



Reward-Based Voltage Scheduling for Fixed-Priority Hard Real-Time Systems

Han-Saem Yun Jihong Kim

Computer Architecture and Embedded Systems Lab.

Seoul National University, KOREA

Workshop on Power-Aware Real-Time Computing
(PARC'04)

Reward-Based Scheduling

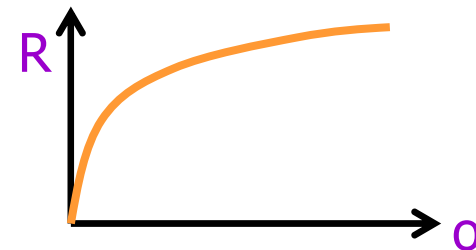
- A generalization of IC(Imprecise Computation) and IRIS(Increasing Reward with Increasing Service) model.
 - can model various **flexible** RT applications,
 - which allow approximate (but timely) result.
 - e.g.: media processing, navigation/control, GA/neural net, ...
- The workload of a task is divided into two parts.

mandatory part

optional part

- A **reward** function is associated with the optional part.

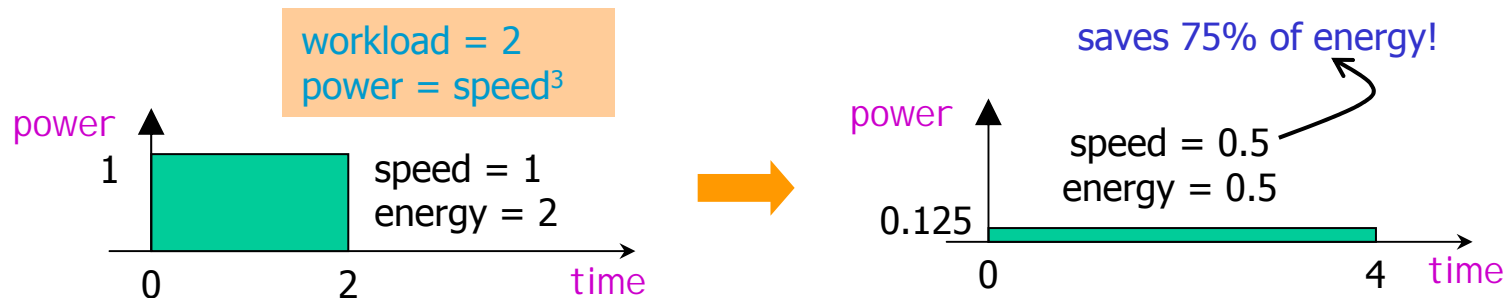
- nondecreasing & concave



- Goal: select optional parts such that the total reward is maximized while meeting all the deadlines.
 - Optional parts should also be completed by deadlines.

Voltage Scheduling (a.k.a. DVS)

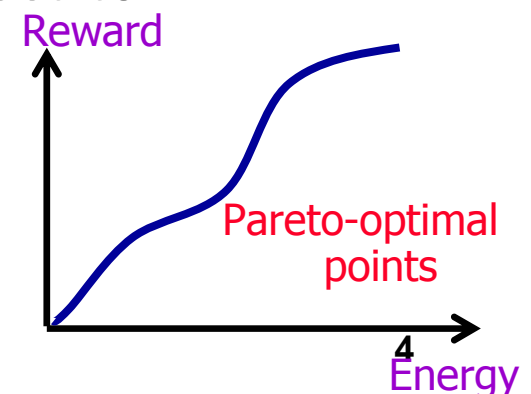
- Variable voltage processor
 - can adjust the voltage (& the speed) dynamically.
 - Lowering the voltage/speed reduces the energy consumption.
 - tradeoff between the response time & the energy consumption
 - e.g., Intel Xscale, AMD Mobile K6, Transmeta Crusoe



- Voltage schedule $S(t)$
 - S : time \rightarrow processor speed (equivalently, voltage)
 - $S(t)$ determines the energy consumption & response time.
- Goal: find $S(t)$ that minimizes the energy consumption
 - while meeting all the deadlines.

