Joint Server and Network Energy Saving in Data Centers for Latency-Sensitive Applications

Liang Zhou, Chih-Hsun Chou, Laxmi N. Bhuyan, K. K. Ramakrishnan
Computer Science and Engineering Dept.
University of California Riverside, USA
lzhou008@ucr.edu, {cchou011, bhuyan, kk}@cs.ucr.edu

Daniel Wong
Electrical and Computer Engineering Dept.
University of California Riverside, USA
dwong@ece.ucr.edu

***Abstract*—Achieving energy proportionality in data centers supporting latency-sensitive applications is challenging because of the strict Service Level Agreements. Previous works individually focus on making the server energy proportional or reducing the data center network's power consumption for latency-tolerant applications. In this paper, we propose EPRONS to minimize the overall data center's power consumption with latency-sensitive applications by trading-off network slack in favor of providing additional slack for computations. We utilize the linear programming model to consolidate latency-sensitive search queries and latency-tolerant background flows to a minimal subnet of the topology by turning off unused switches and links without violating the application deadlines. Servers take advantage of the additional 'network-provided' slack to allow slowing down request processing. For servers, we design a novel power saving technique using Dynamic Voltage and Frequency Scaling (DVFS) based on the *average tail latency* of a request. If needed, we turn on a minimal number of additional network links and switches to reduce network latency while still maximizing entire data center's power saving. Experimental results show that our scheme saves up to 31.25% of a data center's total power budget.**

***Keywords*-Data Center; Latency-Sensitive; Energy Proportional; DVFS; Traffic Consolidation**

I. Introduction

Energy consumption and not meeting application latency constraints defined in Service Level Agreements (SLAs) can significantly impact data center revenue. In 2014, data centers in the U.S. consumed 70 billion kWh of energy, representing 1.8% of the U.S. electricity consumption [1]. Previous work has aimed to improve the energy efficiency of either Data Center Networks (DCNs) [2]–[5] or servers in the data center [6]–[10] separately. However, latency-sensitive applications in data centers, usually with strict latency constraints are a significant portion of the data center workload. They are organized in a multi-tier tree structure and executed across thousands of servers, and it is important to meet their SLAs. These previous works do not address the need to jointly optimize power consumption both at DCN and server levels for latency-sensitive applications.

To make the DCN more energy-proportional, traffic consolidation proactively shifts flows to a minimal subnet of the topology and turns off unused network devices. Such approaches promise to yield as much as 62% DCN power saving. Nevertheless, existing traffic consolidation techniques [2]–[5] only consider flow's bandwidth demand and ignore the network latency constraints that are critical for latency-sensitive applications. On servers, two techniques have been proposed to improve the energy efficiency while meeting latency constraints: Sleeping and Performance Scaling. For example, DynSleep [11] and SleepScale [12] postpone the servicing of requests and cause a longer idle period so that servers can enter into their deepest sleep states. Rubik [10] and Pegasus [6] dynamically adjust frequency using a statistical performance model so as to finish the service just-in-time. However, these papers do not address the power savings in the DCN. Several papers [13], [14] consider server and network power savings together. They jointly optimize both network and server power for latency-tolerant applications by combining traffic consolidation and Virtual Machine (VM) migration. However, the VM consolidation techniques in [13] and [14] are not feasible for latency-sensitive applications [6] because the database storing application-desired information is statically distributed, and can impact latency severely with excess server-to-database access delays with such consolidation.

Previous approaches for power management in data centers have viewed the DCN and server distinctly. However, we find that this "intuitive" approach is not optimal for overall power savings especially when supporting latency-sensitive applications. This motivates us to design the framework of EPRONS *(Energy PROportional Network and Server)*. As we demonstrate in our evaluations, the power savings that are achieved by only optimizing the server DVFS technique or DCN traffic consolidation is considerably less than jointly solving both the problems as we propose in this paper. This is particularly true when applications have latency constraints, as we find that by deliberately turning on carefully selected additional paths in the DCN enables the servers to magnify their power savings by taking advantage of the additional slack. This paper shows the value of turning on the additional switches and links at the right time to meet application latency requirements.

Existing server and DCN energy proportional techniques

[2]–[5], [7], [9]–[11] sometimes do not cooperatively reduce power as they lack the entire system view. TimeTrader [7] is one example that does borrow the network slack to provide an additional budget for computation. However, it only monitors the ECN bits or the RTO signal in the TCP protocol, assumes the DCN provides load balancing, and the lack of a queue build-up is treated simplistically by adding the full network latency budget to the compute slack. However, traffic consolidation invalidates this assumption and a subnet of the topology may be congested, resulting in TimeTrader becoming overly conservative in not providing any slack to the servers. Previous DVFS-based techniques for servers have strict latency constraints in which we choose the maximum frequency to guarantee every request's server-side deadline. This constrains saving power. We recognize that application-level SLAs primarily matter to users. The network can compensate for slower server response, while still meeting application-level SLAs, guaranteeing service quality and increasing the amount of power saved. This observation helps in designing our new server-side technique, called EPRONS-Server.

In this paper, we dynamically adjust the amount of network slack that is provided to servers by selecting different consolidation policies based on the current demand. For the servers, we propose EPRONS-Server, a novel dynamic power management scheme that changes CPU frequency to ensure that the *average tail latency* meets the latency constraints. This enables more requests to complete service closer to their deadlines. We later show that EPRONS-Server can guarantee that the *average tail latency* of services meets the latency constraints while saving more power, compared to existing techniques. At the DCNs, we develop a linear programming model to consolidate the traffic with latency constraints. The model chooses the best subset of active network devices and corresponding path for flows to maximize the entire data center's power saving. We also utilize heuristic algorithm to accelerate finding the solution.

To evaluate EPRONS, we implement it on the MiniNet emulator and POX [15] controller. Experimental results show that the server power saving of EPRONS-Server outperforms the state-of-art energy proportional techniques. EPRONS can save as much as 31.25% of a data center's total power budget at low loads.

In summary, the major contributions of our work include:

- We consolidate latency-tolerant background flows and latency-sensitive queries to save DCN power while controlling latency.
- We propose EPRONS-Server, a power management scheme on servers that dynamically adjusts frequency to enable the average tail latency of services to still meet latency constraints.
- We jointly optimize the power consumption of latency-sensitive applications by developing a new linear programming model as well as a heuristic approach.

