Better Energy-Delay Tradeoff via Server Resource Pooling

Ioannis Kamitsos[†], Lachlan Andrew[*], Hongseok Kim[‡], Sangtae Ha[†], and Mung Chiang[†]

[†]Princeton University, USA
{kamitsos, sangtaeh, chiangm}@princeton.edu

[*]Swinburne, Australia
landrew@swin.edu.au

[‡]Sogang University, Korea
hongseok@sogang.ac.kr

***Abstract*—Multi-core architectures have supplanted single core schemes, because they reduce energy consumption by allowing lower clock frequencies. They provide the additional, less exploited benefit of allowing a trade-off between energy consumption and delay by turning off subsets of cores. We investigate what the benefits of this transition are in managing the trade-off between energy consumption and delay performance, and whether heterogeneity brings additional benefits to outweigh its increased complexity. To do this, we study optimal sleep policies in two settings: switching of homogeneous cores on a fast timescale, which models multiple cores in a CPU, and switching of heterogeneous cores on a slow timescale, which models servers of different generations in a data center. In the homogeneous case, we show the optimal policy is monotone hysteretic, and that the performance is less sensitive to load estimation errors at the design stage when at least two cores are present. In the heterogeneous case, we provide a low complexity algorithm to minimize the power requirements while providing a specified minimum processing speed.**

***Index Terms*—Multi-core, data centers, Pareto tradeoff, robustness, heterogeneous.**

I. Introduction

Resource pooling is a common trend as a computer system scales up. Multi-core computers, server farms, and generally, any system that processes jobs with multiple processing units with a queue storing jobs-in-waiting, share a common operational goal: to minimize job finishing time (which we call *delay*) and to minimize energy expenditure in finishing each job. However, these two objectives are intrinsically in conflict, and there exists an *energy-delay* tradeoff. For concreteness, we speak of investigating the tradeoff in the context of multi-core in a server, but the model applies to any case of resource pooling.

A useful technique for energy saving is turning off the cores when they are not needed. This simple method is highly effective because a significant amount of energy is wasted due to low utilization [1]; for example, a typical data center has utilization of 20%-30% on average [2], [3].

In resource pooling, all available resources are in one common pool and can be collectively controlled. We compare a c-core system with a benchmark of a single core which processes c times faster and uses c times more power than one of the c cores. We seek to answer the following questions: How much more efficient or simple is the tradeoff between delay and energy consumption in a multi-core architecture? Does this benefit require cores to be able to be turned off individually? Are further benefits possible by introducing heterogeneity in the speed and power of cores?

The design of an optimal "sleeping" policy is nontrivial for two reasons: (1) the best tradeoff point depends on the relative importance of energy-saving vs. fast job processing, and (2) it takes, energy, time or both to turn a core on or off. The latter implies that the optimal policy may be to remain on but idle rather than turning off, if there is a high chance of having to turn on again soon.

Energy saving mechanisms exploiting underutilization have long been investigated in the context of single core servers, e.g.[4], [5], [6], [7]. For example, [8] used dynamic programming to design the optimal policy for a stochastic (Poisson) workload that turns on/off a single core on the timescale of job arrivals, considering delay performance and energy consumption. Much less analytic work has been done in the multi-core setting, although see [9], [10], [11], [12]. This paper extends [8] to the case of multiple homogeneous cores and then considers on a slower time-scale the problem of determining the optimal combination of heterogeneous servers to have on.

For a single core, the optimal policy turns a server on when the queue occupancy exceeds a threshold, and off when the queue drains. In a c-core system, this could become much more complicated, with separate thresholds for every possible state of the cores. However, we show that with homogeneous cores, the policy retains the monotone hysteretic structure established in [8], and the optimal policy for a given load is characterized by c thresholds for turning cores on and c for turning them off.

Moreover, the added complexity is compensated by improved performance. It can give a better tradeoff between energy and delay, and the system becomes more robust to load estimation error, which improves performance under uncertain or time-varying load.

Large scale data centers today possess a scale and heterogeneity that deserves further study. In Section V, we investigate the case where the processors are heterogeneous in the sense that they are characterized by different processing speed and power consumption. Also, the scale is large enough that the timescale needs to be slower compared with job arrivals, and so the switching cost is ignored. The problem is formulated as a knapsack problem, which we solve via an efficient greedy algorithm. Numerical results show that the suboptimality gap does not exceed 6%.

