CS341: Operating System

Scheduling Algorithms

Lect14 : 2nd Sept 2014

Dr. A. Sahu
Dept of Comp. Sc. & Engg.
Indian Institute of Technology Guwahati

1

Outline

- Scheduling Problem Specification
  - α (Mac Envt)| β (Job Envt)| γ (Opt Criteria)
- Algorithm for
  - $1|C_{max},1|\sum C_i,1|\sum w_iC_i,1||\sum T_i,P/Q|ptmn|C_{max}$
  - $1|prec|\sum w_iC_i$,(NPC), $P_m||C_{max}$(NPC)
- Algorithm for $Pm|$ tree, $p_j = 1 \mid C_{max}$
- PTAS for $P_m||C_{max}$(NPC), $Pm|prec,p_j=1|C_{max}$
- Others Scheduling
  - Real Time, Multiprocessor,
  - Distributed and Power Aware

2

Recap

3

Classification of Scheduling Problems

Classes of scheduling problems can be specified in terms of the three-field classification

$$\alpha \mid \beta \mid \gamma$$

where

- α specifies the **machine environment**,
- β specifies the **job characteristics**, and
- γ describes the **objective function(s).**

Machine Environment

- 1 single machine
- P parallel identical machines
- Q uniform machines
- R unrelated machines
- MPM multipurpose machines, J job-shop,
- F flow-shop O open-shop

If the number of machines is fixed to m we write **Pm, Qm, Rm**, MPMm, Jm, Fm, Om.

Example: Machine Environment

	M1	M2	M3
P1	5	5	5
P2	8	8	8
P3	6	6	6

Identical

	M1	M2	M3
P1	5	4	6
P2	9	8	4
P3	3	18	4

Unrelated

	M1	M2	M3
P1	5	5/1.5	5/2
P2	9	9/1.5	9/2
P3	9	9/1.5	9/2

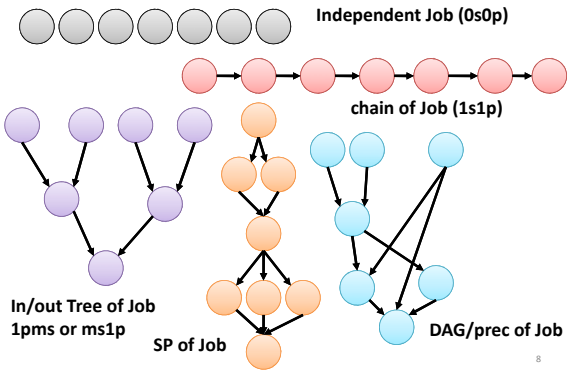
Uniform

6

Job Characteristics

- **pmtn** preemption
- r_j release times /arrival time
- d_j deadlines
- $p_j = 1$ or $p_j = p$ or $p_j \in \{1,2\}$ restricted processing times

Job Precedence Examples



8

Objective Functions

- C_{max} and L_{max} symbolize the bottleneck objective functions with
  - $f_j(C_j) = C_j$ (makespan)
  - $f_j(C_j) = C_j - d_j$ (maximum lateness)
- Common sum objective functions are:
  - ΣC_j (mean flow-time)
  - $\Sigma \omega_j C_j$ (weighted flow-time)

Objective Functions

- **Number of Late Job**
  - ΣU_j (number of late jobs) and $\Sigma \omega_j U_j$ (weighted number of late jobs) where $U_j = 1$ if $C_j > d_j$ and $U_j = 0$ otherwise.
- **Tardiness**
  - ΣT_j (sum of tardiness) and $\Sigma \omega_j T_j$ (weighted sum of tardiness)
  - Tardiness of job **j** is given by

$$T_j = \max \{ 0, C_j - d_j \}.$$

$P_m || C_{max}$

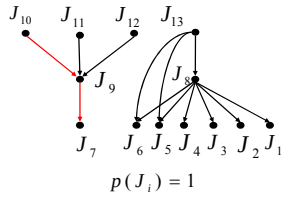
- 2 || Cmax can be easily mapped to 2 set partition problem
  - Pseudo Polynomial Time algorithm
- m≥3, m=3 mapped to 3 partition problem
  - NP Complete Problem
- Approximation Algorithms
  - Grahams List scheduling
  - Longest Task First

11

Precedence constraints (*prec*)

Before certain jobs are allowed to start processing, one or more jobs first have to be completed.

