Exploring the Economic Viability of Sustained Computational Sprinting at the Datacenter Level

Aleis Murphy, Nicola Felice, Matthew Faw, and Elijah Cole

I. INTRODUCTION

Datacenters spend a significant amount of money on electricity: in 2006, datacenters were responsible for 1.5% of the total electricity consumption in the U.S., at a cost of $4.5 billion [1]. Since the increase in server computational power has been outpacing the increase in server energy efficiency, there is good reason to expect these expenditures to continue to grow [2]. Datacenter operators have a clear incentive to reduce these substantial costs. A natural solution is to deploy more efficient or more energy-proportional severs, but this is not a practical short-term solution both for technological reasons and because datacenters already exist that have tremendous investment in older server hardware. Fortunately, there are a variety of power-saving techniques that can be employed without buying new servers. Some consider the datacenter as a whole [3], while others specifically target power consumption by servers [4] [5] [6] or network elements [7] [8]. We will propose an approach that targets CPU power consumption. Many of these strategies seek to reduce total power consumption, whereas we aim to more effectively use the currently available power for computation. In particular, we will determine whether a simple system-level *computational sprinting* strategy can increase performance per watt per dollar.

Computational sprinting is the practice of temporarily operating a chip beyond its sustainable thermal budget to improve performance in some way. Computational sprinting was initially proposed in [9] (and experimentally validated in [10]) as a way to decrease mobile device response latency by temporarily activating cores that were otherwise unpowered due to thermal constraints (so-called *dark silicon* [11]). Computational sprinting was recently extended to the datacenter level as a way to briefly boost the average computing performance of a cluster during spikes in demand [12]. In some of the most recent work on datacenter-level sprinting, Fan et al. took a game-theoretic approach to the problem of allocating resources for sprinting among servers constrained by their shared power supply [13].

Note that we have so far left the actual implementation of computational sprinting vague. This is because computational sprinting is an idea with many different realizations in practice. We have already mentioned one technique – lighting up dark silicon, or enabling cores that are usually left unpowered in nominal operation. Increasing the number of active cores increases performance, but also increases power draw. A more involved sprinting technique involves switching between a high-performance microarchitecture and a low-power microarchitecture such as in [14]. One of the most popular implementations is dynamic voltage and frequency scaling (DVFS) [15]. By increasing the CPU frequency and processor voltage, we increase both performance and power consumption. Similarly, we could decrease processor voltage and frequency to reduce power consumption. Vogeleer et al. propose that there is a well-defined frequency that minimizes power draw in mobile devices [16]. However, in [17] it is shown that the effectiveness of DVFS in any given case is a function of the system conditions and the particular workload under consideration, so the relationship between power consumption and core voltage and frequency can be more complex. In addition to the individual techniques we've discussed, one could also imagine deploying various combinations of these techniques since they are largely orthogonal. This induces a very rich design space, of which we shall only explore a small region: DVFS and *core scaling*, or varying the number of active processor cores.

Regardless of the particular implementation, we tend to think of computational sprinting as an extreme state that can only be safely entered occasionally. Of course this is true at the level of the individual sprinter. However, suppose we consider a slightly higher level of organization, a group of sprinters working together, which we'll call a *sprinting cycle*. With a large enough group (determined by the durations of the sprinting and recovery phases) we should be able to have one member sprinting at all times in a round-robin fashion. This should increase the amount of work getting done, but will also increase power draw. This management strategy is perhaps the simplest possible way to organize sprinting cycles, but it could serve as a proof of concept to motivate more sophisticated implementations of the same basic idea.

To be more specific, our contributions will be as follows:

- We characterize the performance of a broadly representative set of Spark workloads under various computational sprinting configurations.
- We describe sprinting cycles, a simple strategy that would allow for indefinite computational sprinting at the cluster level.
- We evaluate this strategy in terms of the trade-offs between power, performance, and cost. In particular, we will undertake a total cost of ownership (TCO) analysis to elucidate any financial gains or costs associated with operating a system in an indefinite sprinting pattern and then compare sprinting cycles to nominal operation in terms of performance per watt per dollar.