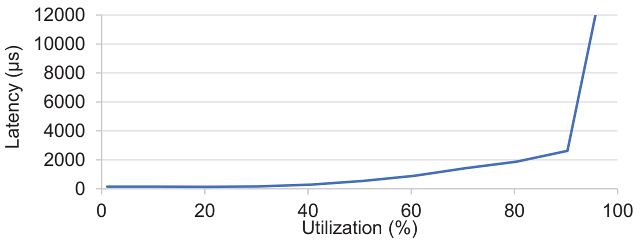


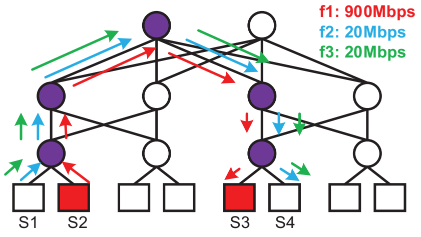
Figure 1. Illustration of latency knee for link utilization vs network latency.

II. Latency-Aware Traffic Consolidation

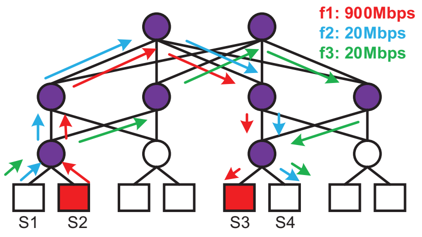
There are three tasks to perform for traffic consolidation: i) measure the traffic statistics and predict future bandwidth demand; ii) optimize the DCN power consumption by shifting flows to the minimal subset of network devices, turning off the other switches and links; iii) reconfigure the flow forwarding rules. This 3-step procedure is performed frequently enough to be responsive to traffic variability in DCNs. The 90th %tile traffic data rate of the last epoch is used to predict the flow's bandwidth demand in the next epoch [3], so as to be able to support the bandwidth demand for all but the outlier cases. To minimize the effect of prediction errors, we incorporate a safety margin for the required link capacity. The primary metric targeted in current traffic consolidation techniques is the bandwidth demand.

In addition to available bandwidth, another important performance metric for latency-sensitive applications in a DCN is the network latency. Previous traffic consolidation techniques [2]–[5] only guarantee the available bandwidth for flows, but may schedule flows on highly utilized links. Thus, latency-sensitive requests or replies may miss their deadlines. On our platform, we measure the average latency of search queries under different link utilizations, and obtain the utilization-latency curve on Figure 1. The results show that the network latency is well behaved at low link utilization (e.g. 20%). Moving a flow from a lightly utilized link to a link with medium utilization does not result in a substantial increase in latency. However, going beyond a utilization threshold the latency increases substantially (the 'knee' of the utilization-latency curve) due to queuing. For example, the latency grows quickly from 139 μs to 11.981 ms beyond this threshold. Thus, we observe that we can continue to consolidate the traffic to a smaller subset of the switches and links in the network until we are close to the knee of the curve, thus maximizing the power savings in the DCN. Beyond this, we run the risk of missing flow deadlines, and at which point we also start taking away the benefit of adding slack to the server 'layer'.

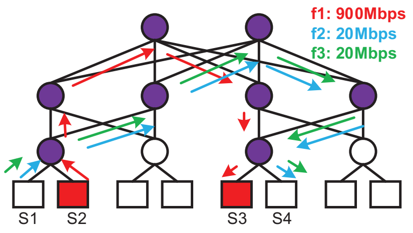
In EPRONS-Network, we leverage an idea similar to FCTcon [16] to control the request and reply latency. We reserve additional bandwidth for requests or replies, compared to their predicted future bandwidth requirement. For example, for a request R with bandwidth requirement of B, the latency-aware traffic consolidation denotes its



(a) K=1



(b) K=2



(c) K=3

Figure 2. Fat-tree topology, 1Gbps link capacity and 50Mbps safety margin. A latency-tolerant flow (red) and 2 latency-sensitive flows (green and blue).

bandwidth requirement as $K * B$. By changing the scale factor K, we can adjust the available bandwidth on the path assigned to request R. A larger available bandwidth on the path reduces the network latency. Latency-aware traffic consolidation dynamically adjusts the scale factor K to control the network latency.

Figure 2 gives an example to show how the scale factor K can affect network latency and the number of active switches. In the Fat-tree topology, all the links have a capacity of 1Gbps. The algorithm provides a 50Mbps safety margin, limiting the maximum available link bandwidth to 950Mbps. There are 3 flows in the DCN. The red flow with 900Mbps data rate is a latency-tolerant 'elephant' flow, while the two latency-sensitive flows (blue and green) have a requirement of 20 Mbps each. In Figure 2(a), the 2 latency-sensitive flows share the same path with the elephant flow when we set the scale factor K to 1. With $K = 1$, the number of active switches is at a minimum and the DCN consumes the least amount of power. But the 2 latency-sensitive flows are likely to suffer a higher latency because many of the inter-switch links are almost fully utilized. In Figure 2(b), we change the scale factor K to 2. This results in moving either the green or blue flow to a new path because of bandwidth constraints. More switches have to be turned on, but we decrease the latency for flows on those links. When we further increase the scale factor K to 3 in Figure 2(c), both the blue flow and the green flow are scheduled on different paths, thus further reducing the network latency.

III. Dynamic Power Management on Servers

A common technique to save energy in latency-sensitive applications is to exploit latency slack by slowing down request processing so that the target tail latency is just satisfied [6], [10]–[12]. In this paper, SLA is defined as 95th %tile tail latency of the service to meet the latency constraints, which is widely used in prior work [6], [7], [10]–[12]. Intuitively, the processor should run at the minimum frequency that satisfies the request's deadline. However, this is difficult as frequency selection also affects the latency of subsequent requests due to request queuing. In TimeTrader [7], frequency settings are periodically updated based on the monitored tail latency. While effective with smooth diurnal traffic variation, these prior techniques result in unacceptable latency constraint violations as they lack responsiveness, which is crucial with fast-varying bursty traffic patterns that is common in latency-sensitive applications [10], [11], [17].

To address this problem, [10], [11] use per-request latency distribution to compute the slowdown/delay for each request, and find the configuration that satisfies *all* queued requests. The frequency setting is then determined by the request with the least latency slack. While satisfying latency constraint, this conservative frequency selection does not fully exploit the energy saving opportunities of latency slack, since the requests other than the limiting request will have their tail latency shorter than the constraint.

These prior techniques all assume a fixed request deadline [6], [10]–[12]. However, when coordinating server power management with network power management, the request deadline is extended with a random value taking advantage of additional network slack, which varies per request. Therefore, the latency variation behavior of the DCN should be considered. In this section, we design the EPRONS-Server under a more generalized assumption, with variable request deadlines. EPRON-Server is a slack variation-aware power management scheme that determines the request processing speed based on the per-request server and network slack.