Definition
- Successor
- Predecessor
- Immediate successor
- Immediate predecessor
- Transitive Reduction

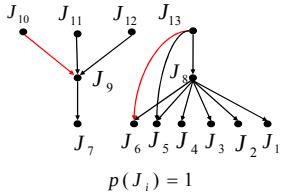


A Sahu

Precedence constraints (*prec*)

One or more job have to be completed before another job is allowed to start processing.

Definition
- Successor
- Predecessor
- Immediate successor
- Immediate predecessor
- Transitive Reduction



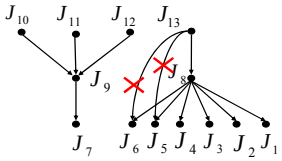
$p(J_i) = 1$

A Sahu

Precedence constraints (*prec*)

One or more job have to be completed before another job is allowed to start processing. *Prec : Arbitrary acyclic graph*

Definition
- Successor
- Predecessor
- Immediate successor
- Immediate predecessor
- Transitive Reduction



$p(J_i) = 1$

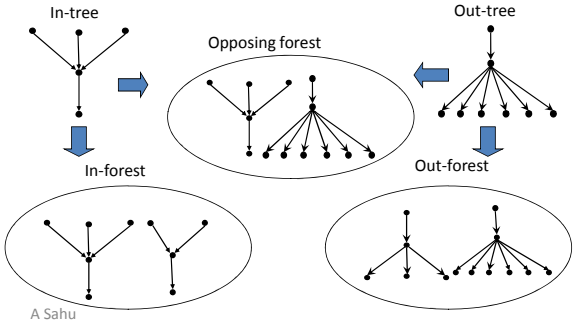
A Sahu

Special precedence constraints

- In-tree (Out-tree)
- In-forest (Out-forest)
- Opposing forest
- *Interval orders*
- *Series-parallel orders*
- *Level orders*

A Sahu

Special precedence constraints



A Sahu

Multiprocessor Scheduling

17

Approximation for:
$Pm|prec,pi=1|C_{max}$ and $Pm||C_{max}$

- **CP Algorithms: Introduction to Algorithms, Corman Leisserson Rivest (CLR) , 3rd Ed, Page 779-783**
- ***Algorithm Design, Eva Tardos, Page 600-605,***

18

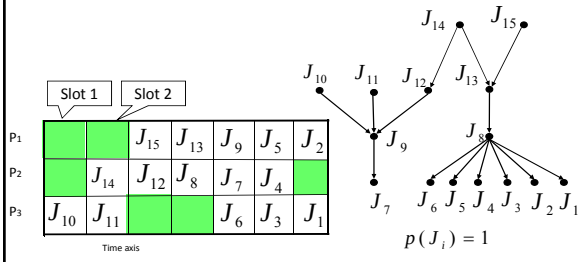
Pm | prec, $p_j = 1$ | C_{max}

- Processor Environment
  - m identical processors are in the system.
- Job characteristics
  - Precedence constraints : precedence graph;
  - Preemption is not allowed;
  - The release time of all the jobs is 0.
- Objective function
  - C_{max} : the time the last job finishes execution.
  - If c_j denotes the finishing time of J_j in a schedule S,

$$C_{max} = max_{1 \le j \le n} c_j$$

Example: Gantt Chart

A Gantt chart indicates the time each job spends in execution, as well as the processor on which it executes ~~*of some Schedule*~~



Pm | prec, $p_j = 1$ | C_{max}

Theorem 1

Pm | prec, $p_j = 1$ | C_{max} is NP-complete.