II. METHODS

A. Hardware Configuration

All testing was performed on a commodity-class PowerEdge R210 II server equipped with 4 x 3.10 GHz Intel Xeon CPU E3-1220 V2 and 4 x 2 GiB DIMM DDR3 Synchronous 1600 MHz memory [18]. Workloads were permitted to use up to 6 GiB of RAM, so 2 GiB were reserved for the Ubuntu 12.04.5 operating system.

Benchmark	Category	Dataset	Data Size
DecisionTree	Classification	kdda2010[19]	2.5G
Correlation	Statistics	kdda2010[19]	2.5G
PageRank	Graph Processing	soc-LiveJournal[20], [21]	1.0G

TABLE I
REPRESENTATIVE SUBSET OF THE SPARK WORKLOADS FROM FAN ET AL. [13].

B. Sprinting Methodology

We utilized two computational sprinting techniques: DVFS and scaling the number of active cores. DVFS was implemented by manually setting a particular frequency through the `userspace` governor in the `cpufreq-set` utility. Core scaling was implemented via CPU hotplugging, a Linux feature which allowed us to disable up to three of the four installed cores.

C. Workload Characterization

It is difficult to predict ahead of time how our two sprinting parameters (number of cores, core frequency) should be set to maximize performance per watt for a given workload. For this reason, we took an experimental approach to exploring the computational sprinting design space for several workloads implemented in Apache Spark.[1]

In [13], a variety of workloads from Spark's machine learning library[2] were characterized in terms of how frequently they sprint under the game-theoretic resource sharing model developed in that paper. These probabilities are reproduced in Figure 1. We can see that some workloads sprint at almost every opportunity; these are workloads that reap approximately the same benefit from sprinting regardless of when they are allowed to do so. Other workloads prefer to sprint more selectively, as they benefit from sprinting only at certain times. We chose to profile a representative subset of these workloads, which is shown in Table I to consist of `Correlation`, `DecisionTree`, and `PageRank`. They are representative in two senses. First, these workloads have high, medium, and low probabilities of sprinting, respectively. Secondly, we chose them so that no two were from the same category of computational technique. For each workload, we used 1, 2, 3, or

[1]http://spark.apache.org/

[2]http://spark.apache.org/mllib/

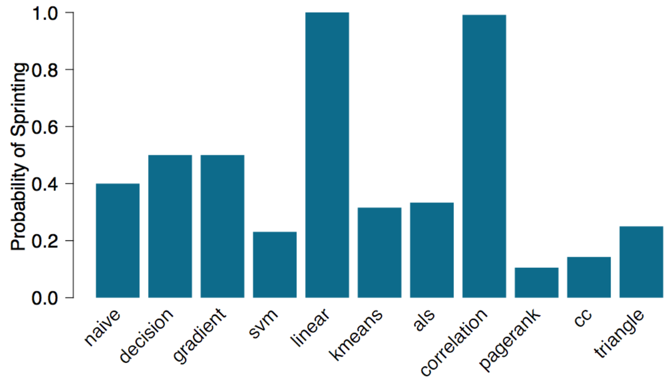

Fig. 1. Probability of sprinting. (from [13])

4 cores operated at 1.8, 2.5, or 3.1GHz, for a total of 12 different experimental configurations per workload. Each workload was profiled in each configuration until either the process finished or until 10 minutes had elapsed. Performance and power metrics were recorded at 1 second intervals with Intel's Performance Counter Monitor (PCM) utility. [3] In particular, over a runtime of T seconds, we measured retired instructions $I(t)$ and CPU energy in joules $E(t)$ at intervals of $\Delta t = 1s$. We then computed the average number of instructions per joule α as

$$\alpha = \frac{1}{T}\left(\sum_{t=1}^{T} \frac{I(t)\cdot\Delta t}{E(t)}\right).$$

Note that this results in units of instructions per joule. Since a watt is a joule per second, these units are equivalent to instructions per second per watt. We take instructions per second to be our measure of performance, so these units are *performance per watt*. We somewhat arbitrarily define *nominal operation* as one core operating at 1.8GHz. The process described in this section allows us to determine what the best sprinting system configuration is for each workload. Later in the paper we will present results in terms of average performance per watt for sprinting in the best configuration, as compared to performance per watt in nominal operation.