A. Details on EPRONS-Server

In prior work, the frequency selected is based on the highest frequency required to process a queued request just-in-time to meet its deadline. This results in a single request meeting tail latency constraints just-in-time, but with all other requests finishing before the deadline. This is illustrated in Figure 3. There are 4 requests in the queue ($R1$, $R2$, $R3$ and $R4$). The latency constraint is given as L.

L_i is the tail latency of request R_i under the frequency determined by the scheduler. In the previous work, the scheduler determines the frequency for each request so that L_i is smaller than L, as a result, all the queued requests

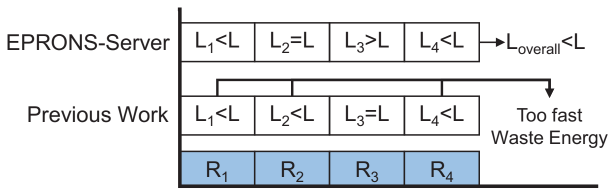


Figure 3. High level idea of EPRONS-Server.

can meet their deadlines. Due to this policy, power savings become worse with the latency slack variation introduced by network power management. To account for latency slack variation, the main insight behind EPRONS-Server is that we can *slow down request processing so that* ***some queued requests*** *will exceed the tail latency constraint*. We only need the *overall tail latency of the queued requests* to satisfy the constraint. One motivation behind sacrificing some queued requests' miss rate for better energy saving is that the 'additional delay' due to lower selected frequency can be compensated by the slack for the reply. However, it is not factored in prior work [6], [7], [10]. Thus, the 'additional delay' for some queued requests on servers has limited impact on the performance and brings better energy saving as demonstrated on Figure 12.

The challenge in designing EPRONS-Server lies in determining the tail latency of each request under different frequencies. To account for the queuing effect and short-term variation, EPRONS-Server uses a performance model based on the request's probability density function (PDF), similar to [10], [11]. In our experiments (section V-A), the PDF is obtained through measuring the service time distribution of Apache Solr search engine [18]. When we dynamically change the frequency setting, the request service time can be calculated accordingly. Based on the model, we derive the *violation possibility* (VP) for each frequency setting, and the new frequency is then determined based on the average VP for all the queued requests at every request arrival and departure instance. The goal is to select an average VP that will not violate the tail latency constraint.

To illustrate this, take the following example. Assume we have three requests ($R1$, $R2$, $R3$) in the queue. In order for the requests to finish just-in-time, they would require to run at 1GHz, 1.2GHz, and 1.1GHz. Prior works would set the processor at 1.2GHz for $R2$, ensuring that the latency constraint would be met. If the SLA is set at the 95th %tile latency, it allows for a violation probability of 5%. Even if we set the processor at 1.2GHz, $R2$ has a 5% chance of exceeding the deadline, while $R1$ and $R3$ would have less due to running at a higher frequency. This results in some lost energy savings potential for $R1$ and $R3$. The goal of EPRONS-Server is to select a frequency where the violation probability of all three requests combined is 5%. In order to achieve this, we simply need the *average VP* of all three queued requests to be 5%.

Figure 4 gives an example of how EPRONS-Server chooses the operating frequency. Assume, at time $t = 0$, the core finishes a request and is ready for the next. Requests $R1$ and $R2$ arrived previously and are queued in the buffer, with deadline $D1$ and $D2$, respectively. In previous work, finding the processing frequency for $R1$ is equivalent to finding the *maximum frequency* that satisfies both request $R1$ with start time $t = 0$ and deadline $D1$, and a request $R2e = R1 + R2$ with start time $t = 0$ and deadline $D2$. We call $R2e$ the *equivalent request* of $R2$, which is defined as the convolution of a given request, and all requests ahead of it in the queue. $R2e$ is a convolution of the amount of work required for both $R1$ and $R2$, as $R2$ can only complete after $R1$ completes.

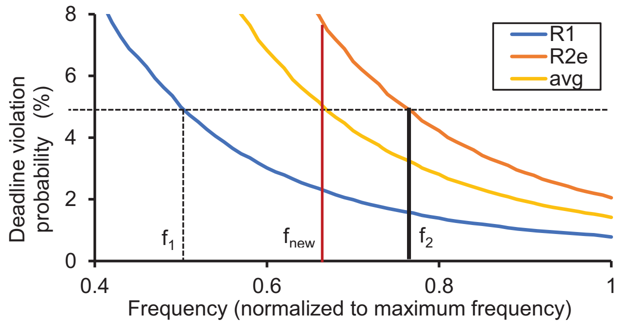


Figure 4. Energy saving opportunities with *average* tail latency.

Figure 4 plots the deadline violation probability for request $R1$ and $R2e$ under different operating frequency (X-axis). The horizontal dotted line represents deadline miss rate constraint of 5%. The frequency f_1 and f_2 will satisfy the deadlines for request $R1$ and $R2e$, respectively. Since $f2$ is the minimum frequency that satisfies the miss rate constraint for both $R1$ and $R2e$, the core will process $R1$ at frequency $f2$. While guaranteeing meeting the deadline miss rate constraint, this conservative frequency selection leads to unnecessarily high request processing speed for $R1$. As we can see, under frequency $f2$, $R1$ will have it's VP (1.8%) far lower than the required miss rate (5%), which results in inefficient energy consumption.

In EPRONS-Server, instead of choosing the maximum frequency required for all requests, we select the frequency based on the *average VP*. As plotted in the Figure 4, the average VP falls between the VP of $R1$ and $R2e$. Thus, we can run at frequency f_{new}, which significantly reduces the frequency and saves more energy. Although it leads to the miss rate constraint violation of $R2e$, the high miss rate of $R2e$ is compensated by the low miss rate for $R1$, hence the overall miss rate constraint is still satisfied.

B. Violation probability

The following model is used for determining the processing frequency at the request departure instance. For a request in the queue, the amount of work that can be done under frequency f until the deadline D is[1]

$$\omega(D) = f \times (D - T_{start}) \tag{1}$$

where T_{start} is the time the request starts to be processed. The violation probability distribution for $\omega(D)$ can be found by computing the complementary cumulative distribution

[1]We assume that the work can be done in time D is proportional to the frequency to simplify the analysis. In our implementation, we use the model taking into account the frequency independent part of the execution [10].

