Optimizing Energy Consumption under Flow and Stretch Constraints

Zhi Zhang, Fei Li

Department of Computer Science
George Mason University
{zzhang8, lifei}@cs.gmu.edu

November 17, 2011
Contents

1  Motivation

2  Model

3  Algorithms and Analysis
   ● Offline Algorithms
   ● Online Algorithms

4  Conclusion
Necessity of Designing Energy-Aware Scheduling Algorithms

Substantial increase of energy consumption has brought up many serious engineering problems and economic concerns to our society.

Challenge of minimizing energy consumption must be addressed.

Figure: Cray XT5HE supercomputers at Oak Ridge National Laboratories.
Credit: http://en.wikipedia.org/wiki/Cray_XT5

Consider the supercomputer Jaguar:

Its peak power consumption is 6,950 kilowatts per hour. At $0.07 per kilowatt an hour, it costs $4 million to run this supercomputer full-bore in one year.

Saving Energy = Saving Money
Necessity of Designing Energy-Aware Scheduling Algorithms

Figure: A simple policy of saving energy consumption

- We do not have to keep all machines active all the time, since sometimes, there is no job to run.
- A simple algorithmic policy is to shut down one machine if there is no job to execute for 30 minutes and to power on machine when new jobs arrive.
- Following this algorithmic power-aware job/machine scheduling policy, Kyoto University saves **200 kW** and **$200 K** per year. [Hikita, et al, IPDPS, 2008]
Dynamic Power Management (DPM)

- System has one higher-power active state and one or more lower-power sleep or standby states. Energy cost per time unit in active state is larger than in sleep state.
- Jobs can be processed only when system is in active state but not in any lower-power state.
- DPM saves energy by setting system into lower-power state when it is idle. Transition is not free. Each transition incurs energy cost $E_{\text{tran}}$.

**Example**

DPM scheduling based on Baptiste’s algorithm [Baptiste, et al, ESA, 2007]

- Three jobs: (release time, deadline, processing time)
  - $J_1 = (0, 24, 6)$ runs in interval [2, 3] and [8, 13],
  - $J_2 = (3, 8, 5)$ runs in interval [3, 8],
  - $J_3 = (13, 20, 7)$ runs in interval [13, 20].
- Transition cost: $E_{\text{tran}} = 6$.
- Energy consumed:
  \[
  E = 6 + (1 + 5 + 5 + 7) \cdot 1 = 24.
  \]
Flow Time and Stretch

A Motivating Question

What side effects does a DPM policy have?
Flow Time and Stretch

A Motivating Question
What side effects does a DPM policy have?

Two Kinds of DPM
- Non-idling DPM: the machine cannot be in idling state if there are pending jobs.
- Idling-allowed DPM: the machine is allowed to be in idling state even if there are pending jobs.

Flow Time and Stretch
- Flow time = completion time - release time.
- Stretch = $\frac{\text{Flow time}}{\text{processing time}}$. 
**Flow Time and Stretch**

**Example**

Scheduling Jobs with non-idling DPM and idling-allowed DPM

- **Non-idling DPM**
  - Three jobs: (release time, processing time)
  - $J_1 = (0, 3), J_2 = (8, 5), J_3 = (20, 4)$.

- **Idling-allowed DPM**
  - $E_{tran} = 6$.

<table>
<thead>
<tr>
<th></th>
<th>Energy Consumption</th>
<th>Maximum Flow Time, Stretch</th>
</tr>
</thead>
<tbody>
<tr>
<td>non-idling DPM</td>
<td>29</td>
<td>5, 1</td>
</tr>
<tr>
<td>idling-allowed DPM</td>
<td>18</td>
<td>15, 5</td>
</tr>
</tbody>
</table>
Flow Time and Stretch

Example

Scheduling Jobs with non-idling DPM and idling-allowed DPM

<table>
<thead>
<tr>
<th>Energy Consumption</th>
<th>Maximum Flow Time, Stretch</th>
</tr>
</thead>
<tbody>
<tr>
<td>non-idling DPM</td>
<td>29</td>
</tr>
<tr>
<td>idling-allowed DPM</td>
<td>18</td>
</tr>
</tbody>
</table>

A Dilemma

to save energy ⇒ grouping jobs together ⇒ flow/stretch increased
to minimize flow/stretch ⇒ executing jobs immediately ⇒ energy increased
Previous Work

- **Stretch is used as the metric in online job scheduling:** [M. A. Bender, et al, SODA, 1998], [M. A. Bender, et al, SODA, 2002].