[3]https://software.intel.com/en-us/articles/intel-performance-counter-monitor

D. Sprinting Cycles

In this section when we refer to "sprinting" for a given workload, we mean sprinting in the best way for that workload as determined by the previously described workload characterization. Suppose we have a number of machines, each of which is able to sprint for S seconds, but must then recover for nS seconds for some positive integer n. Fan et al. considered a chip whose properties allow for $n = 2$, which is the timing we shall assume as well. [13] A set of $n + 1 = 3$ machines, which we'll call a *sprinting cycle*, can in principle operate with one of its machines sprinting at all times for an indefinite duration. The machines in the sprinting cycle sprint in simple round-robin order. It is important to note that profiling was done on a single machine and that later results about sprinting cycle characteristics are the result of extrapolating single-machine results to a sprinting cycle of size three.

E. Total Cost of Ownership

First we will describe the general process of our TCO calculations and our assumptions, then we will explain how we extrapolate single-server values to values for sprinting cycles.

The 3-year Total Cost of Ownership (TCO) was calculated by adding datacenter depreciation, datacenter operational expenses, server depreciation, and server operational expenses. Our TCO analysis was modeled on the 2006 analysis in [22], so we assumed a 12% interest rate with capital expenses at \$15 per watt and operating expenses at \$0.04 per kilowatt over a datacenter amortization period of 15 years. We used 2016 figures for average electricity cost, \$0.12 per kilowatt hour. Taken together, we used these values to compute the datacenter depreciation and operation expense. We also assumed a server lifetime of 3 years, a server operational expense of 5% of the capital expenses per year, and server capital expense of \$1000 per server. We selected a conservative 2.0 power usage effectiveness (PUE). Datacenter operational expenses, server operational expenses, server power, and the PUE overhead were all calculated as dollars per Watt per month. Those dollar per Watt per month values were then summed and multiplied by the server lifetime in months and server peak power to get the final 3-year TCO.

Server power was the key input to the TCO analysis that varied depending on the workload and whether the servers were operated as sprinting cycles. The server power input was calculated from the server average power relative to peak power. Our research only analyzed CPU power, so in order to get the server peak and average power necessary for the TCO calculation, we added a constant 200 Watts to each CPU power value. We used 200 Watts because addition of that value to the maximum observed CPU power values brought us close to the 250 Watt nameplate power value for our server. For each workload, at each core and frequency setting, we computed average CPU power and peak CPU power. The peak value for each workload, at each setting, was used as the peak server power input value for the TCO calculation.

For each workload, we computed the TCO of three servers operated nominally and independently and the TCO of a sprinting cycle of size three. ”Nominal” was defined to be 1 core at 1.8 GHz. The definition of ”best sprinting behavior” was experimentally determined for each workload. To calculate the TCO for servers not utilizing a sprinting method we simply used the average and peak CPU power values for a CPU running the given workload with 1 core at a frequency of 1.8 GHz. That TCO served as the baseline for comparison to the sprinting TCO. To model the TCO for a sprinting scheme, we assumed that two servers were operating at the nominal settings and one server was sprinting at the frequency and core activation settings which maximized performance/Watt measurements for the currently-running workload. The input for peak server power was calculated by averaging 2x the nominal peak CPU power plus the sprint peak CPU power and adding 200 Watts. The input for average server power was calculated by averaging 2x the nominal average CPU power plus the sprint average CPU power and adding 200 Watts.