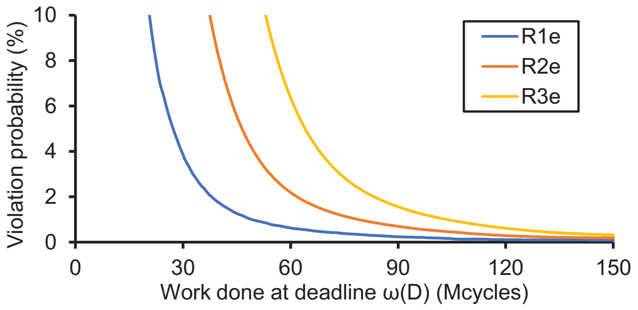


Figure 5. Violation probability of three requests under different $\omega(D)$.

function (CCDF) from the *request's equivalent distribution*. Figure 5 shows the violation probability distribution for three equivalent requests: $R1e$, $R2e$, and $R3e$. Finding the VP is simply finding the corresponding "y" on a line given the "x". By combining the request's equivalent distribution with equation (1), we can get the VP for different frequencies and deadlines.

The above model is used for determining the processing frequency at the request departure instance. It can also be applied at the request arrival instance with a minor modification. Consider a situation when a request arrives at time $t = A$ and finds that the core is processing a request, $R0$. It is equivalent to the scheme that the core just starts processing a request $R0e$ with the distribution equal to the distribution of the *work left* of request $R0$, during time $t = 0 \sim A$. The equivalent distribution of the queued requests will then be $R1e = R0e + R1$, $R2e = R0e + R1 + R2$ and so on.

C. Reducing computation overhead

The most computational intensive portion of our proposed scheme lies in finding the *equivalent distribution* for the queued requests. For the nth queued request, it needs $n - 1$ convolution operations. With the use of Fast Fourier Transform (FFT), computing one convolution requires 20μs in our machine. To further reduce the computation overhead, we take the assumption that the request distribution will be similar within a period of time. Thus, the equivalent distributions can be reused once it is computed. However, at the request arrival instance, since the conditional probability is unknown beforehand, we have to perform n convolutions. This overhead is considered while determining the frequency by replacing D in equation (1) with $D' = D - overhead$.

Once we have the equivalent distributions, the time it takes to determine the operating frequency is shortened by applying binary search on the average VP. In our experiments, it takes less than 30μs and can be safely ignored, since the request processing time usually falls in the millisecond range.

IV. Joint Power Management

A. Latency and Power Analysis

In joint server-network power management, the scale factor K is the key to controlling the entire data center's power consumption while satisfying all the latency constraints. In Figure 6, we show that the scale factor K can be utilized to trade-off the DCN power and DCN latency. As mentioned in section II, the network latency will decrease if the flow is routed along less utilized links. This can be achieved by enlarging the scale factor K and thus allocating more free bandwidth on the assigned path. Accordingly, the larger K will result in higher DCN power consumption because additional links and switches are activated. In real deployments, it would be hard to predict network latency based on current network conditions, as it is highly workload dependent. In EPRONS, we use a portion of the application queries to train our model and then use this model, assuming that the workload features don't change significantly for the same application. Then, we measure the average tail latency of search queries for different scale factors K and use this information for determining network resource allocation.

On servers, power consumption is affected by server utilization and tail latency constraints (i.e., server time budget). Prior works [9], [10], [19] assume that every request of latency-sensitive applications has the same fixed tail latency constraint. If the server layer borrows some slack from the network layer and use it to increase the server time budget, we can make the processing on servers slower and thus save more power, assuming that the server utilization doesn't change. Because the scale factor K can control network latency/slack, it will indirectly affect the server processing time budget and thus the server power consumption. Similarly, we measure the server power consumption for different utilizations and tail latency constraints that may then be used to parameterize our model.

B. Optimization Model

The design motivation behind EPRONS is to jointly optimize the entire data center's power consumption without violating both network and server tail latency constraints. We formalize this optimization problem as a linear programming model, in which the objective function minimizes the total system power by searching for the optimal scale factor K. Our model is as follows:

Objective function:

$$\text{minimize} \quad \sum_{(u,v)\in E} X_{u,v} * l(u,v) + \sum_{u\in V} Y_u * s(u) + N * P_{server} \tag{2}$$

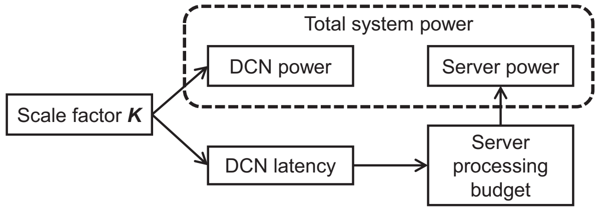


Figure 6. Both the DCN power and Server power can be controlled through adjusting the scale factor K in latency-aware traffic consolidation.

Constraints:

$$1 \leq K \leq K^{max} \tag{3}$$

$$\forall (u,v) \in E, \sum_{i=1}^{j} f_i(u,v) \leq X_{u,v} * c(u,v) \tag{4}$$

$$\forall (u,v) \in E, f_i(u,v) = -f_i(v,u) \tag{5}$$

$$\forall i, \forall u \in V, \sum_{h \in H_u} f_i(u,h) = \begin{cases} K * d_i & u = s_i \\ -K * d_i & u = t_i \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

$$\forall u \in V, \forall h \in H_u, X_{u,h} = X_{h,u} \leq Y_u \tag{7}$$

$$\forall u \in V, Y_u \leq \sum_{h \in H_u} X_{u,h} \tag{8}$$

$$\forall i, \forall (u,v) \in E, f_i(u,v) = K * d_i * Z_i(u,v) \tag{9}$$

Given the network topology $G =< V, E >$, we use X and Y to indicate the ON or OFF states for the links and switches respectively. $l(u,v)$ is the power consumption of link (u,v) and $s(u)$ is the power consumption of switch u. N is the total number of servers and P_{server} is the power consumption of each individual server. The objective function minimizes aggregate power consumption of all the switches and servers. Equations (4), (5), (6) are the well-known constraints in network flow problems. The only difference is that we use a scale factor K to multiply the original flow bandwidth demand. $c(u,v)$ is the capacity of link (u,v) and $f_i(u,v)$ is the size of flow on link (u,v) for flow i. Every flow has source s_i, destination t_i and bandwidth demand d_i. Equations (7) and (8) guarantee that the switch will go to sleep state if all the directly connected links are off. $Z_i(u,v)$ means that if flow i will use link (u,v). With equation (9), we avoid splitting flows to avoid packet reordering. Our optimization model is independent of the network topology. Similar to [2], [3], [5], [13], we run the optimization every 10 mins. This is a reasonable time in practice, but can be changed depending on the network's characteristics. By solving this linear programming model with CPLEX, we can find the best flow assignments and active switches. This model is only used to find the best flow assignments. EPRONS-Server will utilize the actual monitored network slack to extend the server time budget.