  However, energy consumption is **NOT** considered in the above model.

- In the **DVS**\(^1\) setting, an online version of the model of minimizing the sum of total energy and total flow time is considered by [S. Albers, et al, TALG, 2007].

  We consider **DPM** setting here.

- In the **DPM** setting, an offline version of scheduling jobs in minimizing total energy consumption is considered by [P. Baptiste, et al, ESA, 2007].

  However, it does **NOT** consider the tradeoff between energy consumption and jobs’ flow time/stretch.

---

\(^1\) Dynamic voltage scaling (DVS) saves energy by slowing down CPUs.
Our Contribution

- This paper studies a bi-criteria model that minimize maximum flow time or largest stretch under a fixed energy budget and minimize total energy consumption under an upper bound of flow time or stretch.

- Four optimal offline algorithms and two online algorithms are given.

- We also analyze algorithms in running time complexities and weak competitive ratios.
Contents

1. Motivation

2. Model

3. Algorithms and Analysis
   - Offline Algorithms
   - Online Algorithms

4. Conclusion
Model

- Machine environment (a single machine)
  - idling-allowed DPM setting two states: **active** (1) and **sleep** (0).
  - transition cost: $E_{\text{tran}}$.
  - total energy consumption: $E_{\text{total}}$

- A set of jobs, for each job $j$
  - release time: $r_j \in \mathbb{R}^+$
  - processing time: $p_j \in \mathbb{R}^+$

**Definition**

**Flow time; Stretch.** A Job $j$'s flow time $f_j$ is defined as the difference between its completion time $c_j$ and release time $r_j$. Its stretch $s_j$ is defined as ratio between its flow time and its processing time.

\[
 f_j = c_j - r_j.
\]

\[
 s_j = \frac{f_j}{p_j} = \frac{c_j - r_j}{p_j}.
\]

The maximum flow time is $F_{\text{max}} = \max_j f_j$ and the largest stretch is $S_{\text{max}} = \max_j s_j$. 
Model (Cont.)

- **Constraints**
  - upper bound of a job’s flow time: $F_{\text{bound}}$
  - upper bound of a job’s stretch: $S_{\text{bound}}$
  - total energy budget: $E_{\text{budget}}$

- **Objectives**
  1. Model 1: $\min E_{\text{total}}, \text{ subject to } F_{\text{max}} \leq F_{\text{bound}}$
  2. Model 2: $\min F_{\text{max}}, \text{ subject to } E_{\text{total}} \leq E_{\text{budget}}$
  3. Model 3: $\min E_{\text{total}}, \text{ subject to } S_{\text{max}} \leq S_{\text{bound}}$
  4. Model 4: $\min S_{\text{max}}, \text{ subject to } E_{\text{total}} \leq E_{\text{budget}}$
Contents

1 Motivation

2 Model

3 Algorithms and Analysis
   - Offline Algorithms
   - Online Algorithms

4 Conclusion
Model 1: Minimize Energy Consumption under an Upper Bound of Flow Time

Idea
- Design optimal algorithm ALG for non-idling DPM setting. This algorithm gives the minimum flow time.
- Extend the solution of ALG to our solution OPT for idling-allowed DPM setting.

ALG (Schedule Min. Flow under Non-idling DPM Setting)
1. Append newly arrival jobs at the end of the pending job queue.
2. Execute jobs in FIFO order whenever queue is not empty.

Definition
**Batch of jobs.** A set of jobs executed in a back to back and consecutive manner in the schedule given by ALG.
Model 1: Minimize Energy Consumption under an Upper Bound of Flow Time (Cont.)