F. *Synthesizing Accomplishment and Cost*

Simply comparing TCO values is not sufficient because such an analysis fails to incorporate the amount of work being done. Instead, we will compare performance per watt per dollar. Even if the TCO is higher, sprinting cycles may be a better value depending on how much additional work it can complete. Using performance per watt per dollar as our final metric allows us to make that determination. For each workload we compute this value by dividing optimal performance per watt by 3-year TCO:

$$\frac{\text{Performance}}{\text{Watt} \cdot \text{Dollar}} = \frac{\text{Performance/Watt}}{\text{TCO}}.$$

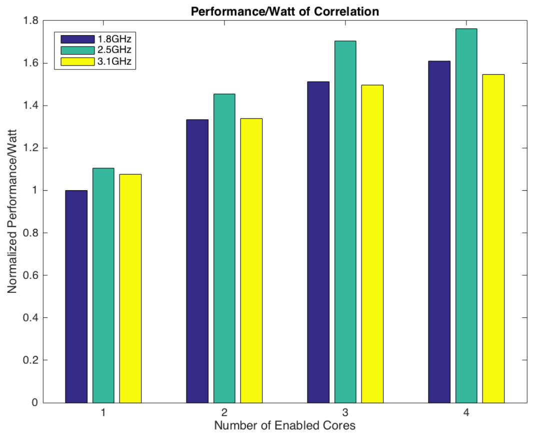

Fig. 2. Normalized performance per watt of `Correlation`.

III. RESULTS

In this section we present our experimental findings. We first determine the optimal sprinting policy for each workload. Then for each workload we compare the TCO for a sprinting cycle (using the corresponding optimal policy) to the TCO for an equal number of machines operated nominally. Finally, we adjust these figures for the amount of work that is accomplished, yielding a metric that we can evaluate to determine the viability of sprinting cycles.

A. *Optimal Sprinting Policy for Each Workload*

We measured the performance per watt of each workload under a variety of computational sprinting configurations to determine the optimal sprinting policy for each workload. In Figure 2, we observe that the best performance per watt for `Correlation` occurs with 4 cores operated at 2.5 GHz. Similarly, Figure 3 shows that

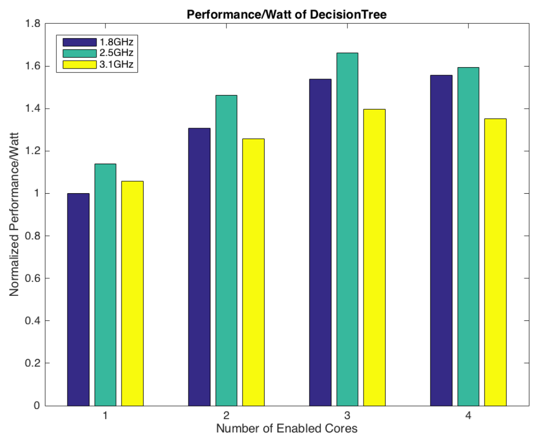

Fig. 3. Normalized performance per watt of `DecisionTree`.

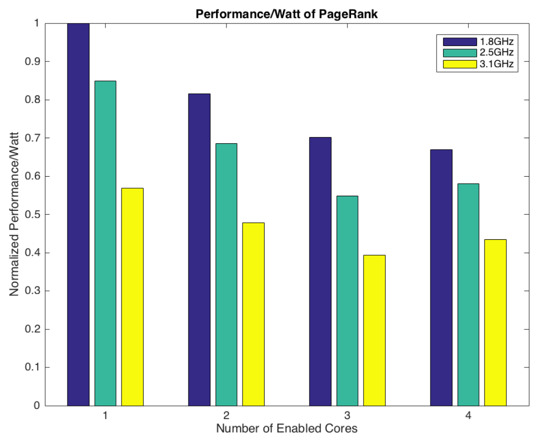

Fig. 4. Normalized performance per watt of `PageRank`.