1. Ullman (1976)

   3SAT ≤ Pm | prec, $p_j = 1$ | C_{max}

2. Lenstra and Rinooy Kan (1978)

   k-clique ≤ Pm | prec, $p_j = 1$ | C_{max}

Corollary 1.1

The problem of determining the existence of a schedule with $C_{max} \le 3$ for the problem Pm | prec, pj = 1 | C_{max} is NP-complete.

Proof: out of Syllabus

Pm | prec, $p_j = 1$ | C_{max}

Mayr (1985)

- **Theorem 2**

  Pm | p_j =1, SP | C_{max} is NP-complete.

  SP: Series - parallel

- **Theorem 3**

  Pm | p_j =1, OF | C_{max} is NP-complete.

  OF: Opposing - forest

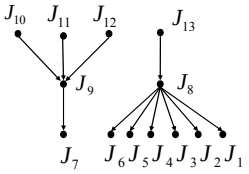
Proof: out of Syllabus

PTAS : Pm | prec, $p_j = 1$ | C_{max}

- PTAS : Polynomial Time Approximation Scheme
- Approximation List scheduling policies
  - Graham's list algorithm
  - HLF algorithm
  - MSF algorithm

List scheduling policies

- Set up a priority list L of jobs.
- When a processor is idle, assign the first ready job to the processor and remove it from the list L.



J_{11}	J_9	J_8	J_6	J_3
J_{10}	J_{13}	J_7	J_5	J_2
J_{12}			J_4	J_1

$$L = (J_9, J_8, J_7, J_6, J_5, J_{11}, J_{10}, J_{12}, J_{13}, J_4, J_3, J_2, J_1)$$

Graham's list algorithm

- Graham first analyzed the performance of the simplest list scheduling algorithm.
- List scheduling algorithm with an arbitrary job list is called Graham's list algorithm.
- Approximation ratio for

$P_m \mid prec, p_j = 1 \mid C_{max}$

App Ratio $\delta = 2 - 1/m$

A Sahu

Pm| | Cmax
Minimum makespan scheduling

- Given
  - Processing times for n jobs $p_1, p_2, \ldots, p_n$
  - And an integer m (number of processor)
- Find an assignment of the jobs to m identical machines
- So that the completion time, also called the makespan, is minimized.

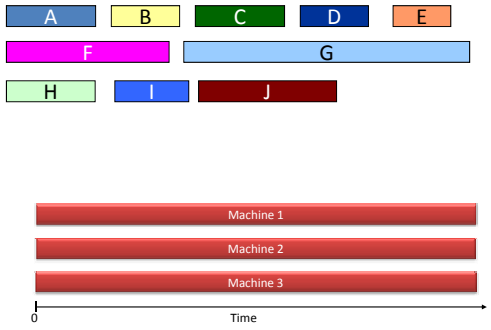
A Sahu

Minimum makespan scheduling: Arbitrary List

- 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.
- Achieves an approximation guarantee of 2