**Properties of OPT**

- Jobs execute in a **FIFO** order in OPT.
- All the jobs in the same batch in ALG are still in the same batch in OPT.
- A batch $G_i$ in ALG will be either
  1. executed at the same start position as in ALG; or
  2. postponed only if in OPT, the machine is powered off before executing $G_i$ and spinning after completing $G_i$.

**Solution**

The algorithm uses **dynamic programming** approach. OPT has a running time of $O(n^2)$. 
Model 1: Minimize Energy Consumption under an Upper Bound of Flow Time

(Cont.)

Properties of OPT

- Jobs execute in a **FIFO** order in OPT.
- All the jobs in the same batch in ALG are still in the same batch in OPT.
- A batch $G_i$ in ALG will be either
  1. executed at the same start position as in ALG; or
  2. postponed only if in OPT, the machine is powered off before executing $G_i$ and spinning after completing $G_i$.

![Figure: Three possible cases in delaying batch 2 for different given flow time bounds.](image_url)
Model 2: Minimize the Maximum Flow Time under a Bounded Energy Consumption

Observation
As energy budget increases, the maximum flow time decreases.

Idea
1. Use ALG to generate the minimal maximum flow time. This is the lower bound of the maximum flow time.
2. Use OPT to generate the minimum energy consumption given the maximum flow time.
3. Use doubling technique to find out the best maximum flow time $F_{\text{max}}$ under $E_{\text{budget}}$.

Theorem
There exists an optimal algorithm minimizing the maximum flow time subject to the total energy consumption bounded by $E_{\text{budget}}$. This algorithm has a running time of $O(n^2 \log F_{\text{max}}^*)$. 
Model 3: Minimize Energy Consumption under an Upper Bound of Stretch

For a job $j$, its stretch

$$s_j = \frac{f_j}{p_j} = \frac{c_j - r_j}{p_j} \leq S_{\text{bound}},$$

which is

$$c_j \leq r_j + p_j \cdot S_{\text{bound}}.$$

**Idea**

1. Give each job $j$ a new variable $d_j = r_j + p_j \cdot S_{\text{bound}}$ to denote its artificial deadline.
2. Use Baptiste, et al’s algorithm\(^a\) scheduling jobs with deadlines to solve the new model.

\(^a\)[P. Baptiste, et al, ESA, 2007]

**Theorem**

*There exists an $O(n^5)$-time algorithm minimizing total energy consumption subject to each job’s stretch bounded by $S_{\text{bound}}$.***
Model 4: Minimize the Largest Stretch under a Bounded Energy Consumption

Observation
An optimal algorithm with energy budget $E_{\text{budget}}^1$ has a no-smaller largest stretch than that of an optimal algorithm with energy budget $E_{\text{budget}}^2$, where $E_{\text{budget}}^1 \leq E_{\text{budget}}^2$.

Idea
1. Use the solution for Model 3 to get the minimum energy cost and the maximum stretch.
2. Use the doubling technique to find the best maximum stretch $S_{\text{max}}^*$.

Theorem
There exists an $O(n^5 \log S_{\text{max}}^*)$-time algorithm minimizing the largest stretch $S_{\text{max}}$ subject to the total energy consumption bounded by $E_{\text{budget}}$. 
## Offline Algorithms Conclusion

<table>
<thead>
<tr>
<th>restrictions</th>
<th>minimizing</th>
<th>performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>bounded maximum flow time</td>
<td>energy consumption</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>bounded energy consumption</td>
<td>maximum flow time</td>
<td>$O(n^2 \log F_{\text{max}}^*)^2$</td>
</tr>
<tr>
<td>bounded largest stretch</td>
<td>energy consumption</td>
<td>$O(n^5)$</td>
</tr>
<tr>
<td>bounded energy consumption</td>
<td>largest stretch</td>
<td>$O(n^5 \log S_{\text{max}}^*)^3$</td>
</tr>
</tbody>
</table>

$F_{\text{max}}^*$ is the optimal maximum flow time

$S_{\text{max}}^*$ is the optimal largest stretch
Contents

1 Motivation
2 Model
3 Algorithms and Analysis
   • Offline Algorithms
   • Online Algorithms
4 Conclusion
Online Algorithms

Definition (Competitive Analysis)

Online algorithm $ON$ is $c$-competitive against an optimal offline algorithm $OPT$, if for any finite input sequence $I$, an additive constant $\delta$,

$$c = \max_{I} \frac{OPT(I) - \delta}{ON(I)}$$

Competitive analysis provides a pessimistic result.
Given a constant $c$, there exists an input instance such that any online algorithm has a competitive ratio (of flow time or total energy consumption) $> c$.

