DISTRIBUTED RESEARCH ON EMERGING APPLICATIONS & MACHINES
dream-lab.in | **Indian Institute of Science**, Bangalore

DREAM:Lab

SE252:Lecture 18/19, Mar 17/19

Application Execution Models on Cloud *(Job & DAG Scheduling)*

Yogesh Simmhan

ILO 4

- Application Execution Models on Clouds:
  - *Design* and *implement* Cloud applications that can scale up on a VM and out across multiple VMs.
  - *Illustrate* the use of load balancing techniques for stateful and stateless applications.
  - *Characterize* resource allocation strategies to leverage elasticity and heterogeneity of Cloud services
  - *Illustrate* the use of NoSQL Cloud storage for information storage and retrieval.

Job Scheduling

Slides on list scheduling courtesy:
Algorithm Design by Éva Tardos and Jon Kleinberg • • Slides by Kevin Wayne

http://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/11ApproximationAlgorithms-2x2.pdf

Approximation Algorithms

Q. Suppose I need to solve an NP-hard problem. What should I do?
A. Theory says you're unlikely to find a poly-time algorithm.

Must sacrifice one of three desired features.

- Solve problem to optimality.
- Solve problem in poly-time.
- Solve arbitrary instances of the problem.

ρ-approximation algorithm.

- Guaranteed to run in poly-time.
- Guaranteed to solve arbitrary instance of the problem
- Guaranteed to find solution within ratio ρ of true optimum.

Challenge. Need to prove a solution's value is close to optimum, without even knowing what optimum value is!

Load Balancing

Input. m identical machines; n jobs, job j has processing time t_j.

- Job j must run contiguously on one machine.
- A machine can process at most one job at a time.

Def. Let J(i) be the subset of jobs assigned to machine i. The load of machine i is $L_i = \sum_{j \in J(i)} t_j$.

Def. The makespan is the maximum load on any machine $L = \max_i L_i$.

Load balancing. Assign each job to a machine to minimize makespan.

Image Processing Jobs on IaaS Cloud

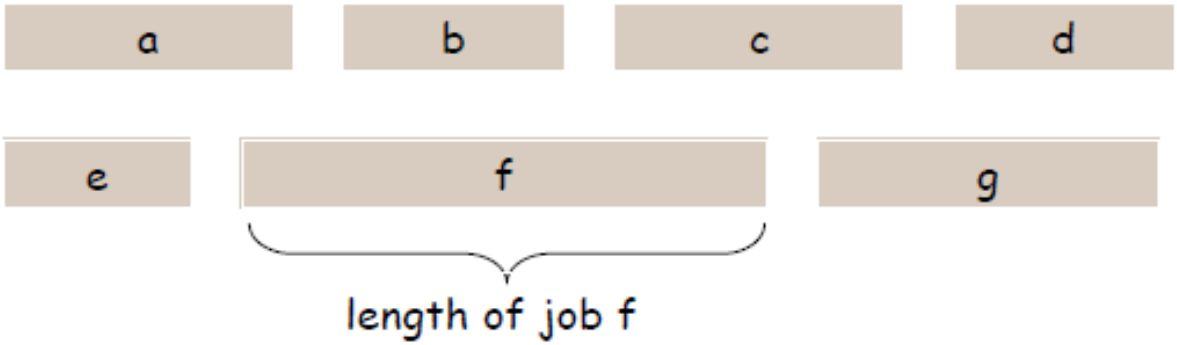
- Users submit "jobs" on a queue
  - *n* jobs available
  - Each job has an estimated time *(e.g. based on image size)*
- Set of *m* VMs are available
- A scheduler looks at all jobs in the queue
- Assigns jobs to VMs
- Goal: To reduce the overall time to process all jobs in the queue

