CS341: Operating System

Scheduling Algorithms

Lect15 : 3rd Sept 2014

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

1

Outline

- Last Class PTAS for $P_m||C_{max}$ **(NPC)**
- Algorithm for $P_m|$ tree, $p_j = 1 | C_{max}$
- Approximation for $P_m|prec,p_j=1|C_{max}$
- Others Scheduling
  - Distributed
  - Real Time
  - Power Aware and Cost Effective

2

Recap : Approx for $Pm||C_{max}$

- ***Algorithm Design, Eva Tardos, Page 600-605***
  - Scheduling using Arbitrary list **2** Approx
  - Scheduling using Arbitrary list **2-(1/m)** Approx
- **Not proved in class**
  - LPT Rule : **3/2** Approx: (Ref: *Algorithm Design, Eva Tardos, Page 600-605)*
  - LPT Rule : **4/3-(1/3m)** Approx Grahm's Analysis

3

Approx:$P_m|prec,p_i=1|C_{max}$

- CP Algorithms: Introduction to Algorithms, Corman Leisserson Rivest (CLR) , 3rd Ed, Page 779-783
- C L Hu's Algorithm for $P_m|$**tree, $p_i=1|C_{max}$**

4

$Pm|$ 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; r_i or a_i=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,

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

A Sahu

$Pm|$ 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} ≤3 for the problem $Pm|$ prec, pj = 1 | C_{max} is NP-complete.

Proof: out of Syllabus

A Sahu

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

A Sahu

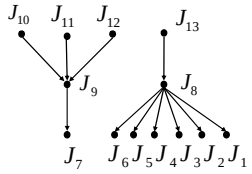
PTAS : Pm | prec, $p_j = 1$ | C_{max}

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

A Sahu

Graham's list scheduling

- **Set up a priority list L of jobs.** How to set the priority ?
- **When a processor is idle, assign the first ready job to 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

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

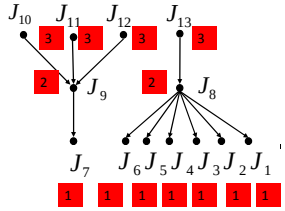
A Sahu

Pm | prec, $p_j = 1$ | C_{max} (HLF/CP Algorithm)

- T. C. Hu (1961), **Critical Path (CP) /Hu's Highest Level First (HLF)** Algorithm
  - Assign a level h to each job.
    - **If job has no successors, h(j) equals 1.**
    - **Otherwise, h(j) equals one plus the maximum level of its immediate successors.**
  - Set up a priority list L by nonincreasing order of the jobs' levels.
  - Execute the list scheduling policy on this level based priority list L

A Sahu

HLF/CP algorithm



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

Level 3 Level 2 Level 1

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

A Sahu

HLF/CP Algorithm

- **Time complexity**
  - O(|V|+|E|) (|V| is the number of jobs and |E| is the number of edges in the precedence graph)
- **The HLF algorithm is Optimal for**
  - **Pm | $p_j = 1$, in-tree (out-tree) | C_{max}.**
  - **Pm | $p_j = 1$, in-forest (out-forest) | C_{max}.**

A Sahu

HLF/CP Algorithm

- The approximation ratio of HLF algorithm for the problem with **general precedence** constraints:

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

Tight!

If m = 2, $\delta_{HLF} \leq 4/3$.

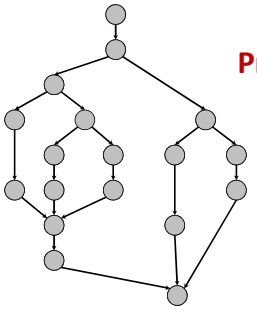
If m ≥ 3, $\delta_{HLF} \leq 2 - 1/(m-1)$.