Although the linear programming model can be solved in polynomial time [2], it takes a long time to find the optimal solution for a large DCN topology, unacceptable with bursty data center traffic. For example, the computation time of the linear programming model can be more than 42 min. on our platform, with 3000 flows in a 4-ary Fat-tree topology. In real deployment, we design the heuristic algorithm (similar to the greedy bin-packing algorithm in [2]) to accelerate the latency-aware traffic consolidation. In the current design, we ignore the switch ON/OFF transition overheads because we use a software switch. However, our measurement on a HPE switch show that the power-on time is about 72.52 sec. We can avoid the transition overheads by having 'backup' paths, as described in [5] or a novel hardware design with sleep states [2].

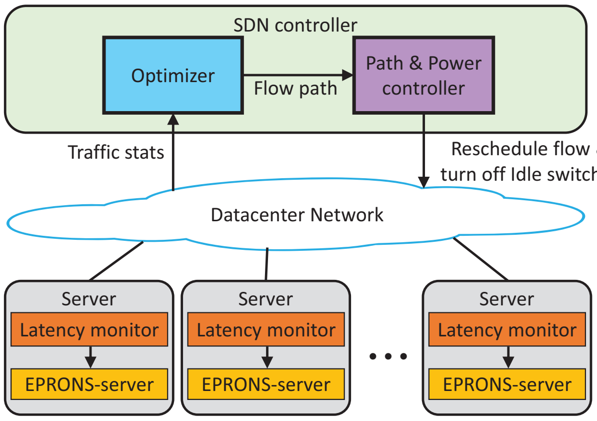


Figure 7. Framework overview of EPRONS.

C. *Framework of EPRONS*

Figure 7 gives an overview of our framework. The system includes two distinct parts: the EPRONS-Server module on every server and the Optimizer (EPRONS-Network) in the centralized Software Defined Networking (SDN) controller. On servers, we assume that each server has the same service time distribution. The latency monitor module measures each request's network latency. To be more conservative, we only use the request slack. The EPRONS-Server module adds the different network slack of each search request to its compute budget. Then we dynamically decide the CPU core frequency to guarantee that the *average tail latency* of requests to meet the tail latency constraints. In the centralized SDN controller, we gather both the utilization at servers and the traffic matrix inside the DCNs. This information is periodically fed to the Optimizer to find the best active subgraph of the topology and corresponding flow assignments. The Path & Power controller is responsible for inserting new forwarding rules and turning off idle switches.

V. Evaluation

A. *Implementation*

MiniNet. We use a 64-core machine with 2GHz CPUs and 64G memory to build a virtual network using the MiniNet emulator that includes virtual hosts, virtual network interfaces, software switches (Open vSwitch) and software SDN controller. Each host in MiniNet is a shell process that runs the actual program. In our implementation, we create a 4-ary Fat-tree topology with 16 servers.

POX. We use POX [15] as our SDN controller. The POX controller fetches flow statistics and link utilization every 2s with an openflow message. Then, the optimizer in POX runs periodically (every 10 mins.) to find flow paths which minimizes the entire data center's power consumption with latency constraints.

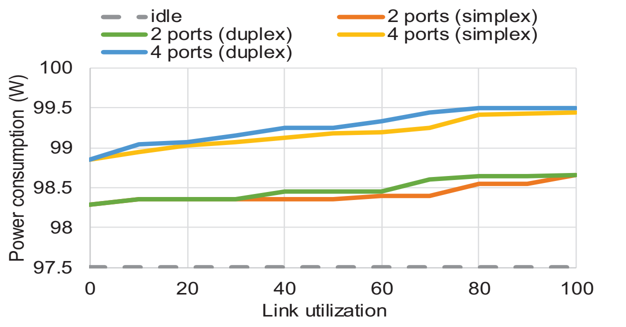


Figure 8. Switch (HPE E3800 J9574A) power for varying link utilization.

Server. The performance of virtual host is limited in single machine MiniNet setup. We attempted to run the Apache Solr search engine inside Containers [18]. But it was difficult to launch 16 Docker Containers and 20 Open vSwitch instances on a single machine because of the limited number of cores and memory. In the future work, the MiniNet network could be extended to a cluster of servers.

To mimic the partition-aggregation behavior, we built a search engine simulator in the MiniNet. Instead of running real query processing, our simulator generates the service time of a query based on the service time distribution, and uses an empirical model to scale the service time for different operating frequencies. We acquire the servers' service time distribution through real machine experiments with the open-source Xapian distributed search engine [20]. We built indexes of the Wikipedia's English XML export [21], which contains all the Wikipedia articles/pages. These document indexes are then partitioned and distributed across 16 Xapian index nodes. We randomly generate 100K search queries, run and log their processing time on the Index Serving Nodes (ISNs). In our simulator, we randomly choose a server to be the aggregator, while the other 15 servers will then be the ISNs for each user query. The aggregator will broadcast sub-queries to all ISNs. Each ISN serves the sub-queries and executes EPRONS-Server to save server power.

Power Evaluation. To acquire the power consumption, our server simulator records the processing time for each frequency setting. The average power consumption is calculated based on the time and power consumption under each frequency setting. We set the frequency range between 1.2-2.7GHz in 100 MHz steps. We measure the power consumption under different frequency settings on a 12-core Xeon E5-2997 v2 CPU. The measured power consumption under maximum and minimum frequency setting is 4.4W and 1.4W for a CPU core, respectively. In our simulator, each server has a CPU with 12 cores. Moreover, we also consider the power consumption of other components (motherboard, main memory, etc.) of the system. Based on the ratio between dynamic power and static power for a Huawei XH320 V2 server [22], the static power is set to be 20W.

To investigate how link utilization affects switch power, we measure the power consumption of a HPE switch (E3800

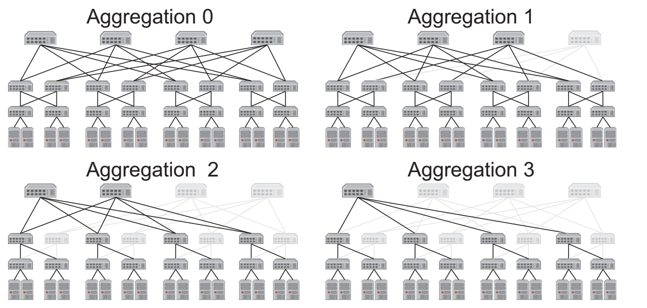
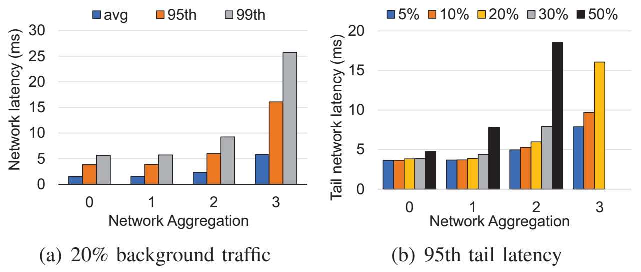


Figure 9. 4 different network topologies after consolidation. The greyed out switches and lines represent the deactivated switches and links, respectively.



