CMPE 117

Scheduling Anomalies

- Theorem [Graham, 1976]:
  If a task set is optimally scheduled on a multiprocessor with some priority assignment, fixed execution times, and precedence constraints, then the following can increase the schedule length:
  - increasing the number of processors
  - reducing execution times
  - weakening the precedence relation

Scheduling: vocabulary

- : arrival time
- C: computation time
- d: deadline
- s: starting time
- f: finishing time
- L = f - d: lateness
- E = max(0, L): tardiness
- X = d - a - C: slack

Scheduling: types

- Preemptive vs. non-preemptive.
- Synchronous (all processes arrive at once) or asynchronous (processes arrive one by one)
- Static (a, C, d of all processes known in advance) or on-line.
- Independent, or with dependency relation.
- Aperiodic vs. periodic vs. hybrid (mixed)
### Scheduling anomalies - example

Weakening the precedence relation

<table>
<thead>
<tr>
<th>1</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
</tr>
</tbody>
</table>

### Multiprocessor Scheduling is NP-complete

- **INSTANCE:** a set $J$ of tasks, a number $m$ of processors, a global deadline $d$, and for each $j \in J$, a computation time $C_j$
- **QUESTION:** is there a partition $J = J_1 \cup J_2 \cup \ldots \cup J_m$ such that
  $$\max_{1 \leq p \leq m} \sum_{j \in J_p} C_j \leq d?$$


---

### Where's the catch?

- **Theorem [Graham, 1976]:**
  - If a task set is optimally scheduled on a multiprocessor with some priority assignment, fixed execution times, and precedence constraints, then the following can increase the schedule length...
  - Priorities are a bad idea! They confuse two concepts:
    - How important is something?
    - What should be scheduled next?

### Scheduling without Precedence Constraints

---

### Where's the catch?

- **Theorem [Graham, 1976]:**
  - If a task set is optimally scheduled on a multiprocessor with some priority assignment, fixed execution times, and precedence constraints, then the following can increase the schedule length...
  - Classical scheduling methods for multiprocessors may not be optimal, but optimality may be hard to achieve (NP-complete).

### Earliest Due Date (EDD)

$1|\text{synch}|L_{\text{max}}$

**Use when:**
- Synch: all jobs have $r=0$ (arrive immediately)
- Optimizes maximum lateness
- No precedence relations
- No preemption necessary
**Earliest Due Date (EDD)**

Given a set of processes 
\[ \{ C_i, d_i \}, \{ C_i, d_i \}, ..., \{ C_i, d_i \} \]

in order to minimize \( L_{\text{max}} \), it suffices to schedule the processes in order of increasing deadlines.

- **EDD Algorithm**:
  1) Sort the deadlines in increasing order
  2) Schedule the processes in order of increasing deadline.

- **Schedulability criterion**:
  Once ordered so that \( d_1 < d_2 < ... < d_m \), check that \( \sum_{i=1}^{k} C_i < d_k \) for all \( k \in \{1, ..., m\} \).

---

**EDD Optimality**

**Assume** \( d_1 < d_2 \)

\[ L_A < L_{A'} \]

\[ L_B' < L_B \]

\[ L_{A'} = L_A \]

Hence

\[ L' = \max \{ L_{A'}, L_{A'}, L_{A'}, L_B \} < L = \max \{ L, L, L, L_B \} \]

---

**Proof of EDF Optimality**

---

**Earliest Deadline First (EDF)**

\[ 1|\text{asynch}|L_{\text{max}} \]

Use when:
- Tasks can have arbitrary arrival time.
- They are preemptable. This is important.

---

**Scheduling with Precedence Constraints**
Scheduling with precedence constraints (1 | prec, syn | Lmax): LDF (Latest Deadline First)

- Build the schedule from last to first, by recursively:
  - Let Q be the subset of jobs all whose descendants have been scheduled.
  - Pick the job p from Q that has the latest deadline.
  - Add p to the front of the schedule.
- Should be really called LDL (Latest deadline last).
- Optimal wrt. Lmax.
LDF Example

**EDF with precedence constraints**

(1 | async, prec, preem | Lmax)

- Algorithm by Chetto, Silly, Bouchenouf (1990).
- Idea: transform timing constraints to enforce precedence (cannot start before predecessors, cannot pre-empt successors), then use EDF.