`DecisionTree` performs best on 3 cores operated at 2.5 GHz. Figure 4 shows that `PageRank` performs best on 1 core at 1.8 GHz, the nominal configuration. This means that `PageRank` does not benefit from any of the sprinting configurations. These results correspond well to what we know about these workloads based on Figure 1. In particular, these results show that the probability of sprinting for a given workload, or the fraction of the time the workload would benefit from sprinting, is roughly proportional to how aggressive that workload's optimal sprinting parameters are.

B. TCO Comparisons

For each workload, we used the power consumed while sprinting optimally (as determined

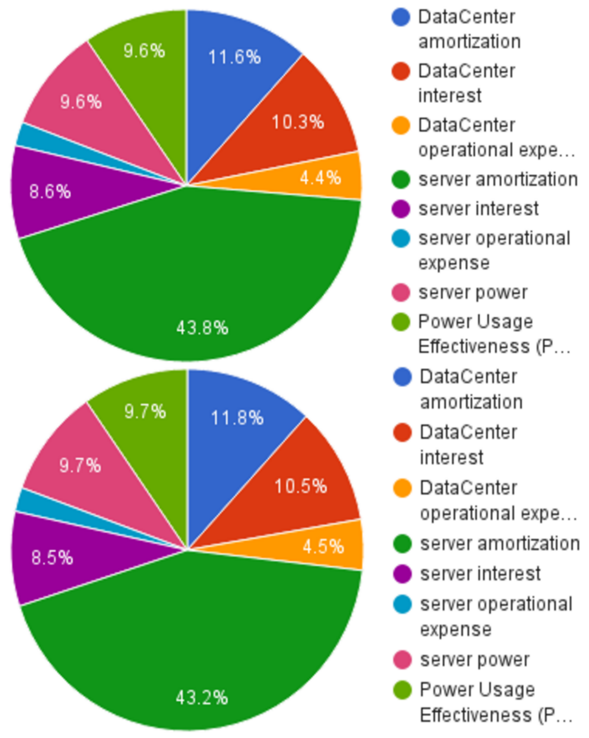

Fig. 5. Example TCO breakdown from our analysis. Top: `Correlation`, nominal. Bottom: `Correlation`, sprinting.

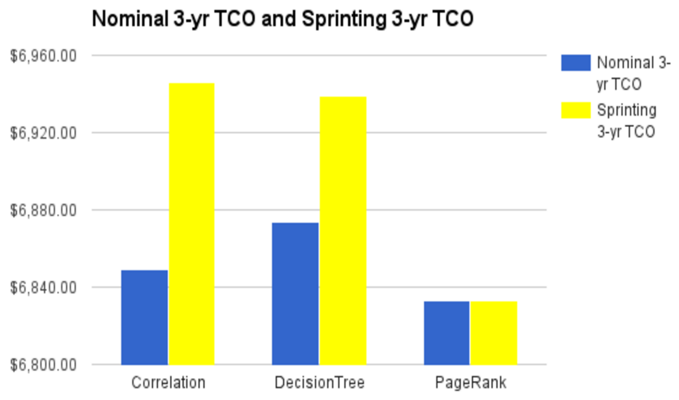

Fig. 6. TCO's of sprinting cycles compared to TCO's of nominally operated machines for each workload.

in the previous section) to compute the TCO of a three-machine sprinting cycle and compared that to the TCO of those three machines operated independently. In Figure 6, we observe that the TCO of three machines operated as a sprinting cycle always exceeds the TCO of those three machines operated independently.

The 3-year TCO for three independently operated machines running `Correlation` and using

1 core at a frequency of 1.8 GHz (the baseline TCO) was $6,849. When `Correlation` was run with one out of the three machines utilizing its optimal sprinting method the 3-year TCO was $6,946. The 3-year baseline TCO for `DecisionTree` was $6,874. The 3-year TCO for `DecisionTree` when utilizing its optimal sprinting method was $6,939. For Pagerank, the baseline 3-year TCO and the optimal sprinting method 3-year TCO were the same because `PageRank` did not benefit from sprinting. That 3-year TCO value was $6,833. A representative example of the TCO breakdown for one of these cases, for both nominal operation and sprinting cycles, is presented in Figure 5.