(a) 20% background traffic (b) 95th tail latency

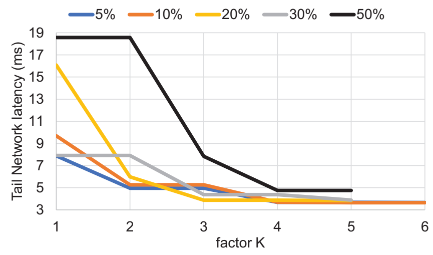
Figure 10. Network latency under different degrees of aggregation.

J9574A) as we gradually increase the link utilization. The switch idle power is 97.5W. Figure 8 shows that the increased power consumption (0.59W) is negligible when we change the link utilization from 0 to 100%, independent of whether we activate 2 or 4 ports. That is only 0.6% of the idle power. So, we assume the switch power remains the same when the utilized network bandwidth changes. Since the power consumption doesn't increase, there is no increase in the temperature of the switch. Both of the above observations are in line with prior work [2], [3], [5]. For network power consumption, we use the measurement result of a 4-port switch in [23]. The power consumption of the active switch is 36W. These power models are selected to be a closer match to our implementation platform.

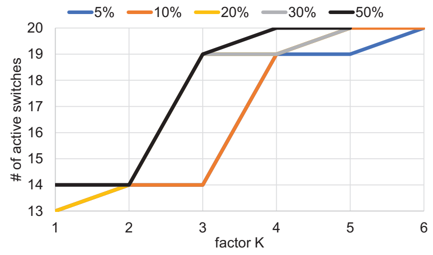
B. Results

In EPRONS, we aim to optimize the entire data center's power consumption while satisfying all the latency constraints. Although there are many parameters in our model, the scale factor K is the only parameter to be tuned for optimization. Network budget and server budget are defined by SLAs. The bandwidth requirement of background traffic and server utilization are determined by instantaneous application features. In this section, we investigate the interplay of all these parameters and evaluate the power saving of EPRONS under different configurations. Then, we compare EPRONS with other energy-proportional techniques.

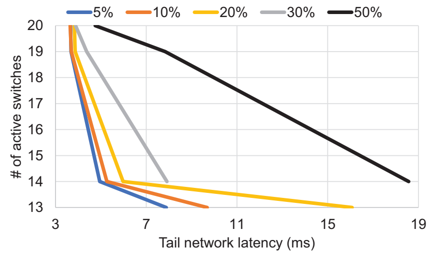
1) Network Power Management: In DCNs, the traffic aggregation policy has a significant impact on the network latency. Figure 9 shows all the possible aggregation policies in the 4-ary Fat-tree topology. From Aggregation 0 to Aggregation 3, we gradually turn off the core-level switches



(a) factor K vs network tail latency

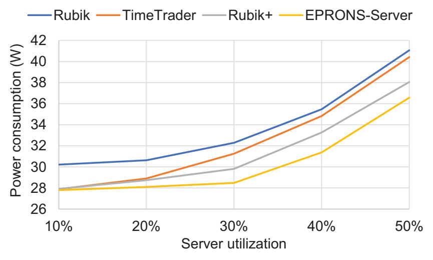


(b) factor K vs # of active switches

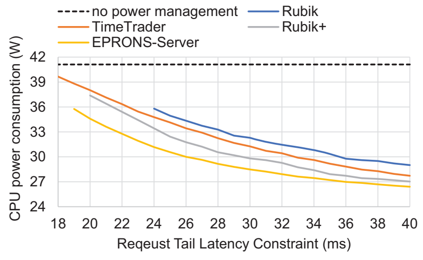


(c) network tail latency vs # of active switches

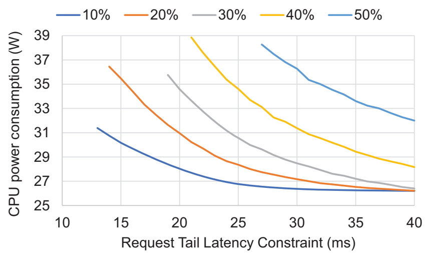
Figure 11. Network sensitivity results.



(a) Server utilization vs Power (latency constraint 30ms)



(b) Request tail latency constraint vs Power (server utilization 30% [9])



(c) (utilization, latency constraint) vs Power for EPRONS-Server

Figure 12. Server sensitivity results.

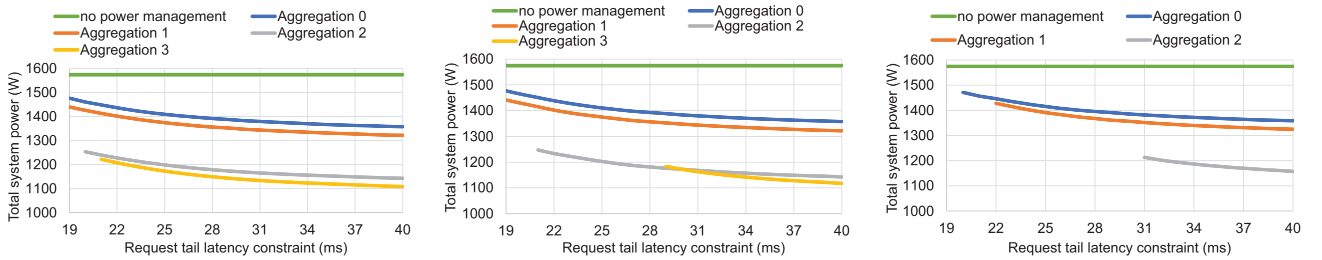
and the corresponding aggregation-level switches.

The network latencies of search queries are shown in Figure 10. Apart from the aggregation policy, the background traffic has an impact on the network latency as well. So, we first fix the network background traffic at 20% of link capacity in Figure 10(a). The network latency (especially 99th %tile) increases considerably when we consolidate traffic to a smaller portion of the topology. From aggregation 0 to aggregation 3, the 99th %tile latency increases from 5.64ms. to 25.74ms. Figure 10(b) plots the impact of the aggregation policy on the 95th %tile tail latency for various background traffic. The 95th %tile tail latency follows a similar increasing trend under different network utilizations.