**LDF Example**

Schedule:

J₁, J₂, J₃, J₄, J₅

**EDF with precedence constraints**

(1 | async, prec, preem | Lmax)

- Transform the release times and deadlines
- Schedule by EDF

**LDF: Proof of Optimality**

J₉ should be last according to LDF

Lmax = max{l₁, l₂, l₃, l₄}

- L₁ unchanged
  - L'₁ = L₁ as B starts earlier
- L₂ unchanged
  - L'₂ = L₂ as J₂ starts earlier
- L₃ = f₋₃ - d₃ ≤ f₋₆ - d₆ (as d₆ > d₃) = L₆
  - So L'₃ ≤ L₆

The argument is concluded by induction.
EDF with precedence constraints
(1 | async, prec, preem | Lmax)

- When \( J_o < J_b \), then:
  - \( s_b \geq r_b \) (\( J_b \) must start after being released)
  - \( s_b \geq r_b + C_o \) (so \( J_o \) has time to finish)
- Thus, replace \( r_b \) by \( r_b^* = \max\{r_b, r_o + C_o\} \)

EDF: Modification of the Deadlines

- Algorithm for deadline modification:
  - For all sinks \( J_o \) of dependency graph, let \( d^*_o = d_o \)
  - Select a task \( J_b \) such that all successors of \( J_b \) have already been modified; let \( H \) be this set.
  - Let \( d^*_b = \min \{d_b, \min_{c \in H} (d^*_c - C_c)\} \)
  - And recur.

EDF Modification Example

EDF with precedence constraints
(1 | async, prec, preem | Lmax)

- Algorithm for release times:
  - For all roots \( J_o \) of dependency graph, let \( r^*_o = r_o \)
  - Select a task \( J_b \) such that all predecessors of \( J_b \) have already been modified; let \( H \) be this set.
  - Let \( r^*_b = \max \{r_b, \max_{c \in H} (r^*_c + C_c)\} \)
  - And recur.

EDF: Modification of the Deadlines

- Given \( J_o \) and \( J_b \) s.t. \( J_o \prec J_b \), then the following conditions must hold:
  - \( f_o \leq d_b \) (or else we miss the deadline)
  - \( f_b \leq d_b - C_b \) (or there won’t be enough time for \( J_o \) to finish)
- Hence, replace \( d_o \) by \( d^*_o = \min\{d_o, d_b - C_b\} \)
EDF Modification Example

Another Example

EDF Modification Example

Another Example

Sometimes we can see right away that the problem is not schedulable.
EDF + prec: Proof of optimality

- The algorithm provides a translation EDF+Prec 1 modified-EDF.
- We can show that the translation preserves schedulability.
- First direction: if EDF+Prec is schedulable, then modified-EDF is also schedulable (the same schedule works).

EDF + prec: Proof of optimality

- Converse: if modified-EDF is schedulable, the same schedule shows that EDF+Prec is schedulable.
- Clearly, the schedule produced by modified EDF meets all release times and deadlines of the original problem; we must only prove that the precedence relation is respected.
- Assume \( J_e \prec J_b \) and \( C_b > 0 \). Then, \( D'_e < D'_b \) and \( r'_e < r'_b \), so EDF schedules \( J_e \) before \( J_b \).

Non-Preemptive Scheduling

- Jobs have release times, arrive dynamically, and there are no precedence relations.
- There are two kinds of algorithms:
  - Non-idle, that always execute a job as long as there is one ready
  - May-Idle, that may choose to do nothing even though there are jobs ready
- There are cases when may-idle algorithms outperform non-idle ones....
Being idle is sometimes a good idea

But: unless the arrival times are known, are may-idle algorithms reasonable? They require knowledge about the future!

EDF is best of non-idle 1|async|L_{\text{max}} algorithms

- This is a theorem by Jeffay, Stanat, and Martel (1991).
- Intuition: if you can’t help being active, at least do the most useful thing at hand, that is, the one with the earliest deadline.

Aperiodic scheduling: summary

<table>
<thead>
<tr>
<th>independent</th>
<th>sync</th>
<th>preempt</th>
<th>non-preempt</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>EDF</td>
<td>EDF</td>
<td>if non-idle, EDF</td>
</tr>
<tr>
<td>prec constr.</td>
<td>LDF</td>
<td>EDF with modifications</td>
<td>Search</td>
</tr>
</tbody>
</table>