Energy-Efficient Flow Time Scheduling: An Experimental Study

Jude-Thaddeus Ojiaku (speaker) * Daniel Thomas *
Prudence W.H. Wong *

1 Introduction

Power management has become a vital issue in the design of modern processors. A popular technology to reduce energy usage is *dynamic speed scaling* [5, 8] where a processor can vary its speed dynamically. This has been adopted in technologies like AMD's Cool 'n' Quiet [1] and Intel's Speedstep [6]. Running a job at a slower speed saves energy, yet it takes longer time and may affect the performance. The challenge arises from the conflicting objectives of providing good "quality of service" (QoS) and conserving energy. In the past few years, a lot of effort has been devoted to revisiting classical scheduling problems with energy in concern (e.g., [2, 4, 9]).

One commonly used QoS measurement is the total flow time (or equivalently average response time). The flow time of a job is the time elapsed from its arrival to completion. When energy is not a concern, the objective is to minimize the total flow time of all jobs, and it is well known that SRPT (shortest remaining processing time first) produces a schedule with the smallest possible flow time. To understand the tradeoff between flow and energy, Albers and Fujiwara [3] initiated the study of minimizing a linear combination of total flow and energy. An algorithm called AJC (active job count) has been proposed [3, 7], in which the speed of the processor is determined by the number of active jobs that have arrived but not completed yet.

We present an experimental study of scheduling to minimize flow time plus energy. We evaluate the performance of AJC along with two fixed speed heuristics of which one has prior knowledge of the job instance and the other does not. We study both single processor and multi-processor settings.

2 Experiment settings

We assume the inifinte speed model, i.e., a processor can vary its speed in the range $[0, \infty)$. When a processor runs at speed s, the energy consumption is given by s^α where $\alpha > 1$ [5] and s units of work is completed in each time unit.

We consider the variable speed function, $\mathrm{S}_{\mathrm{AJC}}$, and fixed speed functions S_{F1} and S_{D}. $\mathrm{S}_{\mathrm{AJC}}$ varies the speed based on the number of active jobs and is defined as $n^{\frac{1}{\alpha}}$, where n is the number of active jobs and α is set to 3. S_{F1} uses a fixed speed of 1. Both

*{J.Ojiaku, danieljt, pwong}@liverpool.ac.uk. Department of Computer Science, University of Liverpool, UK.

S_{AJC} and S_{F1} have no prior information on the jobs. On the other hand, S_D has some partial information of the job set. Given in advance the average job size and average inter-arrival time but not the details of each job, S_D determines one single speed based on the averages.

To test the algorithms, we generate input job sets randomly. We vary average job size and inter-arrival time. For each set of parameters we generate 100 random job sets. Each job set contains 1000 jobs. A job instance is *dense* if jobs arrive very frequently and contains large jobs, otherwise, it is *sparse*.

The simulation program is implememted in C# and the experiments are run on a dual-core CPU with clock speed of 2.9 GHz and 3 GB of main memory.

3 Results and observations

Job selection heuristics. We compare the performance of SJF and SRPT and the experiments show that the cost of using SJF is only about 1.1 times that of SRPT.

Effectiveness of speed scaling. In this experiment we demonstrate the effectiveness of speed scaling by comparing the performance of S_{AJC} to the fixed-speed heuristic, S_{F1}, which uses a fixed speed of 1. We observe that S_{F1} is similar to S_{AJC} when the job instance is sparse but is unable to cope with dense job instances and it could incur a cost of up to 30 times (Figure 1a) that of S_{AJC}. This agrees with the intuition that fixed speed heuristic does not give a good performance without any prior information of the job instance.

Speed scaling vs fixed speed. Here we compare the performance of S_{AJC} to fixed-speed heuristic, S_D. We observe that although S_D incurs higher costs than S_{AJC} in most job instances, it performs slightly better in some cases, as shown in Figure 1a. A possible explanation could be that S_D is less sensitive to the actual variation of the jobs.

