Outline

• List Scheduling Concepts and Proof
• Task with Hard Deadlines
• Real Time Tasks
• Energy Efficiency
• Energy Efficient Scheduling
• Reliability Aware Scheduling
\[ P_m \parallel p_j \parallel C_{\text{max}} \]

**Minimum makespan scheduling**

- \( P_m \parallel p_j \parallel C_{\text{max}} \) in NPC
- Given processing times for \( n \) jobs, \( p_1, p_2, \ldots, p_n \), and an integer \( m \)
- Find an assignment of the jobs to \( m \) identical machines
- So that the completion time, also called the makespan, is minimized.
Minimum makespan scheduling: Arbitrary List

• List Scheduling : Approximation

• Algorithm
  – 1. Order the jobs arbitrarily.
  – 2. Schedule jobs on machines in this order, scheduling the next job on the machine that has been assigned the least amount of work so far.

• Above algorithm achieves an approximation guarantee of 2
Load Balancing: List Scheduling

Machine 1

Machine 2

Machine 3

0

Time

A Sahu
Load Balancing: List Scheduling
Load Balancing: List Scheduling

Optimal Schedule

List schedule
LS is 2 APPRX
Algorithm: **List scheduling**

Basic idea: In a list of jobs, schedule the next one as soon as a machine is free.

```
(a)  b   c   d  
machine 1 2 3 4
```

**LS : 2 APPRX**

Good or bad?
List Scheduling is “2-approximation” (Graham, 1966)

Algorithm: List scheduling
Basic idea: In a list of jobs,
    schedule the next one as soon as a machine is free

job f finishes last, at time A

compare to time OPT of best schedule: how?
List Scheduling is “2-approximation”

job f finishes last, at time A

compare to time OPT of best schedule: how?

(1) job f must be scheduled in the best schedule at some time:
   \[ f \leq \text{OPT}. \quad \Rightarrow \quad A - S \leq \text{OPT}. \]

(2) up to time S, all machines were busy all the time, and OPT cannot beat that, and job f was not yet included: \( S < \text{OPT}. \)

(3) both together: \( A = A - S + S < 2 \, \text{OPT}. \)

“2-approximation” (Graham, 1966)
LS is \((2-1/m)\) APPRX
LS achieves a perf. ratio $2-1/m$.

- **Proof:**
  - Let $T = \sum t_i$, $i=1,2...,n$, the sum of all processing times to be accommodated.
  - We know that the total processing time available in an optimal schedule on the machines is $mT^*$.
  - So, $T^* \geq T/m$.
  - Moreover, $T^* \geq t_k$ for every $k$. 
LS achieves a perf. ratio $2 - 1/m$.

**Proof:**

– Let $T = \sum t_i$, $i=1,2...,n$

– The total processing time available in an optimal schedule on the machines is $m \cdot T^*$. So, $T^* \geq T/m$.

– Moreover, $T^* \geq t_k$ for every $k$.

– Let $A$ be the makespan of the schedule produced by LS.

– By definition there must be a job $k$, with processing time $t_k$, that ends at the makespan time.

– No machine can end its operation before $A - t_k$, because then job $k$ would have been scheduled on that machine, thus reducing the makespan.
LS achieves a perf. ratio \(2-1/m\).

So all machines are busy from time 0 through \(A-t_k\).
Consequently,

\[
T-t_k \geq m(A-t_k) \implies T-t_k \geq mA - mt_k
\]

\[
\implies T-t_k + mt_k \geq mA \implies T + (m-1)t_k \geq mA
\]

So, \(A \leq T/m + t_k (m-1)/m\)

\[
\leq T^* + (1-1/m) T^* = (2-1/m) T^*
\]

\[
A \leq (2-1/m) T^*
\]
Example: Worst Case

makespan: m+1

makespan: 2m

m x m

m x 1

m x 1

A Sahu
List with LPT

• List scheduling can do badly if long jobs at the end of the list spoil an even division of processing times.

• We now assume that the jobs are all given ahead of time, i.e. the LPT rule works only in the off-line situation. Consider the “Largest Processing Time first” or LPT rule that works as follows.
LPT Algorithm

1 sort the jobs in order of decreasing processing times: $t_1 \geq t_2 \geq \ldots \geq t_n$

2 execute list scheduling on the sorted list

3 return the schedule so obtained.