Reward-Based Voltage Scheduling

- What if a flexible application is executed on a VVP?
 - We should consider the combined scheduling problem.
- The problem involves two-dimensional objectives.
 - (1) maximizing the **total reward** (from reward-based sched.)
 - (2) minimizing the **energy consumption** (from voltage sched.)
- Goal: find (1) **optional parts** and (2) **voltage schedule** such that the **total reward** is maximized subject to
 - timing constraints and **energy budget**.
 - Can be defined as duals.



Existing Works

- Rusu et al. [TECS'03] found an optimal off-line solution for
 - frame-based tasks
 - the same period, release time/deadline
 - periodic **EDF** tasks (deadline = period)
- Yun & Kim [RTCSA'04] developed an optimal off-line algorithm and an on-line algorithm for
 - **EDF** static job model
 - used in low-power design community (e.g., Yao [FOCS'95])
 - Each job has its own release time/deadline known off-line.
 - Typical periodic task sets can be transformed into the static job model by considering all the task instances (i.e., jobs) within the hyperperiod.
 - Rusu et al.'s work cannot handle the case deadline \neq period.
 - Allows a finer scheduling granularity

Contributions

(1) Off-line Job-level **fixed-priority** scheduling

- c.f.: [Yun, RTCSA'04] → Job-level off-line **EDF** scheduling
- NP-hard
- A **fully polynomial time approximation scheme** (FPTAS)
 - the best one can hope for an NP-hard problem

(2) Off-line task-level **fixed-priority** scheduling

- c.f.: [Rusu, TECS'03] → Task-level off-line **EDF** scheduling
- NP-hard & a heuristic

(3) On-line **fixed-priority** scheduling

- leverages the workload variation to increase the reward.
- only 6~13% worse than the theoretical (off-line) optimum.

Roadmap

- Problem Formulation
- Off-line Job-Level Scheduling
- Off-line Task-Level Scheduling
- On-line Scheduling
- Experimental Results
- Summary

Problem Formulation (Task-level)

- Task set: $\{T_1, T_2, \dots, T_N\}$ with fixed-priority assignment
 - p_i, d_i : the period & the relative deadline of a task T_i
 - m_i : the mandatory workload of T_i
 - u_i : the upper bound of the optional workload of T_i
 - The optional workload o_i is selected within $[0, u_i]$.
 - $f_i(o_i)$: the reward (a function of the optional workload o_i)
 - e.g., $f_i(o_i) = \log(1+o_i)$
- Reward-based scheduling problem
 - Solution space: $\mathbf{O} = (o_1, o_2, \dots, o_N)$: optional workloads (workload tuple)
 - Objective: the total reward $F(\mathbf{O}) = \sum f_i(o_i)$
 - Constraints: RM/DM schedulability (CPU time: $m_i + o_i/\text{speed}$)

Problem Formulation (Task-level)

- a_i : execution time allocated to T_i
 - Then, T_i 's speed = $(m_i+o_i)/a_i$ (voltage schedule..)
- Solution space: $\langle \mathbf{A}, \mathbf{O} \rangle$
 - $\mathbf{O} = (o_1, o_2, \dots, o_N)$: optional workloads
 - $\mathbf{A} = (a_1, a_2, \dots, a_N)$: voltage schedule
- Objective: the total reward $F(\mathbf{O}) = \sum f_i(o_i) \cdot H/p_i$
- Constraints: the energy budget & deadlines
 - energy constrains: $E(\langle \mathbf{A}, \mathbf{O} \rangle) = \sum a_i \cdot P(o_i/a_i) \cdot H/p_i \leq E_{\text{budget}}$
 - P : power as a function of the processor speed (e.g., $P(s)=s^3$)
 - deadline constraints: RM/DM schedulability
 - CPU time: a_i

Problem Formulation (Job-level)