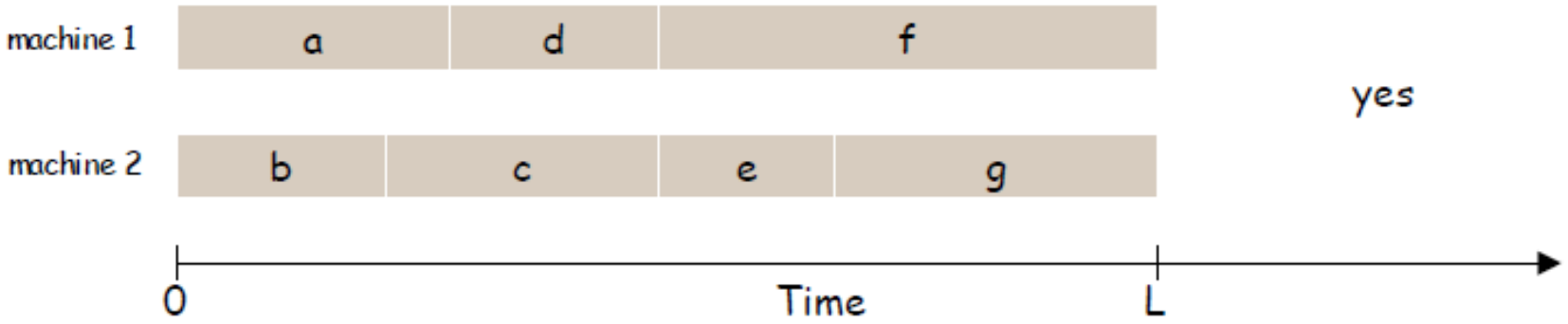
Load Balancing on 2 Machines

Claim. Load balancing is hard even if only 2 machines.

Pf. PARTITION $\leq_p$ LOAD-BALANCE.

PARTITION is a problem where a set of numbers have to be partitioned into two such that the sum of the numbers in each set is as close as possible. This an NP-complete problem, taking $O(2^N)$ for optimal soln.
http://www.americanscientist.org/issues/pub/2002/3/the-easiest-hard-problem/99999





yes

Load Balancing: List Scheduling

List-scheduling algorithm.

- Consider n jobs in some fixed order.
- Assign job j to machine whose load is smallest so far.

```
List-Scheduling(m, n, t_1,t_2,…,t_n) {
    for i = 1 to m {
        L_i ← 0          ← load on machine i
        J(i) ← φ         ← jobs assigned to machine i
    }

    for j = 1 to n {
        i = argmin_k L_k          ← machine i has smallest load
        J(i) ← J(i) ∪ {j}         ← assign job j to machine i
        L_i ← L_i + t_j           ← update load of machine i
    }
}
```

Implementation. O(n log m) using a priority queue.

Load Balancing: List Scheduling Analysis

Theorem. [Graham, 1966] Greedy algorithm is a 2-approximation.

- First worst-case analysis of an approximation algorithm.
- Need to compare resulting solution with optimal makespan L*.

Lemma 1. The optimal makespan $L^* \geq \max_j t_j$.

Pf. Some machine must process the most time-consuming job. ▪

Lemma 2. The optimal makespan $L^* \geq \frac{1}{m} \sum_j t_j$.

Pf.

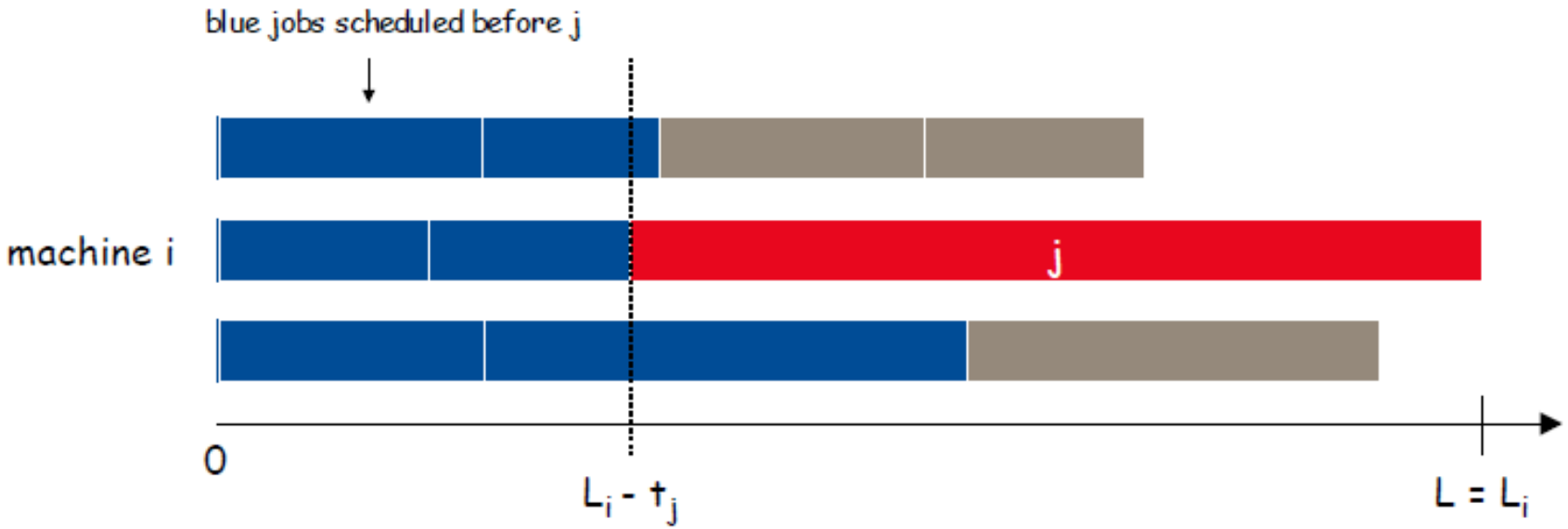
- The total processing time is $\sum_j t_j$.
- One of m machines must do at least a 1/m fraction of total work. ▪