Previous studies of policies for turning on/off for multi-server architectures include Deng and Purvis [13], which investigated the problem of parallelizing network applications without considering energy-related issues. Li and Alfa [14] considered an $M/M/c$ system that turns all servers on at the same time and turns them all off when they are all idle. They derived a closed form for the optimal threshold of turning on the servers, when the only costs are switching costs and delay. Their problem differs from the one considered here, since we do not require all cores to turn on and off simultaneously, or even to turn off at all, Instead, we impose an energy cost for each core which is on.

Our results are summarized below.

1) Theorem 1 shows that the optimal policy for controlling c homogeneous cores is characterized by $2c$ thresholds. This structure simplifies implementation.
2) We quantify the benefit from the flexibility of being able to turn cores off independently, by comparing the energy-delay tradeoff to that of a hypothetical single core server.
3) The optimal policy depends on the mean load. Section IV-C shows that, for more cores the performance is less sensitive to differences between the actual load and the load for which the policy was optimized.
4) Heterogeneity of cores can further sweeten the energy-delay tradeoff. Section V shows that a simple greedy algorithm can select subset of heterogeneous cores which gives performance within 6% of the optimal selection.

II. Homogeneous cores

Let us first consider homogeneous cores, which have the same processing capacity and associated energy cost. We further assume that cores can be turned on and off independently, and the workload is a Poisson process of exponentially sized jobs. We will formulate a Markov decision process (MDP) to minimize the objective of a weighted sum of delay, energy due to processing, and switching costs. Decisions on turning cores on and off are made on the timescale of job arrivals.

We consider a server with c independent cores, a finite buffer of size B and $c+1$ modes. The mode $(0, 1, 2, \ldots, c)$ denotes the number of cores that are ON. The state space is $\Omega = \{0, 1, 2, \ldots, c\} \times \{0, 1, 2, \ldots, B\}$. The system is in state $i = (W, Q)$ if there were W cores ON in the *previous* time interval and *currently* there are Q jobs in the system. A time interval represents the time between two successive states. The action $w \in \mathcal{F} = \{0, 1, \ldots, c\}$ determines the number of cores ON in the current time interval.

Every core consumes a constant power P_{on} when ON, and no power when OFF. Turning a core from OFF to ON or vice versa consumes energy E_{ch}. We compare against a hypothetical single core server consuming power cP_{on}, with switching cost cE_{ch}.

We assume that jobs arrive at the server according to a Poisson process with rate λ. Job service times are exponentially distributed with mean $1/s$, where s is the service rate of one core. If a core is available ($Q \leq p$), the job enters service immediately. Each job is processed by one core at a time.

The memoryless property of the interarrival times allows us to formulate the problem as a Markov Decision Process (MDP). The solution to the MDP is a *policy* denoted by p, indicating how many cores should be ON (the action) in each state. The action in state $i = (W, Q)$ is denoted by $p(i)$, or, equivalently, $p(W, Q)$. By the Markov structure, the policy only changes when a job arrives or departs. Note that the action is deemed to occur immediately after the transition to a given state, so that the number of servers on in state (W, Q) is $p(W, Q)$, not W. This means that the current number of cores ON is not considered part of the state.

The state evolves as follows:

$$W(t+\tau) = p\left(W(t), Q(t)\right), \tag{1}$$

$$Q(t+\tau) = \begin{cases} Q(t)+1, & \text{if an arrival occurs,} \\ Q(t)-1, & \text{if a departure occurs.} \end{cases} \tag{2}$$

where t denotes the current time and $t+\tau$ refers to the time of the next event. Then, the transition probabilities at state (W, Q) with policy $p(W, Q)$ and $0 < Q < B$ are:

$$Pr[(W,Q) \to (p(W,Q), Q+1)] = \frac{\lambda}{\lambda + sp(W,Q)}, \tag{3}$$

$$Pr[(W,Q) \to (p(W,Q), Q-1)] = \frac{sp(W,Q)}{\lambda + sp(W,Q)}. \tag{4}$$

The stage cost $g(i, p(i))$ at state i under policy $p(i)$ consists of two components; the first, $g_1(i, p(i))$, which represents the running cost per unit time, is a weighted sum of the power cost due to the active cores and the delay cost incurred by the jobs waiting to be processed. The delay cost is directly proportional to the number of jobs waiting. Hence,

$$g_1(i, p(i)) = (p(i)P_{\text{on}} + rQ_i), \tag{5}$$

where r is the tradeoff parameter between the energy cost and the cost of delay.