2 Approx from CLR Algorithm Book

A Sahu

CP Algo: CLR Book Page 779-783

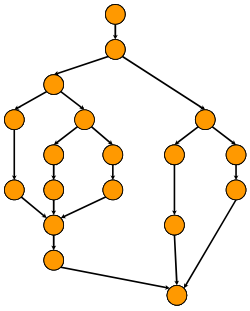
T_P = execution time on P processors



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

CP Algorithms

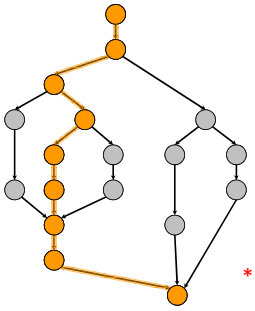
T_P = execution time on P processors



T_1 = ***work***

CP Algorithms

T_P = execution time on P processors



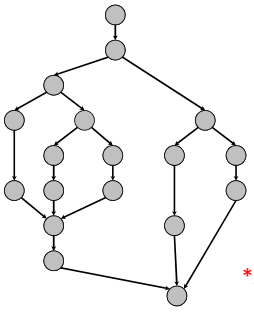
T_1 = ***work***

T_∞ = ***span****

* Also called ***critical-path length*** or ***computational depth***.

CP Algorithms

T_P = execution time on P processors



T_1 = ***work***

T_∞ = ***span****

LOWER BOUNDS

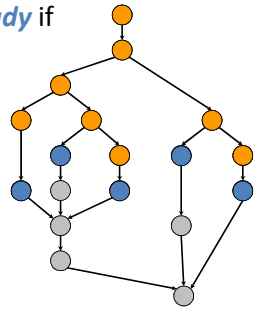
- $T_P \geq T_1/P$
- $T_P \geq T_\infty$

*Also called ***critical-path length*** or ***computational depth***.

CP: Greedy Scheduling

IDEA: Do as much as possible on every step.

Definition: A node is ***ready*** if all its predecessors have ***executed***.



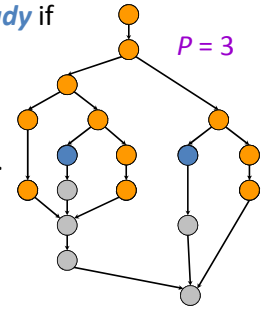
CP : Greedy Scheduling

IDEA: Do as much as possible on every step.

Definition: A node is ***ready*** if all its predecessors have ***executed***.

Complete step

- #ready task ≥ P cores.
- Run any P.



CP : Greedy Scheduling

IDEA: Do as much as possible on every step.

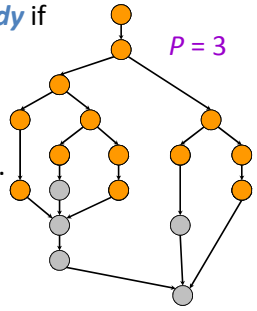
Definition: A node is ***ready*** if all its predecessors have ***executed***.

Complete step

- #ready task ≥ P cores.
- Run any P.

Incomplete step

- # ready task< P cores.
- Run all of them.



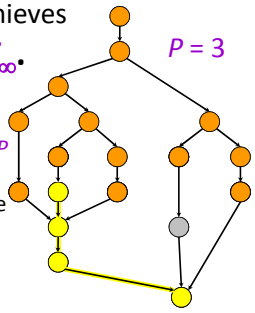
CP: Greedy-Scheduling Theorem

Theorem [Graham '68 & Brent '75]. Any greedy scheduler achieves

$$T_P \le T_1/P + T_\infty .$$

Proof.

- # complete steps ≤ T_1/P, since each complete step performs P work.
- # incomplete steps ≤ T_∞, since each incomplete step reduces the span of the unexecuted dag by 1. ■



CP: Optimality of Greedy

Corollary. Any greedy scheduler achieves within a factor of 2 of optimal.

Proof. Let T_P^* be the execution time produced by the optimal scheduler. Since $T_P^* \ge \max\{T_1/P, T_\infty\}$ (lower bounds), we have

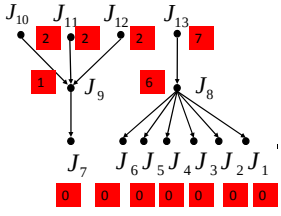
$$\begin{aligned} T_P &\le T_1/P + T_\infty \\ &\le 2 \cdot \max\{T_1/P, T_\infty\} \\ &\le 2T_P^* \ . \ \blacksquare \end{aligned}$$

Most Successors First (MSF) Algo.

- Set up a priority list L by nonincreasing order of the jobs' successors numbers.
  - (i.e. the job having more successors should have a higher priority in L than the job having fewer successors)
- Execute the list scheduling policy based on this priority list L.