Example
Online Algorithms

Definition (Weak Competitive Ratio Respect to Flow Time)

Let online algorithm’s maximum flow time be $F^{ON}$ and energy consumption be $E^{ON}$. The weak competitive ratio respect to extra flow time is $E^{ON}/E^*$ subject to $F^{ON} \leq F^* + \hat{F}$.\(^a\)

\(^a\) $F^*$ is the optimal maximum flow time, $E^*$ is the optimal energy consumption, and $\hat{F}$ is the extra flow time.

Definition (Weak Competitive Ratio Respect to Stretch)

Let online algorithm’s largest stretch be $L^{ON}$ and energy consumption be $E^{ON}$. The weak competitive ratio respect to larger stretch is $E^{ON}/E^*$ subject to $L^{ON} \leq \alpha L^*$, where $\alpha \geq 1$ is a given number.\(^a\)

\(^a\) $L^*$ is the optimal largest stretch, and $E^*$ is the optimal energy consumption
Online algorithm 1: Minimize Energy Consumption under an Upper Bound of Extra Flow Time $\hat{F}$

**Idea**

Use lazy scheduling approach. Schedule a job at the time when its flow time achieves $f^* + \hat{F}$, where $f^*$ is a job's optimal flow time.

**Solution**

1. If the machine is active, execute jobs in a FIFO order.
2. Go to sleep state after being spinning for $Z$ time units.
3. If the machine is idle, power on the machine when the first pending job $j$'s flow time reaches $r_j + \hat{F}$.

**Analysis**

The algorithm has a weak competitive ratio of $(E_{tran} + Z) / \min\{E_{tran}, \hat{F} + Z\}$.
Online Algorithm 2: Minimize Energy Consumption under an Upper Bound of Larger Stretch $\alpha$

Idea

1. Run the job with the current largest stretch.
2. Update each job's artificial deadline according to the current largest stretch.
3. Execute jobs in an increasing order of their artificial deadlines.

Theorem

*Algorithm ONS is 2-competitive in terms of total energy consumption.*
### Table: Summary of our results on trading off energy consumption and flow time or stretch\(^4\)

<table>
<thead>
<tr>
<th>setting</th>
<th>restrictions</th>
<th>minimizing</th>
<th>performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>offline</td>
<td>bounded maximum flow time</td>
<td>energy consumption</td>
<td>(O(n^2))</td>
</tr>
<tr>
<td>offline</td>
<td>bounded energy consumption</td>
<td>maximum flow time</td>
<td>(O(n^2 \log F_{\text{max}}^<em>)); (F_{\text{max}}^</em>) is the optimal maximum flow time</td>
</tr>
<tr>
<td>offline</td>
<td>bounded largest stretch</td>
<td>energy consumption</td>
<td>(O(n^5))</td>
</tr>
<tr>
<td>offline</td>
<td>bounded energy consumption</td>
<td>largest stretch</td>
<td>(O(n^5 \log S_{\text{max}}^<em>)); (S_{\text{max}}^</em>) is the optimal largest stretch</td>
</tr>
<tr>
<td>online</td>
<td>bounded flow time</td>
<td>energy consumption</td>
<td>no worse than 2-competitive against non-idling adversaries</td>
</tr>
<tr>
<td>online</td>
<td>bounded stretch</td>
<td>energy consumption</td>
<td>2-competitive against adversaries</td>
</tr>
</tbody>
</table>

\(^4\) \(n\) is the number of jobs released
Future Work

- Find simpler and faster offline algorithms, or good approximation algorithms with (sub-)linear running time.

- Shrink the gaps between the lower bound and the upper bound of competitive ratio.

- Multiple-machine environment?
Thank you!

Questions?