The second component, $g_2(i, p(i))$, captures the switching cost, including the power spent for turning a core on or off and a penalty modeling the delay cost incurred. It is given by

$$g_2(i, p(i)) = |W_i - p(i)|E_{\text{ch}} \tag{6}$$

where E_{ch} is the switching cost.

Since the duration of a stage depends on the policy, it is convenient to use uniformization [15] to convert this continuous time problem into a discrete time one. The cost per uniformized transition is given by

$$\hat{g}(i, p(i)) = \frac{1}{\beta + v}(p(i)P_{\text{on}} + rQ_i) + |W_i - p(i)|E_{\text{ch}}, \tag{7}$$

where the uniformized rate is $\beta + v$ for some $\beta > 0$, and $v = \lambda + cs$.

Since we are interested in the remaining cost over all future time, the cost-to-go will be infinite. To enable us to optimize the time-average cost, we approximate this objective by a discounted objective, with a discount rate α close to 1. In the context of on-line adaptation of the policy, this discounting also provides robustness against uncertainty in future loads.

We write $\alpha = v/(\beta + v)$ and select a small $\beta << v$ to ensure $\alpha \approx 1$.

Our objective is then to minimize the average discounted sum of costs [15], given by

$$V(i) = \min_{p(i)} \left\{ \frac{g_1(i,p(i))}{\beta + v} + g_2(i,p(i)) + \alpha \sum_{j \in \Omega} \hat{M}_{i \to j}^{p(i)} V(j) \right\}, \tag{8}$$

where $\hat{M}_{i \to j}^{p(i)}$ denotes the uniform transition probability going from state i to state j under policy $p(i)$, given in [15], whereas $M_{i \to j}^{p(i)}$ denotes the respective nonuniform transition probability. Also, v_i represents the nonuniform total transition rate from state i. The average discounted sum of costs consists of the current stage cost plus the discounted average of all future costs.

The optimal solution to our problem as well as other related results will be presented in the next section.

III. Optimal Control Policy

The following definitions are important for the main analytic results.

Definition 1: A policy p is called hysteretic if for all $a \in \mathcal{F}$, $p(a, Q) = \gamma$ implies $p(\gamma, Q) = \gamma$.

Definition 2: [16] A hysteretic policy p is called monotone if there are l_α, u_α, $\alpha = 1, 2, \ldots, |A| + 1$, where $|A|$ is the set of service levels, with $l_\alpha \leq u_\alpha$ for $\alpha = 1, 2, \ldots, |A| + 1$, $l_\alpha \leq u_{\alpha+1}$ for $\alpha = 1, 2, \ldots |A|$, $l_\alpha = u_\alpha = 0$ and $l_{|A|+1} = u_{|A|+1} = \infty$, such that for all $(i, d) \in \Omega$, we have

$$p(i,d) = d, \text{if } l_d \leq i < u_{d+1}$$
$$p(i,d) = p(i, d+1), \text{if } i \geq u_{d+1}$$
$$p(i,d) = p(i, d-1), \text{if } i < l_d$$

The structure of a monotone hysteretic policy is shown in Fig. 1. Such a policy can be summarized by only $2c$ thresholds (c for turning cores on, and c for turning cores off), instead of $\max(W) \times \max(Q)$ individual policies. This makes policies much more efficient to calculate and store. This allows policies to be pre-computed, and an appropriate policy to be chosen for the current job arrival rate.

Based on the problem formulation discussed in the previous section the following result can be shown using the same techniques as in [8].

Theorem 1: The optimal policy for the above MDP is a monotone hysteretic policy.

Given that most data centers are underutilized, it is important to understand the benefits of the multi-core approach over the single core one for low traffic load.

Proposition 1: For low traffic load ($\lambda \to 0$) the ratio of the mean cost in the single core case over the mean cost for a multi-core case asymptotically converges to c.