In Figure 11, we show that the scale factor K is a good parameter to trade-off between network tail latency and network power consumption. Figure 11(a) and Figure 11(b) compare the variance of tail network latency and # of active switches when we increase the scale factor K. Each line represents one possible background traffic, in terms of link utilization. Larger K results in smaller network tail latency. Accordingly, more switches are activated. For example, the network tail latency with 50% background traffic (black line) decreases to 4.75ms if the K is set to 4. The reason is shown on Figure 11(b). We observe that 6 more switches are turned on. Figure 11(c) plots the # of active switches vs. the tail network latency. Each point on the line represents one value of K. The optimal scale factor K is the point that is closest to origin (0, 0). We observe that K can effectively trade-off between # of active switches and tail network latency.

2) Server Power Management: Figure 12 gives the power consumption with only the EPRONS-Server. In these experiments, we do not apply any network power management, and the background traffic is set to achieve 20% network utilization [2], [7]. To show the effectiveness of EPRONS-Server, we also implement and compare our results with two state-of-the-art server power management techniques: Rubik [10] and TimeTrader [7]. Since Rubik does not consider network latency/slack, for a fair comparison, we also implement a network aware version of Rubik (Rubik+), which monitors and uses the network slack.

Figure 12(a) shows the CPU power consumption for different server utilizations. In these experiments, we set the request's tail latency constraint to be 30ms (25ms server budget and 5ms network budget). We can see that Rubik consumes the most power across all load levels. Since Rubik does not consider network latency, it does not fully exploit the slack in the request's latency. Moreover, except at very low loads (10%), Rubik+ and EPRONS-Server consistently outperform TimeTrader. TimeTrader uses feedback to select the CPU frequency to meet the request tail latency constraint. The simple control algorithm in TimeTrader changes the CPU frequency every 5 seconds [7], and is unable to fully exploit power saving opportunities. But Rubik+ and EPRONS-Server adjust the CPU frequency based on a statistical performance model at a per-request granularity. Finally, EPRONS-Server outperforms all other schemes across the entire utilization range. Compared with Rubik+, EPRONS-Server reorders requests based on their deadlines and uses the average VP, instead of the maximum, to determine the request processing frequency. By doing so, EPRONS-Server can better utilize the network aware slack in compute budget and achieve lower power consumption.



(a) low background traffic (1%) (b) medium background traffic (20%) (c) high background traffic (50%)

Figure 13. Total system power under different degree of network consolidations. This result is scaled based on the result of our MiniNet experiments with 30% server utilization. Each switch consumes 36W [23], and each server has a 12-cores CPU, and static server power consumption is 20W [22].

In Figure 12(b), we vary the server budget from 5 to 35ms while fixing the network latency budget at 5ms. The search load is set so as to have the server utilization at 30% [9]. First, no scheme is able to meet a request's tail latency constraint smaller than 18ms. Second, in the range of 18-19 ms, TimeTrader is able to meet the request tail latency constraint and save power. But, for a request tail latency constraint at 19 ms or greater than 18ms, EPRONS-Server consistently outperforms all other schemes because of its efficient utilization of the request slack.

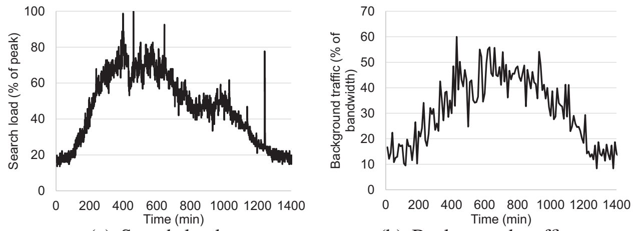
Figure 12(c) shows the power consumption of EPRONS-Server with different latency constraints and search loads. Each line in the figure represents a server utilization ranging from 10% to 50%. As we see in the figure, the CPU power consumption decreases significantly for all server utilization levels as the request latency constraint increases at the smaller values. This justifies the approach of EPRONS to give the additional slack from network budget to the server to achieve a significant reduction of CPU power consumption.

3) Joint Power Management: We now show the compelling benefit of combining appropriately the optimization of server and network power while meeting application latency constraints. In one experiment (Figure 13), the server utilization is set at 30% [9]. We then change parameters such as request tail latency constraint, network aggregation policy and background traffic to see their impact on total system power consumption. When the background traffic is negligible (1%, Figure 13(a)), the 4 different aggregation policies can nearly meet all the tail latency constraints. Aggregation 3 consumes the least total system power. As we increase the tail latency constraint, the data center consumes less power, particularly because we can have longer idle times at servers. We then add more background traffic (20%), as seen in Figure 13(b). The total system power has a similar trend, except that aggregation 3 cannot support a tail latency constraint less than 29ms.

Between a tail latency constraint of 29ms and 31ms, one interesting result is that EPRONS can save more power if we deliberately turn on a switch. From aggregation 3 to aggregation 2, more switches are activated and thus we have larger network slack. The benefits of this network slack is amplified in our EPRONS-Server technique which can more than compensate for the increased power consumption in the network. The goal of EPRONS is to properly trade-off network slack and server slack to minimize total system power, not just simply combining them. Finally, as we increase the background traffic to 50%, Figure 13(c) shows that aggregation 3 is not feasible, and even aggregation 2 requires a latency constraint greater than 31 ms.

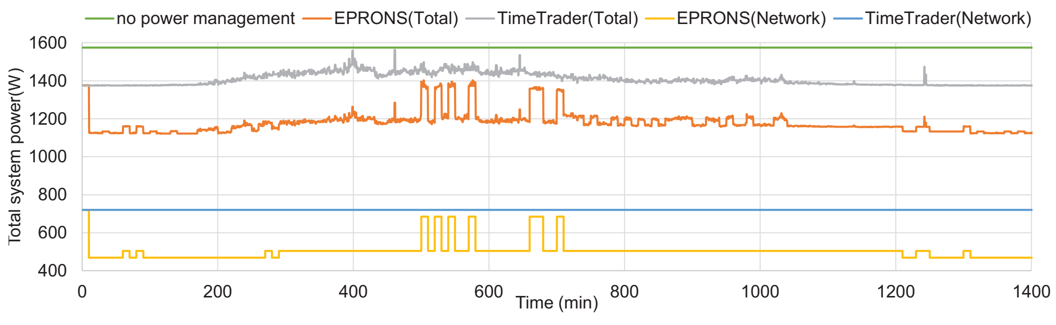
In the following experiments, we use the Wikipedia trace [21] to evaluate EPRONS. Both the search load and background traffic (Figure 14) spans one 24 hour period, indicating that it follows a diurnal pattern. As we noted, prior traffic consolidation techniques [2], [3], [5] don't guarantee network latency constraints, and energy-proportional server techniques [6], [9]–[11] don't save DCN power. We compare EPRONS with TimeTrader [7], which is a cross-layer energy-proportional technique and meets both network and server latency constraints. Although previous work, such as [13], [14] jointly optimize server-network power, they ignore latency constraints.