C. Comparing Economic Merit Metrics

The TCO comparison in the previous section does not account for the actual amount of work being done. For this reason, we determine the viability of sprinting cycles by comparing performance per watt per dollar. In Figure 7, we can see that operating three machines as a sprinting cycle increases the performance per watt per dollar relative to nominal independent operation. This means that although operating machines as a sprinting cycle increases TCO, the performance per watt increases more.

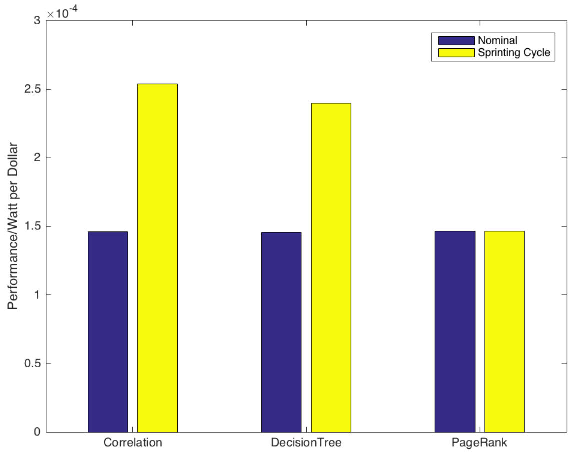

Fig. 7. Performance per watt per dollar (3-year TCO) for three Spark workloads.

IV. RELATED WORK

While there is a substantial body of work on datacenter power management (surveyed in the introduction), there is actually relatively little work on sprinting at the datacenter level. To our knowledge, the only two directly related papers are the previously mentioned works by Zheng et al. [12] and Fan et al. [13] The latter is most closely related. It considered problem of allocating resources for sprinting among servers sharing a common power supply. The basic idea of the sprinting cycle is similar to that paper's greedy model, but instead of analyzing how machines in a rack should share power, each rack in a system designed to support sprinting cycles would be provisioned to permit indefinite, simultaneous sprinting in each sprinting cycle in the rack. Thus, the power emergencies cited as the main flaw with their greedy model are not a concern.

V. CONCLUSION

We found that for workloads that spend much of their time CPU-bound, like `Correlation` and `DecisionTree`, it is more cost-effective in terms of performance per watt per dollar to operate machines in sprinting cycles than to operate them nominally and independently. Workloads that do not spend much time CPU-bound, like `PageRank`, do not benefit from sprinting cycles. The reason for this is simple: since the strategy we've described is essentially round-robin, the probability that a machine running `PageRank` will be selected sprinting at a beneficial time is low, since such beneficial times are relatively uncommon.

Our conclusions do come with some caveats that result from the limitations of our data. For instance, we did not analyze the dynamic behavior that occurs during a transition into or out of a sprinting cycle. There may also be significant practical issues to overcome in order to implement scheduling and task assignment. While we believe that most of our assumptions are reasonable, there are plenty of sources of error in this analysis. That being said, these results can be interpreted as a promising proof of concept that a strategy similar to this one might be of some real use.

FUTURE WORK

There are several potentially interesting applications of policies like the one presented here. Fewer machines could be provisioned for a cluster, with peak load handled by engaging sprinting cycles. If some number of machines failed, a number of sprinting cycles could be activated to temporarily make up some or all of the slack. Perhaps sprinting cycles could be activated in order to help datacenters satisfy latency requirements when servers transition from sleeping state to active state. Whether such policies would be viable would require a detailed study of the particular use case and a more thorough investigation of the assumptions on which we have built our conclusions.