Proof: This proposition can be easily proven by calculating the stationary probabilities of states (0,1) and (1,0) for both single core and multi-core cases, given that for $\lambda \to 0$ we cannot have more than one job in the buffer. ■

The optimal policy can be found by value iteration [15].

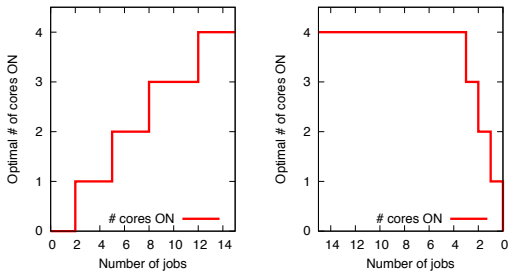

Fig. 1. Optimal policy for a 4-core server under traffic load 20%. The graph to the left refers to the ON policy while the graph to right refers to the OFF policy. Note the monotone hysteretic structure of the policy, with the ON and OFF thresholds.

IV. Numerical Evaluation

A. Optimal Policy Structure

Initially, we will numerically evaluate the structure of the optimal sleeping policy. The following simulations consider jobs arriving to a 4-core server with power consumption equal to $P_{\text{on}} = 62.5\,\text{W}$ per core. The energy vs delay tradeoff parameter r was 5 J/job and the switching cost was roughly $E_{\text{ch}} = 100\,\text{J}$. We consider large jobs, with average duration 2 seconds. Up to B=100 jobs can be buffered awaiting service. Initially, we tested the system under relatively low traffic load (20% of peak), which models an underutilized server in a data center. The optimal policy, presented in Fig. 1, is clearly monotone hysteretic, as Theorem 1 predicted. It is characterized by one ON threshold per core. With the traffic load of 20%, it is optimal to turn on 1–4 cores when there are $\theta_1 = 2$, $\theta_2 = 5$, $\theta_3 = 8$ and $\theta_4 = 12$ jobs, respectively. When the load is high, it is typically optimal to turn cores on at lower occupancies, since future arrivals are more likely.

Note that when the load is moderately high, it is never optimal to turn off all cores. This is useful from an operational point of view, since at least one core is then always available to process administrative tasks, such as enqueueing jobs. For a single core, this only occurs for very high load, when no energy saving is possible.

B. Energy/delay tradeoff

It is interesting to observe the Pareto optimal tradeoff between energy consumption and average delay. Fig. 2 compares the performance of a 4-core and 8-core with that of a hypothetical single core server, under traffic load of 20%. The improved performance of the multi-core systems comes from the finer granularity of control. In particular, the use of a 4-core system results in reducing of energy consumption by almost 30% for given average delay compared to the single core architecture.

Interestingly, there are two cases in which additional cores might present a disadvantage. The root cause of these is the well known fact that an M/M/c queue has a higher mean delay than an M/M/1 queue with a server c times as fast, since the aggregate speed is lower at low queue occupancies. First, if the activation policy is suboptimal, such as requiring all cores to switch on or off simultaneously, then multi-core loses its

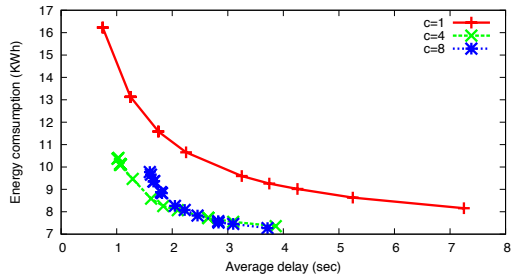

Fig. 2. Accumulated energy over 5×10^5 seconds vs delay, for 1, 4 and 8 cores and variable cost of delay, at 20% load. Increasing the number of cores improves performance compared to single core case. When the queue occupancy is low, having slower servers outweighs the finer control.