Load Balancing: List Scheduling Analysis

Theorem. Greedy algorithm is a 2-approximation.

Pf. Consider load L_i of bottleneck machine i.

- Let j be last job scheduled on machine i.
- When job j assigned to machine i, i had smallest load. Its load before assignment is $L_i - t_j \Rightarrow L_i - t_j \leq L_k$ for all $1 \leq k \leq m$.



Load Balancing: List Scheduling Analysis

Theorem. Greedy algorithm is a 2-approximation.

Pf. Consider load L_i of bottleneck machine i.

- Let j be last job scheduled on machine i.
- When job j assigned to machine i, i had smallest load. Its load before assignment is $L_i - t_j \Rightarrow L_i - t_j \le L_k$ for all $1 \le k \le m$.
- Sum inequalities over all k and divide by m:

$$\begin{aligned} L_i - t_j &\le \tfrac{1}{m}\textstyle\sum_k L_k \\ &= \tfrac{1}{m}\textstyle\sum_k t_k \\ \text{Lemma 1} \rightarrow &\le L^* \end{aligned}$$

- Now $L_i = \underbrace{(L_i - t_j)}_{\le L^*} + \underbrace{t_j}_{\le L^*} \le 2L^*.$ ▪

Lemma 2

Load Balancing: List Scheduling Analysis

Q. Is our analysis tight?

A. Essentially yes.

Ex: m machines, m(m-1) jobs length 1 jobs, one job of length m

1	11	21	31	41	51	61	71	81	91
2	12	22	32	42	52	62	72	82	Machine 2 idle
3	13	23	33	43	53	63	73	83	Machine 3 idle
4	14	24	34	44	54	64	74	84	Machine 4 idle
5	15	25	35	45	55	65	75	85	Machine 5 idle
6	16	26	36	46	56	66	76	86	Machine 6 idle
7	17	27	37	47	57	67	77	87	Machine 7 idle
8	18	28	38	48	58	68	78	88	Machine 8 idle
9	19	29	39	49	59	69	79	89	Machine 9 idle
10	20	30	40	50	60	70	80	90	Machine 10 idle

m = 10, list scheduling makespan = 19

Load Balancing: List Scheduling Analysis

Q. Is our analysis tight?

A. Essentially yes.

Ex: m machines, m(m-1) jobs length 1 jobs, one job of length m

1	11	21	31	41	51	61	71	81	10
2	12	22	32	42	52	62	72	82	20
3	13	23	33	43	53	63	73	83	30
4	14	24	34	44	54	64	74	84	40
5	15	25	35	45	55	65	75	85	50
6	16	26	36	46	56	66	76	86	60
7	17	27	37	47	57	67	77	87	70
8	18	28	38	48	58	68	78	88	80
9	19	29	39	49	59	69	79	89	90
91									

m = 10, optimal makespan = 11

Load Balancing: LPT Rule

Longest processing time (LPT). Sort n jobs in descending order of processing time, and then run list scheduling algorithm.

```
LPT-List-Scheduling(m, n, t_1,t_2,…,t_n) {
    Sort jobs so that t_1 ≥ t_2 ≥ … ≥ t_n

    for i = 1 to m {
        L_i ← 0          ← load on machine i
        J(i) ← φ         ← jobs assigned to machine i
    }

    for j = 1 to n {
        i = argmin_k L_k          ← machine i has smallest load
        J(i) ← J(i) ∪ {j}         ← assign job j to machine i
        L_i ← L_i + t_j           ← update load of machine i
    }
}
```

Load Balancing: LPT Rule

Observation. If at most m jobs, then list-scheduling is optimal.