A Sahu

MSF Algorithm



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

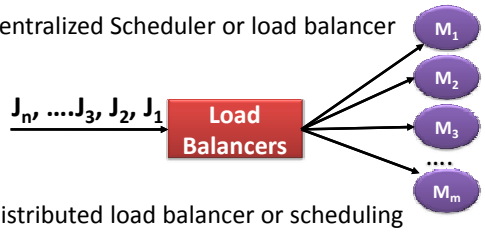
$$L=(J_{13}, J_8, J_{12}, J_{11}, J_{10}, J_9, J_7, J_6, J_5, J_4, J_3, J_2, J_1)$$

7 6 2 2 2 1 0 0 0 0 0 0 0

A Sahu

P||Cmax In Practice

- Centralized Scheduler or load balancer



- Distributed load balancer or scheduling
  - Every one act as peers
  - Every one participate in load balancing

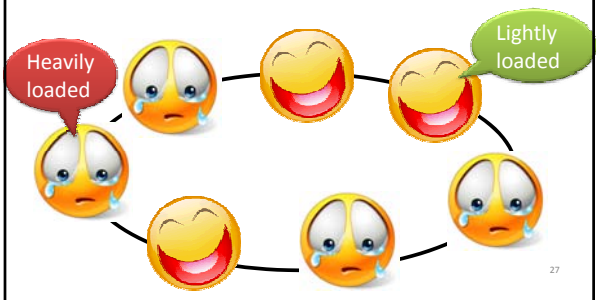
25

Distributed Scheduling

- Resource management component of a system
  - Which moves jobs around the processors
  - To balance load
  - And maximize overall performance
- Needed because of uneven distribution of tasks on individual processors
  - Can be due to several reasons
  - Can even make sense for homogeneous systems with (on average) even loads.

26

Load Sharing



27

Load: How does one characterize ?

- Performance : average response time
- Load:
  - It has been shown that queue lengths for resources (e.g. CPUs) can be a good indicator.
  - How does one handle the delay of transfer when systems are unevenly loaded and we seek to rectify that ?
    - Timeouts, holddowns
  - Queue length not very appropriate for (nor correlated with) CPU utilization for some tasks (e.g. interactive).

Load Balancing Approaches

- Static
  - Decisions are "hard wired"
  - A-priori into the system based on designers understanding
- Dynamic
  - Maintain state information for the system
  - And make decisions based on them
  - Better than static, but have more overhead.
- Adaptive
  - A subtype of dynamic, they can change the parameters they analyze based on system load.

Load Balancing Approaches

- Load balancing vs. Load sharing
  - Centralized: Balancing typically involves more transfers.
  - Distributed: Sharing algorithms transfer in anticipation

Load Sharing

- Sender Initiated
- Receiver Initiated
- Symmetrically Initiated

Sender Initiated Algorithms

- The overloaded node attempts to send tasks to lightly loaded node
- Transfer Policy
  - If new Tasks takes you above threshold, become sender.
  - If receiving task will not lead to crossing over threshold, then become receiver

Sender Initiated Algorithms

- Selection Policy: Newly arrived tasks
- Location Policy
  - Random – still better than no sharing. Constrain by limiting the number of transfers
  - Threshold – chose nodes randomly but poll them before sending task. Limited no. of polls. If process fails execute locally.
  - Shortest – Poll all randomly selected nodes and transfer to least loaded. Doesn't improve much over threshold.

Receiver initiated Algorithms

- Receiver
  - Load sharing process initiated by a lightly loaded node
- Transfer Policy
  - Threshold based
- Selection Policy
  - Can be anything

Receiver initiated Algorithms

- Location Policy
  - Receiver selects upto N nodes and polls them, transferring task from the first sender.
  - If none are found, wait for a predetermined time, check load and try again
- Stability
  - At high loads, few polls needed since senders easy to find.
  - At low loads, more polls but not a problem. However, transfers will tend to be preemptive.

Symmetric Algorithms

- Simple idea
  - Combine the previous two
  - One works well at high loads, the other at low loads.
- Above Average Algorithm
  - Keep load within a range
- Transfer Policy
  - Maintain 2 thresholds equidistant from average
  - Nodes with load > upper are senders, Nodes with load < lower are receivers.

Thanks

37