- Job set $\{J_1, J_2, \dots, J_N\}$: instances of fixed-priority tasks
 - r_i, d_i : the release time & deadline of a job J_i
 - m_i : the mandatory workload of J_i
 - u_i : the upper bound of the optional workload of J_i
 - $f_i(o_i)$: the reward (a function of the optional workload o_i)
- The solution space: $\langle \mathbf{A}, \mathbf{O} \rangle$'s
 - $\mathbf{A} = (a_1, a_2, \dots, a_N)$: voltage schedule
 - $\mathbf{O} = (o_1, o_2, \dots, o_N)$: optional workloads
- Objective: the total reward $F(\mathbf{O}) = \sum f_i(o_i)$
- Constraints: the energy budget & deadlines
 - energy constrains: $E(\langle \mathbf{A}, \mathbf{O} \rangle) = \sum P(o_i/a_i) \cdot a_i \leq E_{\text{budget}}$
 - deadline constraints: dependent only on \mathbf{A} (similar to RMA..)

Roadmap

- Problem Formulation
- Off-line Job-Level Scheduling
- Off-line Task-Level Scheduling
- On-line Scheduling
- Experimental Results
- Summary

Formulation Revisited

- Job set $\{J_1, J_2, \dots, J_N\}$: instances of fixed-priority tasks
 - p_i : the priority of a job J_i
 - r_i, d_i : the release time & deadline of J_i
- Note that the system model covers all the priority policy
 - Fixed-priority (RM/DM) periodic/aperiodic task set
 - EDF periodic/aperiodic task set
 - $p_i < p_j$ iff $d_i < d_j$
 - This special case can be solved in poly-time by Yao [FOCS'95]
- Source of difficulty: deadline constraints is complicated
 - For EDF job sets, the condition is very simple

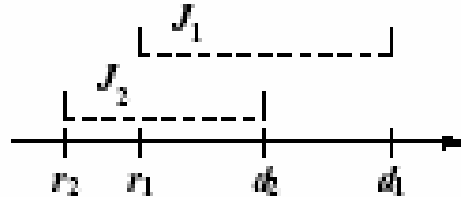
EDF

$$\text{For any } r_i < d_j \text{ (} 1 \leq i, j \leq |\mathcal{J}| \text{), } \sum_{k/[r_k, d_k] \subseteq [r_i, d_j]} a_k \leq d_j - r_i$$

in general

$$\begin{aligned} &\text{There exists a } |\mathcal{J}|\text{-tuple } (f_1, f_2, \dots, f_{|\mathcal{J}|}) \in \mathcal{T}^{\mathcal{J}} \text{ such that} \\ &\forall 1 \leq i \leq |\mathcal{J}| \quad \forall r \in \{t | t \in R_{\mathcal{J}} \wedge t < f_i\} \\ &\quad \sum_{J_k / p_{J_k} \leq p_{J_i} \wedge r_{J_k} \in [r, f_i]} a_k \leq f_i - r. \end{aligned}$$

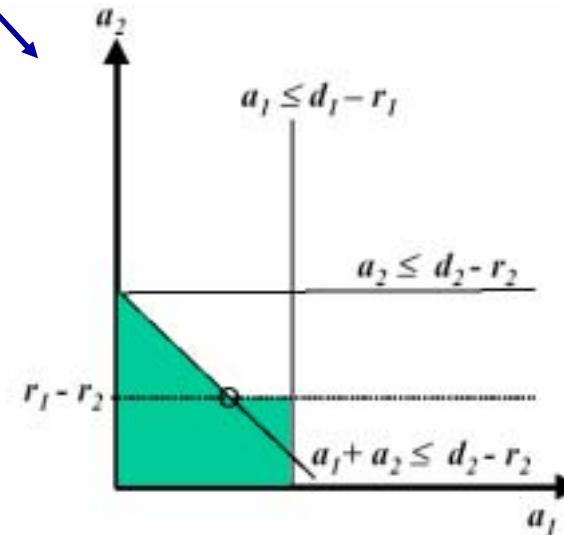
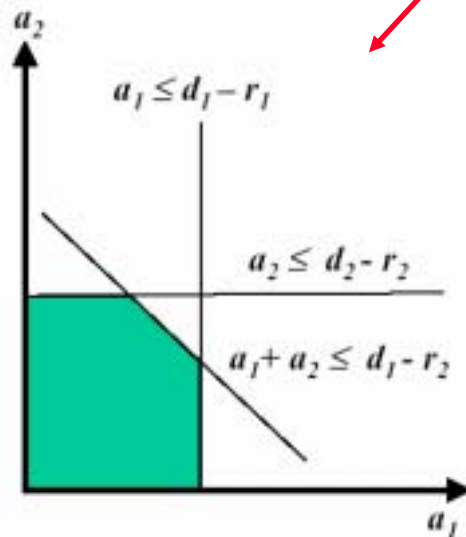
More on Solution Space $\langle A, O \rangle$