Pf. Each job put on its own machine. ▪

Lemma 3. If there are more than m jobs, $L^* \geq 2\, t_{m+1}$.

Pf.

- Consider first m+1 jobs $t_1, \ldots, t_{m+1}$.
- Since the t_i's are in descending order, each takes at least t_{m+1} time.
- There are m+1 jobs and m machines, so by pigeonhole principle, at least one machine gets two jobs. ▪

Theorem. LPT rule is a 3/2 approximation algorithm.

Pf. Same basic approach as for list scheduling.

$$L_i = \underbrace{(L_i - t_j)}_{\leq L^*} + \underbrace{t_j}_{\leq \frac{1}{2}L^*} \leq \tfrac{3}{2}L^*. \quad \blacksquare$$

Lemma 3
(by observation, can assume number of jobs > m)

Load Balancing: LPT Rule

Q. Is our 3/2 analysis tight?

A. No.

Theorem. [Graham, 1969] LPT rule is a 4/3-approximation.

Pf. More sophisticated analysis of same algorithm.

Q. Is Graham's 4/3 analysis tight?

A. Essentially yes.

Ex: m machines, n = 2m+1 jobs, 2 jobs of length m+1, m+2, ..., 2m-1 and one job of length m.

DREAM:Lab

DAG Scheduling

DAG Scheduling

- Map all tasks in a DAG to computing resources
  - Computational time
  - Data dependencies, transfer costs
- Often done statically
  - Assumes deterministic behavior of apps and machines
  - Batch operation