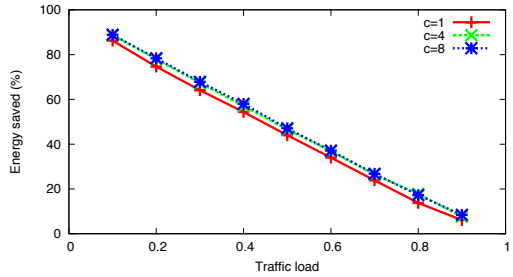

Fig. 3. Fraction of energy saved over traffic load for different number of cores. Energy savings are significant for lower loads, and increase when we control more cores.

benefits that come from flexible resource allocation. Second, if the expected number of jobs in the system is much lower than the number of cores, then the disadvantage of having slower servers outweighs the finer control; this occurs at the high-energy/low-delay point in the tradeoff when there are many cores. However, when job sizes are sufficiently heavy tailed, an M/G/c queue has lower mean delay than a faster M/G/1 queue (even without sleeping) [21]; we expect that sleep considerations will never decrease the optimal number of cores. See also [22] for a discussion of tails of response times.

Fig. 3 presents how the fraction of energy saved varies with traffic load for different number of cores. The energy savings are in comparison to the case where all cores are ON, without sleeping enabled. Also, the application was assumed to be rather delay tolerant, with $r = 5$ J/job. For all loads, the higher the number of cores we can control, the higher the fraction of energy saved. Naturally, for all numbers of cores, the higher the traffic load the lower the energy savings.

C. Robustness

In practice, the actual load is not known a priori, and its estimation is often noisy and unreliable. *Robustness* against misestimating the traffic load is as important as Pareto optimality of the tradeoff. For example, it was recently shown in [17] that increased dynamic flexibility in the power/speed tradeoff significantly increases robustness to such errors. We demonstrate that benefits to robustness are obtained by being able to turn on or off multiple cores as server resources

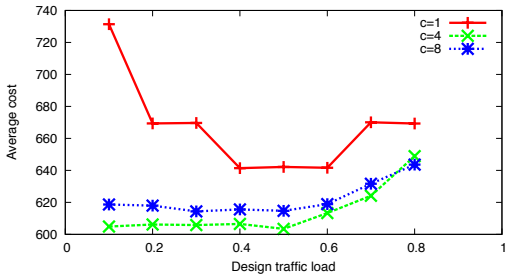

Fig. 4. Optimal cost under traffic load 50% over design traffic load. The flatter curve for multi-core shows lower sensitivity to the load estimate.

are pooled. Adopting a "delayed-off" policy such as the one described in [10] can increase the system's robustness, however it doesn't allow for jobs to be queued, and such a policy is only optimal in the limit of large systems.

Suppose the system is designed for a particular load ρ_0 and the optimal average cost when it operates under this load is V_0. Also suppose that the system is operated with an actual load of $\rho \neq \rho_0$ and the average cost under this load is V. Then we say that a system is more robust than another if $\Delta V = \max\{V_1, V_2, ...\} - V_0$ is less for the first system, where the maximum is taken among all possible V. This translates to a flatter average cost curve.

The following experiment demonstrates this. Suppose we have a system with c independent cores. We calculate the optimal policy for different traffic loads and then we let the system evolve under traffic load 50%. Then we calculate the optimal expected cost as

$$\mathbb{E}[V] = \mathbb{E}[E] + r\mathbb{E}[D] \tag{9}$$

where $\mathbb{E}[E]$ stands for the average energy consumption and $\mathbb{E}[D]$ stands for the average delay. Fig. 4 shows how the optimal expected cost under traffic load 50% varies with design traffic load (the optimal policy for each case is calculated under the respective traffic load).

It is now readily seen that having multiple cores improves robustness to traffic load misestimation. In particular, the variation from the optimal cost achieved under traffic load 50% decreases, giving a flatter plot. It can be observed that the finer granularity of control provides an almost 75% "more robust" system. Note, however, that having 8 cores does not further improve the robustness over having 4 cores. Only a small amount of flexibility is required to obtain most of the benefits. Also, given that the expected queue occupancy is low at traffic load of 50%, controlling 4 cores results in lower cost than when controlling 8 cores at that traffic load. As explained earlier, having slower cores at lower queue occupancies outperforms the finer granularity.

V. Heterogeneous servers

In the previous sections we studied the case of homogeneous cores/servers where we make optimal decisions on the timescale of arrivals/departures. However, when we have hundreds or thousands of processors (e.g. in large scale data centers) it is somewhat impractical to make decisions for turning on/off different processors every time we have an arrival or a departure. In this section, we consider the case

of a large scale data center, containing several different types of processors, with many processors of the same type. Our objective is to find the optimal combination of processors that can be put to sleep in order to minimize the total power consumption given a fixed total processing speed requirement. The decision on which processors need to be on, is not made on an arrival/departure timescale, but on a slower one.