a_1 : time allocated to J_1
 a_2 : time allocated to J_2

$p_1 > p_2$ (EDF)

$p_1 < p_2$ (general)

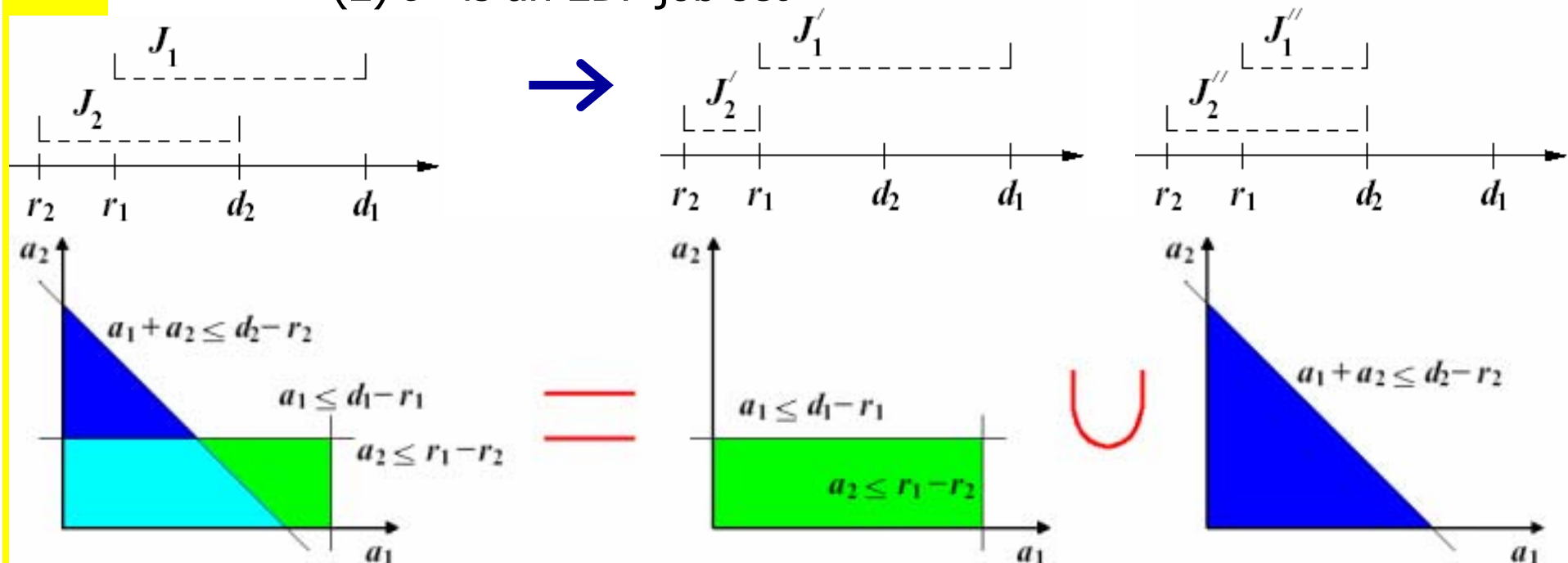


- generally, convex
 - easy to solve
 - c.f., Yao's algorithm

- concave
 - generally, intractable
 - **how to explore ??**

More on Solution Space $\langle A, O \rangle$

- $F_J: \{ A \mid A \text{ meet deadline constraints of a job set } J \}$
- For a fixed-priority job set J , $F_J = F_{J_1} \cup F_{J_2} \cup \dots$
 - where F_{J_i} 's are **EDF-equivalent** job set of J [Quan, DATE'02]
 - J' is EDF-equivalent to J if
 - (1) each deadline is modified to an earlier (or equal) one, and