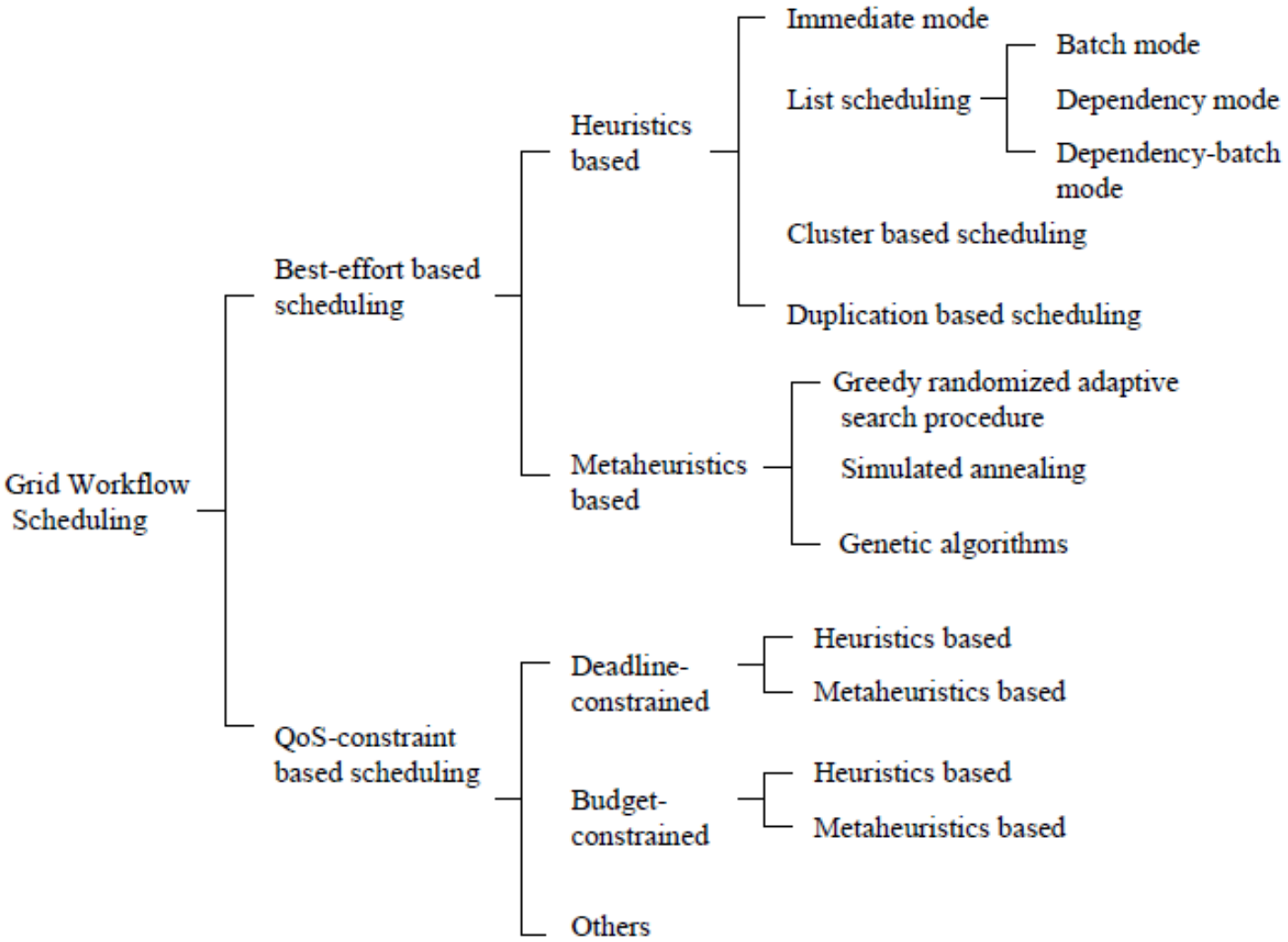


Fig. 5.2. A taxonomy of Grid workflow scheduling algorithms.

Workflow Scheduling for Grid Computing, Wu, 2008

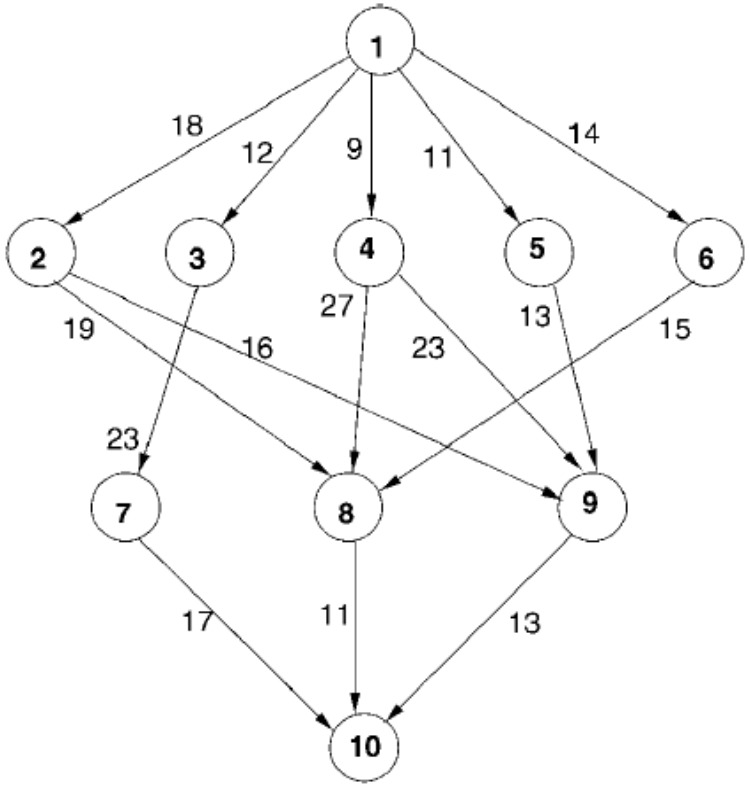
Best Effort: HEFT

- Heterogeneous Earliest Finish Time (HEFT)
  - Heuristic Scheduling
  - Asymmetric resources
  - Communication costs between tasks
  - Known time per task
- Static scheduling at start of runtime
  - Pick a schedule and stick to it