- The LPT rule achieves $3/2$-Approx

- The LPT rule achieves a performance ratio $4/3 - 1/(3m)$. Prove out of Syllabus
LPT 3/2-Approx: Jobs are sorted

- Job Time: \( t_1 \geq t_2 \geq t_3 \geq \ldots \geq t_j \)
- Suppose \( j (=m+1) \) jobs (\( j > m \)), in LPT \( T^* \geq 2.t_{m+1} \)
- Suppose a machine \( M_i \) have at least two jobs and \( t_j \) be last job (\( j \geq m+1 \)) assigned to \( M_i \)
  \[ t_j \leq t_{m+1} \leq T^*/2 \]
- Also we have \( t_j \leq T^* \) and \( T_i - t_j \leq T^* \), where \( T_i \) is sum of ET of task assigned to \( M_i \)
- \( T_i - t_j \leq T^* \Rightarrow T_i \leq T^* + t_j \Rightarrow T_i \leq T^* + T^*/2 \)
  \[ T_i \leq (3/2) T^* \]
Review $P|p_j=1|\Sigma w_jU_j$

- Sorting task based on $d_i$ and $d_1 \leq d_2 \leq \ldots \leq d_n$
- Simply scheduling and by rejected unfit task will not minimize $w_i$
  - not work : you need to take care of weight
- Sorting task based on $w_i/d_i$
  - Gives priority of task with higher weight but
  - Simply may reject a task based on deadline
  - not work : for optimality
Review $P|p_j=1|\Sigma w_jU_j$

- Sort all the jobs with $d_1 \leq d_2 \leq \ldots \leq d_n$
- Set $S=\emptyset$
- For $i=1$ to $n$ do
  - If ($i_{th}$ task is late when scheduled in the earliest time slot on a machine)
    - Find a task $i^*$ with $w_{i^*} = \min$ weight of tasks in the already scheduled tasks of the set $S$
    - If ($w_{i^*} < w_i$) replace $i^*$ with $i_{th}$ task in the schedule and in $S$.
  - else add $i_{th}$ task to $S$ and schedule the task in the earliest time slot
\( P | | \Sigma U_j \)

- NPC: Sorting based on deadlines is excellent heuristics for most of the case, Experimentally
- But not optimal
- Counter example: \( J(p_j, d_j) \): \( J1(1,1) \), \( J2(1,2) \) and \( J3(3,3.5) \) on two processor
- EDF (J3 misses) but the Optimal
\[ P \mid ptmn \mid \Sigma U_j \]

- In NPC
Review $Q | ptmn | \Sigma C_j$

- LPT on High speed is good to optimize $\Sigma e_j$ the sum of task execution time but not $\Sigma C_j$

- Modified version of SPT (shortest remaining time) rule. As $\Sigma C_j$ include waiting time of all the tasks

- Order the tasks according to non-decreasing processing time.

- Schedule task 1 on available highest speed machine up to time $t_1 = p_1 / s_1$.

- Schedule 2$^{nd}$ task on M2 for $t_1$ time and then on $M_1$ from time $t1$ to time $t_2 \geq t_1$ until it is completed and same process continues
Review Q|ptmn|ΣC<sub>j</sub>

- Example m=3, s1=3, s2=2, s3=1 and n=4, p1=10, p2=8, p3=8, p4=3
- SRT Job J<sub>4</sub> get scheduled on M1 with speed s1 for 1 time unit. Job 3 get scheduled on M<sub>2</sub> upto time 1 and then shifted to M1. Gant chart is given bellow with ΣC<sub>i</sub>=14
Real Time Scheduling
Real Time Scheduling

• MPEG, Audio
  – 30 frame/Sec, 50 f/s, 60f/s

• Can you run 4K MKV file on Mobile?

• Periodic Task

• **Nice Value in Linux**
  – 0-100 for real time task, 101-140 non real time task
  – Size of processor quantum (share) based on nice value
Periodic Task: Real Time Scheduler

• Task with periods
• Each task have to finish before deadline within the period