 - (2) J' is an EDF job set



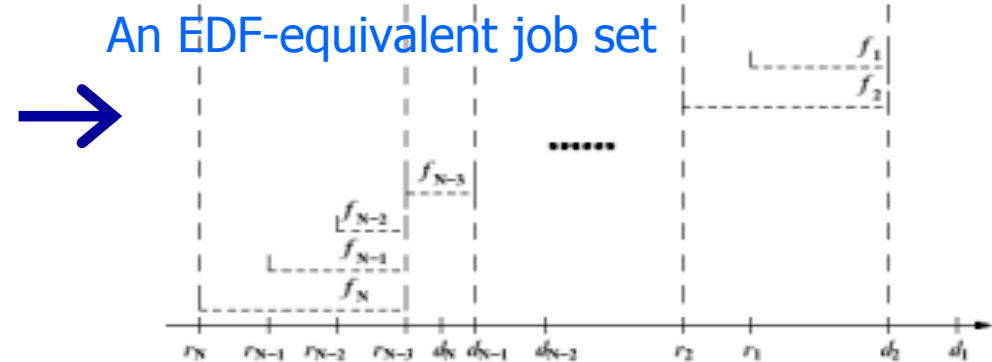
More on Solution Space $\langle A, O \rangle$

- $F_J: \{ A \mid A \text{ meet deadline constraints of a job set } J \}$
- For a fixed-priority job set J , $F_J = F_{J_1} \cup F_{J_2} \cup \dots$
 - where F_{J_i} 's are **EDF-equivalent** job set of J [Quan, DATE'02]
- Then, how can we enumerate all the EDF-equivalent sets?
 - For an N-jobs set, there are $O(N!)$ EDF-equivalent job sets.
- [Yun & Kim, TECS'03] used dynamic programming formulation to enumerate EDF-equivalent job sets
 - by introducing **blocking tuples** (e.g. $(r_N, r_{N-3}, d_{N-1}, \dots, r_2, d_2)$)