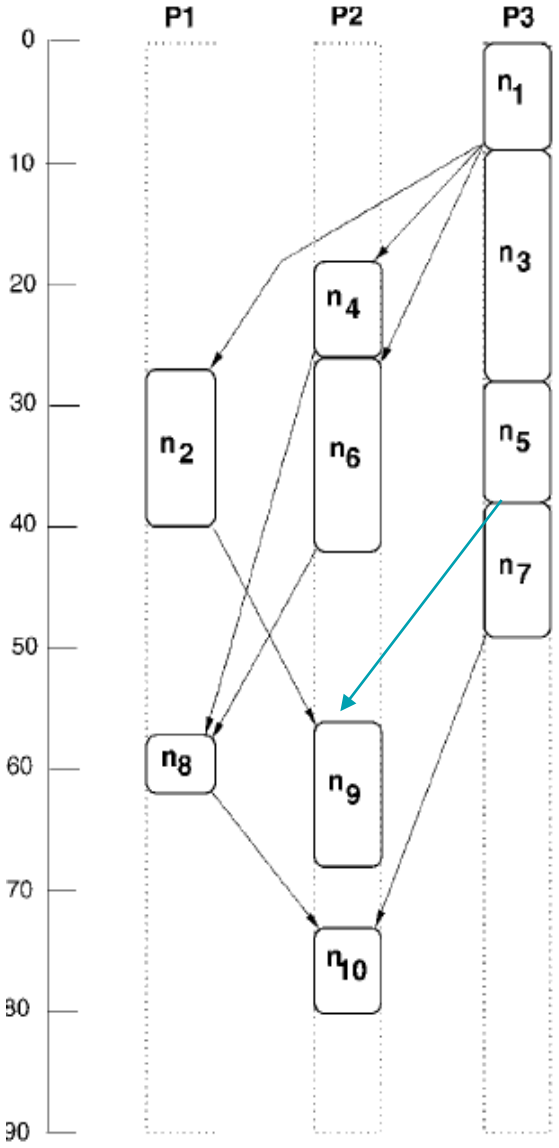
Topcuoglu, Hariri and Wu, Performance-effective and Low Complexity task Scheduling for Heterogeneous Computing, *TPDS*, 2002

Problem Definition

- Given a DAG $G = (v, e)$
  - The v vertices are tasks
  - The e edges are ***control*** dependencies
  - There are only one *Entry* & one *Exit* tasks
- $data$ is a $v \times v$ matrix of data exchanges between tasks
- q heterogeneous machines in the cluster
- W is a $v \times q$ matrix with execution times for tasks v on machines q
- L is vector of size q with the communication latency time for each machine
- B is a $q \times q$ matrix with bandwidth between each pair of q machines



Task	P1	P2	P3
1	14	16	9
2	13	19	18
3	11	13	19
4	13	8	17
5	12	13	10
6	13	16	9
7	7	15	11
8	5	11	14
9	18	12	20
10	21	7	16



Topcuoglu, Hariri and Wu, Performance-effective and Low Complexity task Scheduling for Heterogeneous Computing, *TPDS*, 2002

Data & Compute Estimates

- Average *task processing time* of task i

$$\overline{w_i} = \frac{\sum_{m=1}^{q} W_{(i,m)}}{q}$$

- *Communication time* for task i on machine m with a successor task k on machine n is:

$$c_{i,k} = L_m + {data_{i,k}}/{B_{m,n}}$$

- Average communication time for task i

$$\overline{c_{i,k}} = \bar{L} + {data_{i,k}}/{\overline{B}}$$

Observations of Heterogonous Earliest Finish Time (HEFT) Algorithm, Kevin Tzeng

Critical Path Estimates

- **Upward Rank:** Length of Critical Path from task i to $exit$ task, including computation cost

$$rank_u(i) = \overline{w_i} + \max_{j \in \text{succ}(i)} (\overline{c_{i,j}} + rank_u(j))$$

$$\text{where } rank_u(exit) = \overline{w_{exit}}$$

- **Downward Rank:** Length of Critical Path from $entry$ task to task i, excluding computation cost

$$rank_d(i) = \max_{j \in \text{pred}(i)} (rank_d(j) + \overline{c_{j,i}} + \overline{w_j})$$

$$\text{where } rank_d(entry) = 0$$

Start & Finish times of tasks

- **Earliest Start Time:** Earliest possible time at which a task i can be started on machine m. Machine m must be available, *and* predecessors of i must be finished.