Specifically, suppose we have a collection $\mathcal{N}$ of N different types of heterogeneous processors, with discrete processing speeds $s_1, s_2, \ldots, s_N$, measured in packets per second (pps), and power consumption $P_1, P_2, \ldots, P_N$. Also suppose that we have c_j identical copies of processors of type $j, j = 1, 2, \ldots, N$, so that $c = \sum_{j=1}^{N} c_j$. Then we want to select w_j processors of each type j that can stay ON, such that the total power consumption is minimized given that the total processing speed is higher than a fixed speed requirement s_0. In other words, our objective is to solve the following optimization problem.

$$\min_{w_j{}_{j=1}^{N}} \quad \sum_{j=1}^{N} P_j w_j \tag{10}$$

$$\text{s.t.} \quad \sum_{j=1}^{N} s_j w_j \geq s_0 \tag{11}$$

$$0 \leq w_j \leq c_j, \tag{12}$$

$$w_j \text{ integer}, j = 1, 2, \ldots, N \tag{13}$$

$$\tag{14}$$

A. Greedy algorithm

When the number of processors is large, the following greedy algorithm gives an almost optimal solution.

Algorithm 1 Greedy Algorithm

Sort types $j = 1, 2, \ldots, N$ in increasing order of P_j/s_j.
Find a threshold k^* satisfying
$k^* = \min\left\{k : \sum_{j=1}^{k-1} s_j c_j \leq \sum_{j=1}^{N} s_j c_j - s_0\right\}$.
if $P_j/s_j < P_{k^*}/s_{k^*}$ **then**
  $w_j = c_j$.
else
  if $P_j/s_j > P_{k^*}/s_{k^*}$ **then**
    $w_j = 0$.
  end if
else
  $x_{k^*} = \left\lfloor c_{k^*} - \frac{\sum_{j=1}^{N} s_j c_j - s_0 - \sum_{j=1}^{k^*-1} s_j c_j}{s_{k^*}} \right\rfloor$.
end if
Return $w_j \ \forall j = 1, 2, \ldots, N$.

The complexity of the greedy algorithm is $O(N \log N)$, from the sorting.

This formulation is a generalization of that of [18], which had unlimited servers of each type. It is also a special case of the multi-dimensional knapsack problem considered in [19]. Both of these papers consider greedy algorithms closely related to the above. In [18], conditions were found for the greedy algorithm to be optimal, and an exact expression was found for the suboptimality in other cases. The suboptimality is bounded by the quantization resolution; that is, the cost of a single machine. In the context of large data centers, this approximation error becomes negligible.

B. Numerical evaluation

We now investigate how well the greedy algorithm presented above performs compared to the optimal solution that can be found via a dynamic programming approach. Suppose we have a data center consisting of several type processors (multiple processors per type), which are given in Table I [20].

TABLE I
PROCESSOR TYPES

Type	Description	Speed (kpps)	Power (W)
1	AMD Athlon 64 1 core, 2.0 GHz, RAM: 667 MHz	600	70
2	AMD Athlon 64 1 core, 2.2 GHz, RAM: 200 MHz	800	80
3	AMD Opteron 2 core, 1.8 GHz, RAM: 400 MHz	900	190
4	AMD Opteron 4 core, 1.8 GHz, RAM: 400 MHz	1400	205
5	Xeon 5050 "Dempsey" 2 core, 3.0 GHz, RAM: 533 MHz	1600	250
6	Xeon 5050 "Dempsey" 2 core, 3.0 GHz, RAM: 533 MHz	1800	280

Assuming that there are 10 servers per type, Fig. 5 presents the greedy and optimal solutions for different speed requirements varying from 0.8 Mpps up to 70 Mpps, which corresponds to a speed almost as high as if all the servers were turned on. It can be observed that the suboptimality gap is small for the whole range of speed requirements, and it never exceeds 6%. The computational efficiency of the greedy algorithm comes from the fact that we have multiple servers of the same type.