ACKNOWLEDGEMENTS

We're grateful to Dr. Benjamin Lee for providing server hardware and guidance in project planning. We'd also like to thank his students Seyed Zahedi and Songchun Fan for their invaluable advice on Spark workload deployment and system profiling.

REFERENCES

[1] U.S. Environmental Protection Agency. Server and data center energy efficiency, 2007.

[2] Kenneth G. Brill. The invisible crisis in the data center: The economic meltdown of moore's law, 2007.

[3] Asfandyar Qureshi, Rick Weber, Hari Balakrishnan, John Guttag, and Bruce Maggs. Cutting the electric bill for internet-scale systems. *SIGCOMM*, 2009.

[4] David Meisner, Brian T. Gold, and Thomas F. Wenisch. Powernap: Eliminating server idle power. *ASPLOS*, 2009.

[5] David Meisner, Christopher M. Sadler, Luiz Andre Barroso, Wolf-Dietrich Weber, and Thomas F. Wenisch. Power management of online data-intensive services. *ISCA*, 2011.

[6] David Lo, Liqun Cheng, Rama Govindaraju, Luiz Andre Barroso, and Christos Kozyrakis. Towards energy proportionality for large-scale latency-critical workloads. *ISCA*, 2014.

[7] Brandon Heller, Srini Seetharaman, Priya Mahedevan, Yiannis Yiakoumis, Puneet Sharma, Sujata Banerjee, and Nick McKeown. Elastictree: Saving energy in data center networks. *NSDI*, 2010.

[8] Dennis Abts, Michael R. Marty, Phillip M. Wells, Peter Klausler, and Hong Liu. Energy proportional datacenter networks. *ISCA*, 2010.

[9] Arun Raghavan, Yixin Luo, Anuj Chandawalla, Marios Papaefthymiou, Kevin P. Pipe, Thomas F. Wenisch, and Milo M.K. Martin. Computational sprinting. *HPCA*, 2011.

[10] Arun Raghavan, Laurel Emurian, Lei Shao, Marios Papaefthymiou, Kevin P. Pipe, Thomas F. Wenisch, and Milo M.K. Martin. Computational sprinting on a hardware/software testbed. *ASPLOS*, 2013.

[11] Hadi Esmaeilzadeh, Emily Blem, Renee St. Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. *ISCA*, 2011.

[12] Wenli Zheng and Xiaorui Wang. Data center sprinting: Enabling computational sprinting at the data center level. *ICDCS*, 2015.

[13] Songchun Fan, Seyed Majid Zahedi, and Benjamin C. Lee. The computational sprinting game. *ASPLOS*, 2016.

[14] David M. Brooks, Pradip Bose, Stanley E. Schuster, Hans Jacobson, Prabhakar N. Kudva, Alper Buyuktosunoglu, John-David Wellman, Victor Zyuban, Manish Gupta, and Peter W. Cook. Power-aware microarchitecture. *Proceedings of the 27th Annual International Symposium on Microarchitecture*, 2000.

[15] Mark Weiser, Brent Welch, Alan Demers, and Scott Shenker. Scheduling for reduced cpu energy. 1994.

[16] Karel De Vogeleer, Gerard Memmi, Pierre Jouvelot, and Fabien Coelho. The energy/frequency convexity rule: Modeling and experimental validation on mobile devices. 2014.

[17] David Snowdon, Sergio Ruocco, and Gernot Heiser. Power management and dynamic voltage scaling: Myths and facts. 2005.

[18] Poweredge r210 ii technical guide.

[19] J. Stamper. Algebra i 2008-2009 challenge data set from kdd cup 2010 educational data mining challenge.

[20] Lars Backstrom, Dan Huttenlocher, Jon Kleinberg, and Xiangyang Lan. Group formation in large social networks: Membership, growth, and evolution. *KDD*, 2006.

[21] Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 2008.

[22] Luiz Andre Barroso, Jimmy Clidaras, and Urs Holzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Synthesis Lectures on Computer Architecture. Morgan and Claypool, 2 edition, 2013.