Power and Speed Scaling to Trade Efficiency, Simplicity, Robustness and Fairness

Lachlan L. H. Andrew, (Swinburne, Australia), Minghong Lin (Caltech),
Ao Tang (Cornell) and Adam Wierman (Caltech)
landrew@swin.edu.au, {mhlin, adamw}@caltech.edu, atang@ece.cornell.edu

1. Introduction

Energy consumption is placing ever greater demands on most computing tasks, and multimedia is no exception. Current electronics can reduce its energy consumption by reducing the speed at which operations are performed, at the expense of a degraded user experience. Many techniques have been proposed for managing this trade-off, and a rigorous theoretical underpinning is beginning to emerge [1]--[9].

This letter will describe recent results for a "very soft real-time" setting, which does not impose deadlines on individual tasks, but seeks to minimize the weighted sum of energy consumption and average delay of each task, as done in [1]--[6]. This is suitable for, say, compressing multiple incoming video streams to disk; the average delay must be kept small to limit the memory requirements of the uncompressed video, but no specific deadlines are imposed. In this context, good performance for a given average load is possible by selecting the processing speed to depend only on the current number of uncompleted tasks.

This raises questions of how well the best speed scaling algorithms can minimize the energy plus delay, how the choice of speeds interacts with the choice of task scheduling, how sophisticated the hardware support must be, and whether optimizing this objective causes any undesirable side-effects. These will be discussed in turn.

2. Efficiency: Dynamic speed scaling

The most flexible model of speed scaling is of a system that can run at any arbitrary speed, but consumes more energy when running at a higher speed. Speed scaling decisions must be made in real time. The quality of an algorithm can be measured by comparing it with a hypothetical algorithm which has the benefit of perfect prescience, and knows the sizes and arrival times of all tasks which will ever occur. The maximum ratio of the cost incurred by the actual algorithm to that incurred by the optimal hypothetical algorithm is called the "competitive ratio".

In order to balance the cost of delay and the cost of energy, a natural approach is to set the speed such that these costs are equal, which we call "energy proportional" speed scaling. This turns out [2][3] to be within a constant factor of optimal with respect of competitive ratio.

Until recently, it was only known that the prescient algorithm was no more than three times as good as energy-proportional speed scaling, meaning that its competitive ratio is between 1 and 3. It has recently been shown [2] that energy-proportional speed scaling has a competitive ratio of exactly 2, when used with the optimal task scheduler, "SRPT" [10]. Moreover, it was shown that no other algorithms, from a wide class, can guarantee better performance. When used with a suboptimal Processor Sharing (PS) scheduler, the competitive ratio of energy-proportional scaling is again independent of the number of jobs [2]. This is in contrast to the performance without speed scaling, and suggests that the choice of scheduler becomes less important when speed scaling is allowed.

Although real algorithms cannot know what work will arrive in future, they can often benefit from statistical workload models. Specifically, when work comes from multiple sources, it can often be modeled as a Poisson process, which is only slightly more bursty than periodic workloads.

The optimal speed scaling for PS in this case was shown in [11] to have a form similar to energy-proportionality, provided that the power consumed increases polynomially with the speed. The optimal speeds for the optimal SRPT scheduler are difficult to calculate, but [2] shows that the speeds which are optimal for PS also work well for SRPT; in fact, both systems have a constant competitive ratio in the worst case, without making any assumptions about the workload. It is uncommon for such worst-case bounds to exist for algorithms optimized for statistical workloads, and is made possible in this case since the optimum is close to energy-proportionality.

3. Simplicity: Static power-aware speeds

It can be costly to change the processing speed too frequently, and designing a system to work at a single speed can reduce hardware complexity. This makes it interesting to see how well a system can perform if it always operates at a single speed, or sleeps when it is idle.

When the tasks are scheduled using processor sharing and the workload is Poisson of a known average rate, it was proven in [11] that this simple scheme performs within a factor of 2 of the optimal (non-prescient) dynamic scheme, and numerical results suggest that it is actually within around 10% of the performance.

The optimal SRPT scheduler is again much harder to study. In [2], an approximately optimal scheme was studied. The speeds were calculated separately using "heavy traffic" and "light traffic" approximations, and the slower of the two speeds is the appropriate choice. The more difficult heavy-traffic case built on recent results for the mean delay under SRPT from [12]. The optimal strategy depends on how heavy the tail of the task size distribution is; that is, on how likely very large tasks are. If the variance of the task sizes is infinite, then the optimal speeds depend only on the average load, while if the variance is finite then the optimal speeds also depend on the relative frequencies of large and small jobs. As for PS scheduling, this can be proven to give delay+energy performance within a constant factor of the optimal dynamic scheme, and numerical results indicate that very little is lost.