Finally, Fig. 6 shows the gain in terms of power consumption from using the heterogeneous setup described in Table I (with 10 processors per type) over a homogeneous setup where each server consumes the average power and runs at an average speed. This gain is much more significant when the speed requirement in the data center is lower. Hence, when the speed requirement is low (something required to save energy), using the heterogeneous setup described in the previous section is much more efficient than using an "average" homogeneous setup. Knowing that heterogeneity helps may also lead to savings, since it encourages the operator to keep slower old-generation equipment around longer.

VI. CONCLUSIONS

The optimal sleep policy for controlling multiple homogeneous cores has a useful monotone hysteretic structure. Unlike the single core case, it can be optimal to retain one or more cores ON even at moderate load; that is important for implementations, since one core must often be active for housekeeping tasks.

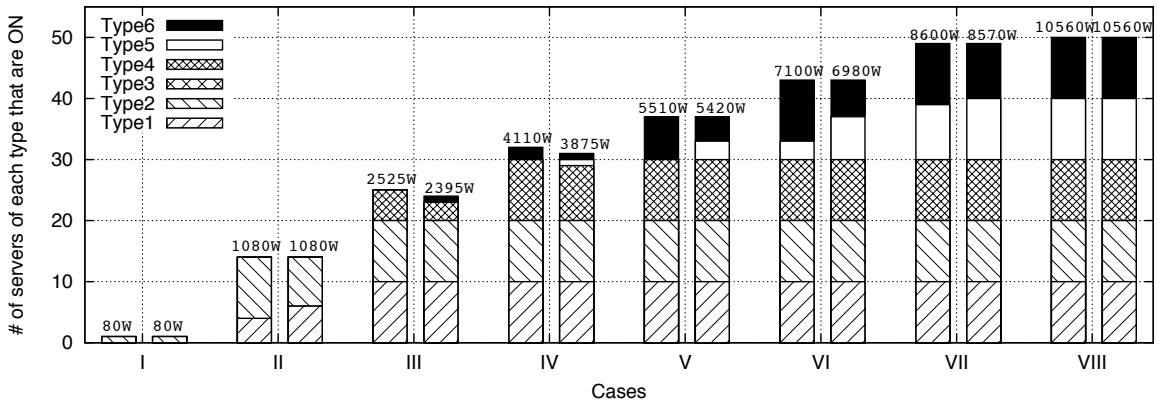

Fig. 5. Greedy (left) and optimal (right) solution for different speed requirements. Case I: $s_0 = 0.8Mpps$, Case II:$s_0 = 10Mpps$, Case III:$s_0 = 20Mpps$, Case IV:$s_0 = 30Mpps$, Case V: $s_0 = 40Mpps$, Case VI: $s_0 = 50Mpps$, Case VII: $s_0 = 60Mpps$, Case VIII: $s_0 = 70Mpps$

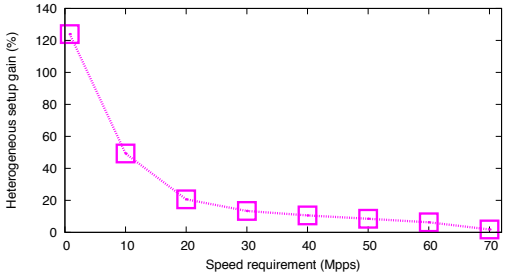

Fig. 6. Gain from using the heterogeneous setup described in Table I (10 processors of each type) over an average homogeneous setup where each server consumes the average power and runs on an average speed.

The performance of a homogeneous multi-core system is also less sensitive to mismatch between the actual load and the load for which it is optimized. Having two cores provides almost as much benefit as having a large number of cores. This benefit vanishes if all cores must be switched on or off simultaneously.

Systems with heterogeneous cores are much more difficult to manage dynamically. Instead of switching cores frequently, substantial savings can be obtained by setting the number of active cores to match the average load. For this, a simple greedy algorithm is sufficient. In this context, heterogeneity significantly reduces power consumption compared to a system in which all servers have the average performance, especially for underutilized data centers.

VII. ACKNOWLEDGEMENTS

This work was in part supported by Google and in part by Princeton Grand Challenge grant.