$$EST(i,m) = \max\left\{avail[m], \max_{j \in \text{pred}(i)} (AFT(j) + c_{j,i})\right\}$$

$$EST(entry,m) = 0$$

- **Earliest Finish Time:** Earliest possible time at which task i can finish its execution on machine m

$$EFT(i,m) = EST(i,m) + w_{(i,m)}$$

- **Actual Start Time (AST)**
- **Actual Finish Time (AFT)**

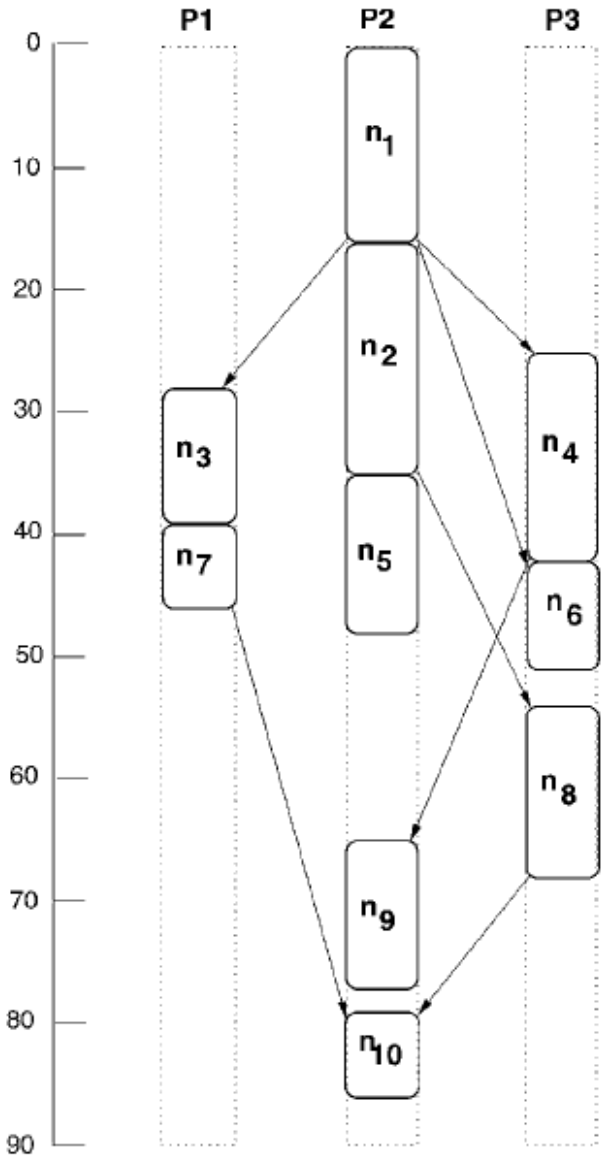
Pseudo Code: HEFT

1. Set the computation costs of tasks and communication costs of edges to their mean values.
2. Compute the $rank_u$ for all tasks by traversing graph backwards, starting from the exit task.
3. Sort the tasks in a scheduling list by non-increasing order of $rank_u$ values.
4. **while** there are unscheduled tasks in the list **do**
   - 4.1. Remove the first task i, from the list for scheduling.
   - 4.2. **For** each machine m **do**
     - 4.2.1. Compute the $EST(i,m)$ value
   - 4.3. Assign task i to the machine m that minimized EFT of the task i.
5. **End while**

Computational Complexity = $O(e \times q)$

Critical Path on a Processor: CPOP

- Uses critical path as the lower bound of schedule
- Task priority is based on *sum of upward & downward ranks*
- Schedule highest priority pending task, iteratively
- Attempts to co-locate critical path processors



Ongoing Assignments

- Upload all project midterm slides & code to GitHub by TODAY (pre-req for grades)
- HW B will be posted on Mar 19

Reading Assignment

- Text book Chs: 2.4.1, 2.4.2, 4.5.2, 6.2.6
- *Workflow Scheduling for Grid Computing, Wu, 2008*
- Topcuoglu, Hariri and Wu, Performance-effective and Low Complexity task Scheduling for Heterogeneous Computing, *TPDS*, 2002