Experiments on multiple processors. We examine the performance of several processor allocation heuristics that assigns jobs to the processor with the minimum total number of jobs, total size of jobs, total cost of active jobs or number of active jobs. We simulate multi-processor environments with 2, 4, 8 and 16 processors. We observe that increasing the number of processors reduces the total cost. The improvement is small for a small number of processors (the cost reduces by 10% with 2 processors) while significant improvement is observed for more processors (90% with 16 processors). The different job allocation heuristics have very small difference. We show the case for minimum number of active jobs in Figure 1b.

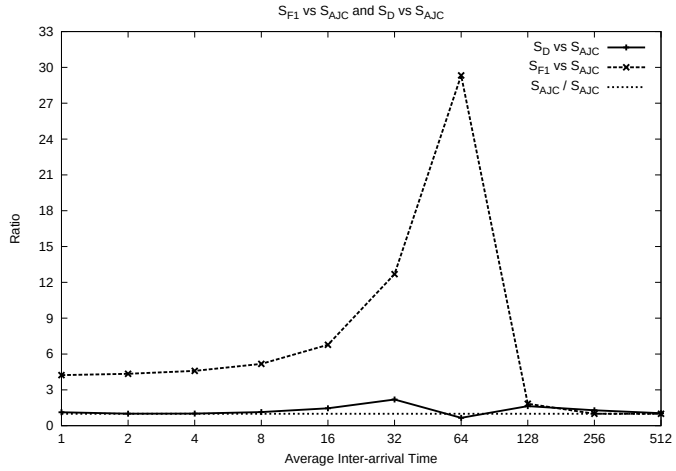
References

[1] Advanced Micro Devices. AMD Cool 'n' Quiet^{TM} technology. `http://www.amd.com/us/products/technologies/cool-n-quiet/Pages/cool-n-quiet.aspx`. Online; Accessed 08-Feb-2013.

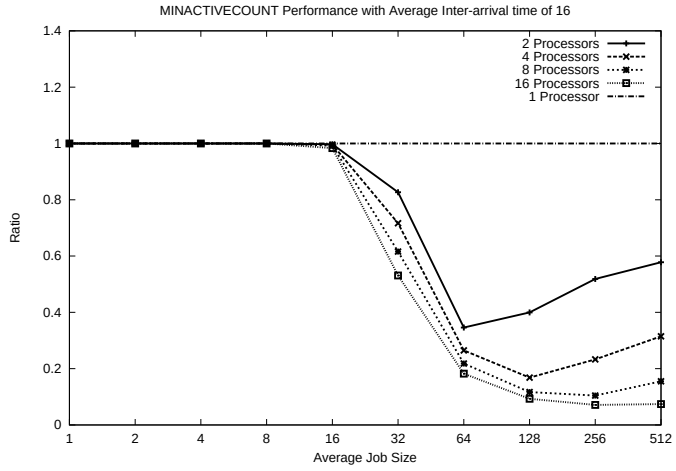
[2] S. Albers. Energy-efficient algorithms. *Communications of the ACM*, 53(5):86–96, May 2010.

[3] S. Albers and H. Fujiwara. Energy-efficient algorithms for flow time minimization. *ACM Transactions on Algorithms (TALG)*, 3(4), 2007.

[4] N. Bansal, T. Kimbrel, and K. Pruhs. Dynamic speed scaling to manage energy and temperature. In *IEEE Proc. FOCS*, pages 520–529, 2004.



(a) Average job size = 128



(b) Average inter-arrival time=16

Figure 1: (a) Performances of S_{AJC}, S_D and S_{F1}. (d) Multi- processor allocation based on number of active jobs.

[5] D. Brooks, P. Bose, S. Schuster, H. Jacobson, P. Kudva, A. Buyuktosunoglu, J. Wellman, V. Zyuban, M. Gupta, and P. Cook. Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. *Micro, IEEE*, 20(6):26–44, 2000.

[6] Intel Corporation. Enhanced Intel Speedstep Technology for the Intel Pentium M Processor. `ftp://download.intel.com/design/network/papers/30117401.pdf`. Online; Accessed 08-Feb-2013.

[7] T. Lam, L. Lee, I. To, and P. Wong. Speed scaling functions for flow time scheduling based on active job count. In *16th ESA*, pages 647–659. Springer, 2008.

[8] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced cpu energy. *Mobile Computing*, pages 449–471, 1996.

[9] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced cpu energy. In *36th IEEE FOCS*, pages 374–382, 1995.