Fixed-priority job set

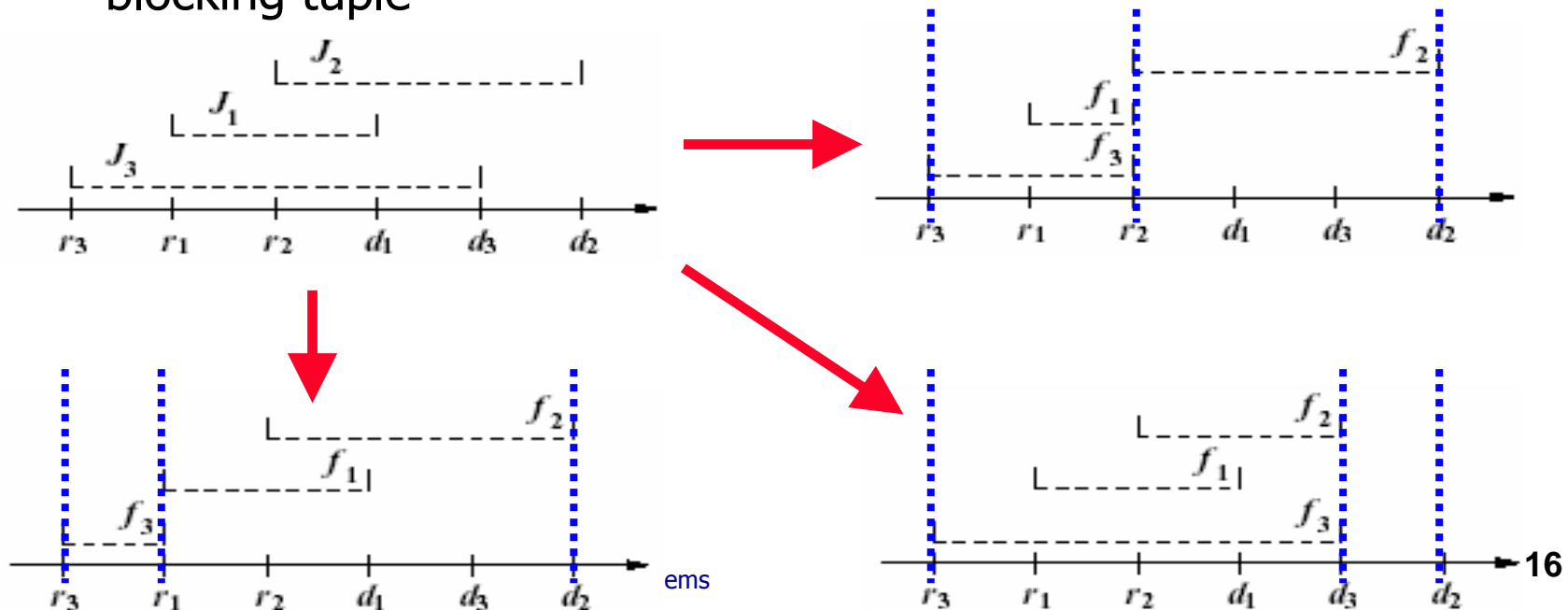


An EDF-equivalent job set



Blocking Tuple (informally..)

- $\mathcal{J}[t, t'] : \{ J_i' \mid J_i \in \mathcal{J}, r_i \in [t, t'] \}$
 - where $J_i' \equiv J_i$ except that the deadline of J_i' is modified to t' .
 - i.e., obtained by trimming away jobs in \mathcal{J} that are released outside $[t, t']$ and advancing deadlines to t' .
- $[t, t']$ is said to be atomic if $\mathcal{J}[t, t']$ is an EDF job set.
- (b_1, b_2, \dots, b_k) is a blocking tuple if $[b_j, b_{j+1}]$'s are atomic.
- For any EDF-equivalent job set of $\{J_1, J_2, J_3\}$, there is a corresponding blocking tuple



More on Blocking Tuples

- $F(J',e)$: the maximum achievable reward for the job set J' with the energy budget of e
- For an atomic interval $[t,t']$, $F(J[t,t'],e)$ can be found by the optimal EDF algorithm [Yun and Kim, RTCSA'04]
 - Note that $J[t,t']$ is an EDF job set (since $[t,t']$ is atomic)
- Assume that for any EDF-equivalent job set of J , there is a corresponding blocking tuple.
 - Can be extended by introducing background workload..
- Then,

$$F(J, E_{\text{budget}}) = \max\left\{ \sum_{j=1}^{k-1} F(J[b_j, b_{j+1}], e_j) \mid [b_j, b_{j+1}]'s \text{ are atomic and } \sum_{j=1}^{k-1} e_j \leq E_{\text{budget}} \right\}$$

Dynamic Programming

$$F(\mathcal{J}, E_{\text{budget}}) = \max\{ \sum_{j=1 \text{ to } k-1} F(\mathcal{J}[b_j, b_{j+1}], e_j) \mid (b_j, b_{j+1})'s \text{ are atomic and } \sum_{j=1 \text{ to } k-1} e_j \leq E_{\text{budget}} \}$$