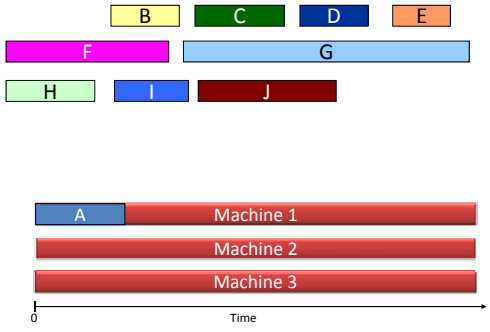
A Sahu

Load Balancing: List Scheduling



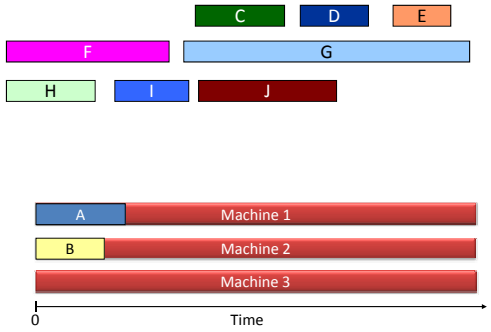
A Sahu

Load Balancing: List Scheduling



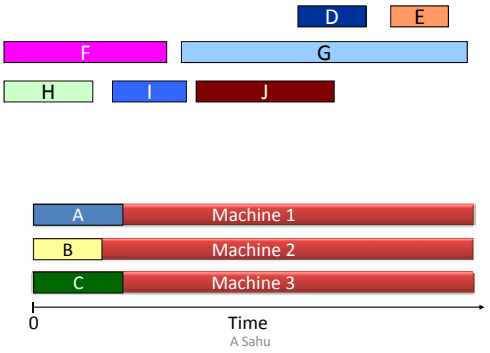
A Sahu

Load Balancing: List Scheduling

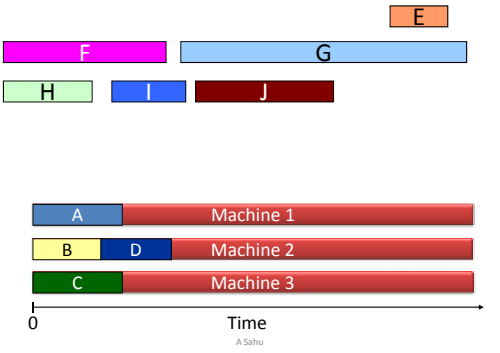


A Sahu

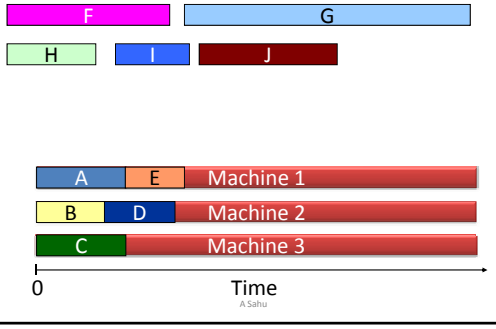
Load Balancing: List Scheduling



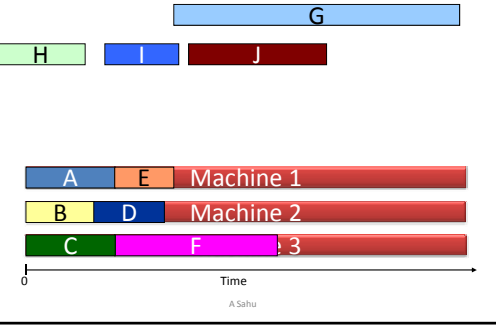
Load Balancing: List Scheduling



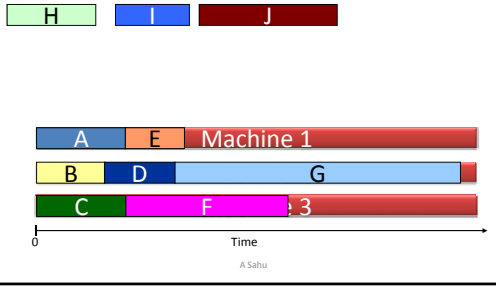
Load Balancing: List Scheduling



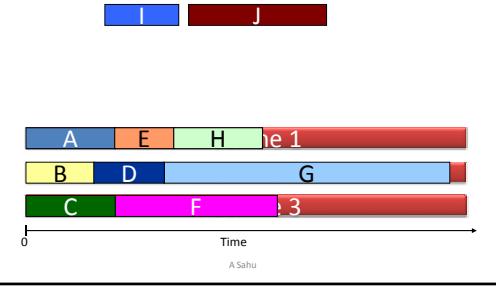
Load Balancing: List Scheduling



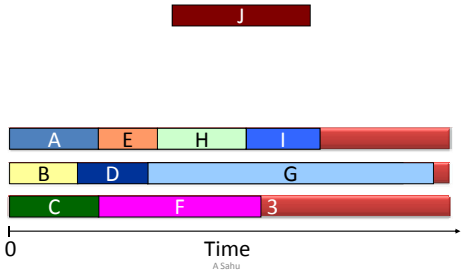
Load Balancing: List Scheduling



Load Balancing: List Scheduling

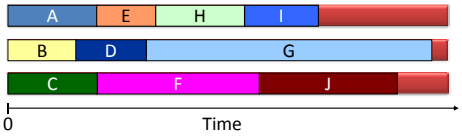


Load Balancing: List Scheduling



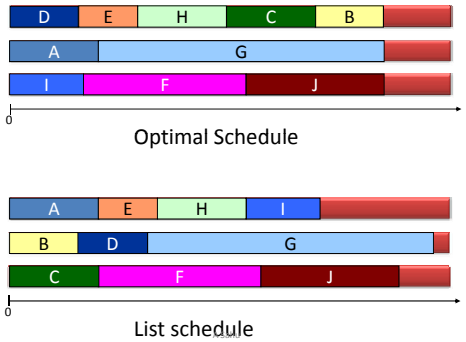
A Sahu

Load Balancing: List Scheduling



A Sahu

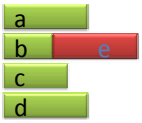
Load Balancing: List Scheduling



A Sahu

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



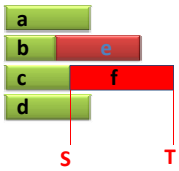
machine 1
machine 2
machine 3
machine 4

Good or bad ?

A Sahu

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



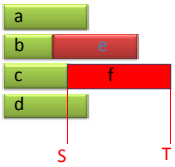
machine 1
machine 2
machine 3
machine 4

job f finishes last, at time T

compare to time OPT of best schedule: how ?

A Sahu

List Scheduling is "2-approximation"



machine 1
machine 2
machine 3
machine 4

job f finishes last, at time T