4. Robustness

Although static speeds can perform well when the load is what they were designed for, they can perform very poorly at lower or higher loads, or if work is more bursty than the Poisson model.

This is in stark contrast to the stochastically optimal speed scaling rules for SRPT and PS. As mentioned above, these schemes achieve within a constant factor of the performance of the hypothetical prescient algorithm for *arbitrary* workloads, including non-Poisson loads and loads with a different average amount of work arriving per second. (This constant factor depends on the load for which the speeds were optimized, but not on the actual workload experienced.) This shows that the main benefit from the increased complexity of dynamic speed scaling is robustness, rather than improved performance under expected conditions.

A different tradeoff can be achieved when the workload is known to be Poisson (i.e., slightly bursty), but the average rate is not known. Setting the speed to equal the number of unfinished tasks is a robust strategy, in the sense that it does not rely on knowledge of the average load. However, it was shown in [11] that if the power consumption is quadratic in the speed, then this algorithm has exactly the same cost as if a static speed had been chosen specifically for this particular load. That is, the result is provably within a factor of 2 of the optimal (non-prescient) scheme, and typically very much closer. However, this scheme can perform arbitrarily poorly when arrivals are more bursty than Poisson.

5. Fairness

If speeds are changed dynamically, then some tasks are run faster than others. This is not a problem if all tasks have an equal chance of being run quickly, but in practice some tasks are more likely to be run quickly than others. In particular, the tasks which are run quickly are those which are run when there are many tasks present in the system. These tend to be those tasks which are already treated well by the scheduler: If only one task is present, then the scheduler plays no role; it is when many tasks are present that biases in the scheduler take effect. For example, SRPT prefers to run short tasks ahead of long tasks, and speed scaling introduces a further bias against large tasks.

Using power-aware static speeds gives equal performance to all tasks, and results in fairness, but does not provide robustness against uncertainties in traffic. Processor sharing also treats all active tasks equally, and remains fair even when dynamic speed scaling is used.

6. Conclusions

Even delay-sensitive systems can save energy by running at low speeds during periods of low load. Techniques have been developed which provide provable efficiency, robustness and fairness guarantees. Any pair of these attributes can be achieved, but no current algorithm is able to provide all three.

References

[1] Susanne Albers and Hiroshi Fujiwara. Energy-efficient algorithms for flow time minimization, in ACM Transactions on Algorithms, v.3, 2007.

[2] Lachlan L. H. Andrew, Minghong Lin and Adam Wierman. Optimality, fairness and robustness in speed scaling designs. In ACM SIGMETRICS, New York, NY, USA, 2010. ACM.

[3] Nikhil Bansal, Ho-Leung Chan and Kirk Pruhs. Speed scaling with an arbitrary power function. In ACM-SIAM Syposium on Discrete Algorithms, 2009

[4] Nikhil Bansal, Kirk Pruhs and Cliff Stein. Speed scaling for weighted flow time. In ACM-SIAM Symposium on Discrete Algorithms, 2007

[5]Ho-Leung Chan, Jeff Edmonds, Tak-Wah Lam, Lap-Kei Lee, Alberto Marchetti-Spaccamela and Kirk Pruhs. Nonclairvoyant Speed Scaling for Flow and Energy, In Symposium on Theoretical Aspects of Computer Science, 2009

[6] Nikhil Bansal, Ho-leung Chan, Tak-wah Lam and Lap-kei Lee. Scheduling for speed bounded processors. In International Colloquium on Automata, Languages and Programming, 2008

[7] Frances Yao, Alan Demers and Scott Shenker. A scheduling model for reduced CPU energy. In Proceedings of the IEEE Syposium on Foundations of Computer Science, 1995.

[8] Kirk Pruhs, Patchrawat Uthaisombut and Gerhard Woeginger. Getting the best response for your erg. In ACM Trans. Algorithms, 2008.

[9] Susanne Albers. Energy-Efficient Algorithms: Algorithmic solutions can help reduce energy consumption in computing environs. Communications of the ACM, 2010.

[10] Linus E. Schrage and Louis W. Miller. The Queue M/G/1 with the Shortest Remaining Processing Time Discipline. In Operations Research, 1966.

[11] Adam Wierman, Lachlan L. H. Andrew and Ao Tang. Power-Aware Speed Scaling in Processor Sharing Systems, In *IEEE INFOCOM*, pages 2007—22015, Rio de Janeiro, Brazil, Apr 2009.

[12] Minghong Lin, Adam Wierman and Bert Zwart. The average response time in a heavy-traffic SRPT queue. In Workshop on MAthematical performance Modeling and Analysis, 2010.