- An exact (but infinite) dynamic programming formulation:

$$F(\mathcal{J}[0, t_i], e) = \max\{ F(\mathcal{J}[0, t_{j'}], e - \Delta e) \mid [t_{j'}, t_i] \text{ is atomic, } 0 \leq \Delta e \leq e \}$$

- Can be transformed into an FTPAS by discretizing energy values (e.g., e_1, e_2, \dots)

$$F(\mathcal{J}[0, t_i], e_j) = \max\{ F(\mathcal{J}[0, t_{j'}], e_j - e_{j'}) \mid [t_{j'}, t_i] \text{ is atomic, } e_{j'} \leq e_j \}$$

- We can take discretized energy values e_1, e_2, \dots such that
 - e_1, e_2, \dots are close enough to guarantee the error bound
 - while keep the running time bounded by a polynomial

Roadmap

- Problem Formulation
- Off-line Job-Level Scheduling
- **Off-line Task-Level Scheduling**
- On-line Scheduling
- Experimental Results
- Summary

Off-line Task-level Scheduling

- Let $A[\mathbf{O}]$ represent the voltage schedule (for fixed workload \mathbf{O}) obtained by a voltage scheduling algorithm
 - We use Gruian's voltage scheduling algorithm [ISLPED'01]
- Start by $\mathbf{O} := \mathbf{U} = (u_1, u_2, \dots, u_N)$
 - \mathbf{U} : the upper bounds of \mathbf{O} ($0 \leq o_i \leq u_i$)
 - $F(\mathbf{U})$: the maximum total reward when $E_{\text{budget}} = \infty$
 - return \mathbf{U} if $E(\langle A[\mathbf{U}], \mathbf{U} \rangle) \leq E_{\text{budget}}$
- Otherwise, decrease \mathbf{O} until $E(\langle A[\mathbf{O}], \mathbf{O} \rangle)$ reaches E_{budget}
 - as with the general descent method

```
while (  $E(\langle A[\mathbf{O}], \mathbf{O} \rangle) \leq E_{\text{budget}}$  )  
     $\delta := \text{compute\_descent\_direction}();$   
     $\Delta := \text{compute\_descent\_step\_size}();$   
     $\mathbf{O} := \mathbf{O} - \Delta \cdot \delta$ 
```

Off-line Task-level Scheduling

```
while (  $E(\langle A[\mathbf{O}], \mathbf{O} \rangle) \leq E_{\text{budget}}$  )  
     $\delta := \text{compute\_descent\_direction}();$   
     $\Delta := \text{compute\_descent\_step\_size}();$   
     $\mathbf{O} := \mathbf{O} - \Delta \cdot \delta$ 
```

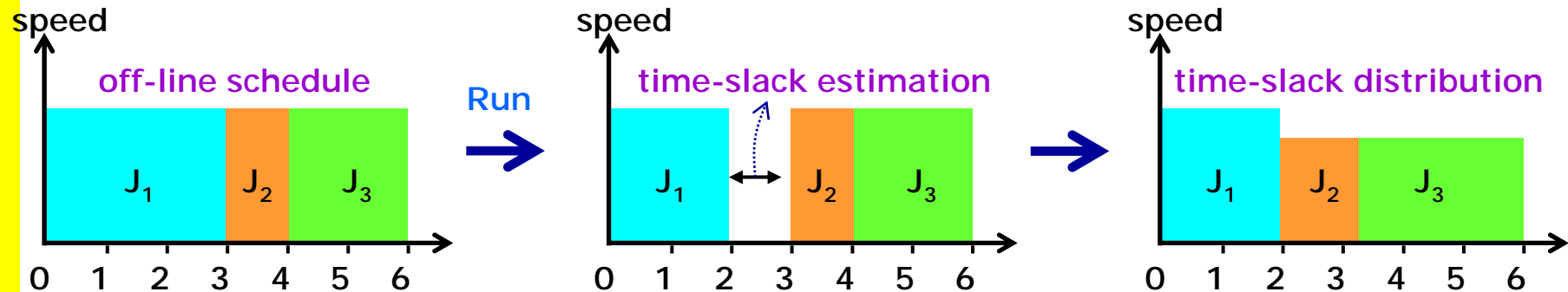
- **Descent direction**: the one that minimizes $dF(\mathbf{O})/dE(\langle A[\mathbf{O}], \mathbf{O} \rangle)$, i.e.,
 - the decrease in the reward per unit decrease in the energy
 - keep the reward decrease as small as possible
 - However, $E(\langle A[\mathbf{O}], \mathbf{O} \rangle)$ is not in closed-form.
 - We fix $A[\mathbf{O}]$ when considering the differential, regarding that a small change in \mathbf{O} will not lead to a large variation in $A[\mathbf{O}]$
- **Descent step size**: sufficiently large to guarantee a polynomial number of steps