compare to time OPT of best schedule: how ?

(1) job f must be scheduled in the best schedule at some time: **T – S <= 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 < OPT.**
(3) both together: **T = T – S + S < 2 OPT.**

"2-approximation" (Graham, 1966)

A Sahu

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 m.OPT(I). So, **OPT(I)≥T/m.**
  - Moreover, OPT(I)≥t_k for every k.

A Sahu

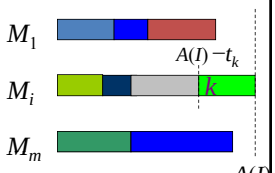
LS achieves a perf. ratio 2–1/*m*.

- Let $A(I)$ 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 $A(I) - t_k$, because then job k would have been scheduled on that machine, thus reducing the makespan.

A Sahu

LS achieves a perf. ratio 2–1/*m*.

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



$T - t_k \geq m(A(I) - t_k)$

→ $T + (m-1)t_k \geq mA(I)$

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

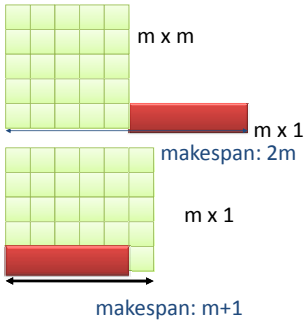
$\leq$ OPT(I) $+(1-1/m)$ OPT(I) $=(2-1/m)$OPT(I)

$A(I) \leq (2-1/m)$OPT(I)

Prev Slide : As m.OPT(I) ≥ T.
So, OPT(I)≥T/m.
Also OPT(I)≥t_k for every k.

A Sahu

Example: Worst Case



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.

A Sahu

List with LPT

LPT(I)

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.

Prove out of Syllabus

- The LPT rule achieves a performance ratio **3/2**
  - ***Page 605, Algorithms Design Eva Tardos***
- The LPT rule achieves a performance ratio **4/3- 1/(3*m*)**. **Ref: Grahams 1969**

A Sahu