REFERENCES

[1] J. Mogul, "Improving energy efficiency for networked applications", in *Architectures for Networking and Comunications Systems (ANCS)*, 2007.
[2] P. Bohrer, E. Elnozahy, T. Keller, M. Kistler, C. Lefurgy, and R. Rajamony, "The case for power management in web servers", in *Power Aware Computing*, January 2002.
[3] D. Meisner, B.T. Gold and T.F. Wenisch, "PowerNap: Eliminating Server Idle Power", In *Proc. Int. Conf. Arch. Support Prog. Lang. Op. Sys.*, pp. 205-216, 2009.
[4] L. Chiaraviglio, M. Mellia and F. Neri, "Reducing Power Consumption in Backbone Networks", In *Proc. ICC*, Dresden, Germany, 2009.
[5] K. Pruhs, P. Uthaisombut and G. Woeginger, "Getting the best response for your erg", In *Scandinavian Worksh. Alg. Theory*, 2004.
[6] A. Wierman, L. L. H. Andrew and A. Tang, "Power-Aware Speed Scaling in Processor Sharing Systems", In *Proc. IEEE INFOCOM*, April 2009.
[7] N. Vasic and D. Kostic, "Energy-Aware Traffic Engineering", In *EPFL Technical Report NSL-REPORT-2008-004*, 2008.
[8] I. Kamitsos, L. L. H. Andrew, H. Kim, and M. Chiang, "Optimal Sleeping Patterns for Serving Delay Tolerant Jobs", in *Proc. e-Energy*, pp. 31-40, Passau, Germany, April 2010.
[9] S. Albers, F. Müller and Swen Schmelzer, "Speed scaling on parallel processors," in *Proc SPAA*, 2007.
[10] A. Gandhi, V. Gupta, M. Harchol-Balter, and M. A. Kozuch, "Optimality analysis of energy-performance trade-off for server farm management," *Perform. Eval.*, vol. 67, no. 11, pp. 1155 – 1171, 2010.
[11] K. Pruhs and C. Stein, "How to schedule when you have to buy your energy," in *Approx., Randomiz. Comb. Opt. Alg. Techniq.*, LNCS 6302, pp 352–365, 2010.
[12] M. Lin, A. Wierman, L. L. H. Andrew and E. Thereska, "Dynamic right-sizing for power-proportional data centers," *Proc. IEEE INFOCOM*, 2011.
[13] J. Deng and M. Purvis, "Queueing Analysis for Multi-Core Performance Improvement: Two Case Studies", in *Proc. Australasian Telecommunication Networks and Applications Conference*, 2007.
[14] W. Li and A. Alfa, "Optimal Policies for M/M/m Queue with Two Different Kinds of (N, T)-Policies", in *Naval Research Logistics*, vol. 47, pp. 240-258, 2000.
[15] D.P. Bertsekas, "Dynamic Programming and Optimal Control", vol. 2, Athena Scientific, 2007.
[16] S.K. Hipp, and U.D. Holzbaur, "Decision Processes with Monotone Hysteretic Policie", *Op. Res.* vol. 36, no. 4, pp. 585-588, Jul-Aug 1988.
[17] L. L. H. Andrew, M. Lin, and A. Wierman, "Optimality, fairness and robustness in speed scaling designs," in *Proc. ACM SIGMETRICS*, 2010.
[18] M. J. Magazine, G. L. Nemhauser and L. E. Trotter, "When the greedy solution solves a class of knapsack problems", In *Op. Res.*, vol. 23, no. 2, Mar.-Apr., 1975.
[19] Y. Ak cay, H. Li and S. H. Xu, "Greedy algorithm for the general multidimensional knapsack problem", In *Ann. Oper. Res*, vol. 150, pp. 17-29, 2007.
[20] R.Bolla, R. Bruschi and A. Ranieri, "Green support for PC-based software router: performance evaluation and modeling", In *Proc. ICC'09*, pp 2200-2205.
[21] A. Scheller-Wolf and R. Vesilo, "Structural interpretation and derivation of necessary and sufficient conditions for delay moments in FIFO multiserver queues", *Queueing Systems*, vol. 54, no. 3, pp. 221-232, 2006.
[22] J. Nair, A. Wierman and B. Zwart, "Tail-robust scheduling via limited processor sharing", *Perform. Eval.*, vol. 67, no. 11, pp. 978–995, Nov, 2010.