Roadmap

- Problem Formulation
- Off-line Job-Level Scheduling
- Off-line Task-Level Scheduling
- **On-line Scheduling**
- Experimental Results
- Summary

On-Line Algorithm

- Extend conventional on-line voltage scheduling
 - on-line voltage scheduling consists of two parts:
 - (1) **time-slack estimation** (e.g., priority-based slack stealing)
 - (2) **time-slack distribution** (e.g., greedy, mean-proportional)



- In addition to **time-slack**, our on-line algorithm manages **energy-slack**
 - Residual energy reserved by a lower speed or idle time
 - $E_{\text{slack}}(t) = E_{\text{off-line}}(t) - E_{\text{run-time}}(t)$
- As with the time-slack, energy-slack management consists of two parts: (1) estimation and (2) distribution

On-line Algorithm

	time-slack	energy-slack
estimation	priority-based slack stealing (existing method)	$E_{\text{slack}}(t) = E_{\text{off-line}}(t) - E_{\text{run-time}}(t)$
distribution	in the following...	in the following ...

- Time-slack & energy-slack are distributed among jobs in the ready queue s.t. the **gradient** (instead of the **speed**) of each job is **as flat as possible**
 - a property from the off-line (task-level) algorithm
 - c.f.: on-line voltage-scheduling algorithms flatten the **speed**
 - **gradient** of a job J_i : $P'(s_i)/f'_i(m_i+o_i)$ (s_i : the speed of J_i)
- Distribute Δt and ΔE among J_{i_1}, \dots, J_{i_k}
 - Find $\Delta a_{i_1}, \dots, \Delta a_{i_k}$ and $\Delta s_{i_1}, \dots, \Delta s_{i_k}$ such that
 - $P'(s_i + \Delta s_i)/f'_i((a_i + \Delta a_i) \cdot (s_i + \Delta s_i))$ is as uniform as possible
 - subject to (1) $\Delta t \geq \sum \Delta a_{i_j}$ and
 (2) $\Delta E \geq \sum P(s_{i_j} + \Delta s_{i_j}) \cdot (a_{i_j} + \Delta a_{i_j}) - P(s_{i_j}) \cdot a_{i_j}$

Experimental Results

- Benchmarks: MPEG4 Videophone, CNC, Avionics
- Logarithmic reward functions: $a_i \cdot \log(b_i \cdot o_i + 1)$
- Task-level heuristic VS. job-level FPTAS (1%)
 - MPEG4 : within 0.7% , CNC/Avionics : within 2~3%
 - Periods are almost harmonic. utilization ≈ 1
- On-line algorithm VS. job-level FPTAS (1%)
 - Workload fluctuation: uniform within $[WCET/2, WCET]$
 - Job-level FPTAS compute the near-optimal solution with the complete trace information (i.e., actual workload)
 - On-line algorithm is worse 5~12% than the base schedule

Summary

- Consider the combined scheduling problem of
 - reward-based scheduling and voltage scheduling
- FPTAS for the off-line job-level scheduling problem
- Efficient heuristic for the task-level scheduling problem
- On-line algorithm
- Future work: fixed-priority scheduling for 0/1 reward functions