Figure 15 gives the power saving results. On average (Figure 15(b)), the total system power saving of EPRONS is a factor of *more than 2* better than TimeTrader. EPRONS saves as much as 25% of the total system power, compared with only 8% in TimeTrader. Since EPRONS-Server is a more responsive per-request technique compared with the feedback-based TimeTrader, even our server-side power saving outperforms TimeTrader by 2%. The detailed total system power over the 24 hour interval is shown in Figure 15(a), with a 1 min. granularity. We observe that the DCN power consumption of EPRONS follows the diurnal pattern. On the other hand, TimeTrader doesn't save any DCN power. For the total data center power consumption, EPRONS can



(a) Search load (b) Background traffic

Figure 14. Traffic variation in one day.



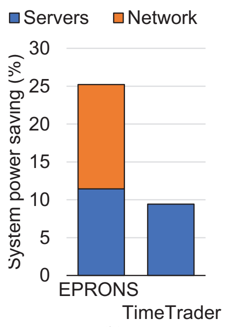


Figure 15. (a) Total system power consumption and (b) average power saving with the varying search load and background traffic.

save up to a maximum of 31.25%, measured over one minute intervals, of total system power. This occurs during the night, because of the lower workload intensity. The maximum power saving in TimeTrader is only 12.5%. We conclude that jointly optimizing power consumption on both DCN and servers (EPRONS) can make the entire data center more energy efficient.

VI. Conclusion

We observe that solving for the power saving of servers and the DCN independently so that they are energy proportional results in their working at cross-purposes. Turning off network switches and links may save network power but will cause servers to work harder to meet the resulting tighter latency constraints. What is more, our results prove that the entire data center could be more power efficient if we, at times, deliberately turn on more switches and links to enable servers to have more slack in their latency targets.

These findings motivate us to design EPRONS, a novel joint energy proportional technique for the entire data center. EPRONS minimizes the entire data center's power consumption through dynamically searching the optimal parameter K in our linear programming model while guaranteeing the latency constraints at both DCNs and servers. Furthermore, the server part of EPRONS outperforms the existing state-of-the-art energy proportional techniques. Combining both the server and DCN energy management, EPRONS can save as much as 31.25% at the peak, and 25% on average, for the data center's total power budget for search workloads.

Acknowledgment

The research was partly supported by NSF grant CNS 1216014.

References

[1] "United states data center energy usage report 2016," https://eta.lbl.gov/publications/united-states-data-center-energy.
[2] B. Heller, S. Seetharaman, P. Mahadevan *et al.*, "Elastictree: Saving energy in data center networks," in *NSDI*, 2010.
[3] X. Wang, Y. Yao, X. Wang, K. Lu, and Q. Cao, "Carpo: Correlation-aware power optimization in data center networks," in *INFOCOM*, 2012.
[4] M. Zhang, C. Yi, B. Liu, and B. Zhang, "Greente: Power-aware traffic engineering," in *ICNP*, 2010.
[5] N. Vasić, P. Bhurat, D. Novaković, M. Canini *et al.*, "Identifying and using energy-critical paths," in *CoNEXT*, 2011.
[6] D. Lo, L. Cheng, R. Govindaraju, L. A. Barroso, and C. Kozyrakis, "Towards energy proportionality for large-scale latency-critical workloads," in *ISCA*, 2014.
[7] B. Vamanan, H. B. Sohail, J. Hasan, and T. N. Vijaykumar, "Timetrader: Exploiting latency tail to save datacenter energy for online search," in *MICRO*, 2015.
[8] D. Wong and M. Annavaram, "Knightshift: Scaling the energy proportionality wall through server-level heterogeneity," in *MICRO*, 2012.
[9] D. Meisner, B. T. Gold, and T. F. Wenisch, "Powernap: Eliminating server idle power," in *ASPLOS*, 2009.
[10] H. Kasture, D. B. Bartolini, N. Beckmann, and D. Sanchez, "Rubik: Fast analytical power management for latency-critical systems," in *MICRO*, 2015.
[11] C.-H. Chou, D. Wong, and L. N. Bhuyan, "Dynsleep: Fine-grained power management for a latency-critical data center application," in *ISLPED*, 2016.
[12] Y. Liu, S. C. Draper, and N. S. Kim, "Sleepscale: Runtime joint speed scaling and sleep states management for power efficient data centers," in *ISCA*, 2014.
[13] K. Zheng, X. Wang, L. Li, and X. Wang, "Joint power optimization of data center network and servers with correlation analysis," in *INFOCOM*, 2014.
[14] H. Jin, T. Cheocherngngarn, D. Levy *et al.*, "Joint host-network optimization for energy-efficient data center networking," in *IPDPS*, 2013.
[15] "Pox controller," https://github.com/noxrepo/pox.
[16] K. Zheng and X. Wang, "Dynamic control of flow completion time for power efficient data center networks," in *ICDCS*, 2017.
[17] M. Jeon, Y. He, S. Elnikety, A. L. Cox, and S. Rixner, "Adaptive parallelism for web search," in *EuroSys*, 2013.
[18] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee *et al.*, "Clearing the Clouds: A Study of Emerging Scale-out Workloads on Modern Hardware," in *ASPLOS*, 2012.
[19] E. Le Sueur and G. Heiser, "Dynamic voltage and frequency scaling: The laws of diminishing returns," in *HotPower*, 2010.
[20] "Xapian," https://xapian.org/.
[21] G. Urdaneta, G. Pierre, and M. van Steen, "Wikipedia workload analysis for decentralized hosting," *Computer Networks*, Jul. 2009.
[22] D. Wong, "Peak efficiency aware scheduling for highly energy proportional servers," in *ISCA*, 2016.
[23] M. Xu, Y. Shang, D. Li, and X. Wang, "Greening data center networks with throughput-guaranteed power-aware routing," *Computer Networks*, vol. 57, no. 15, pp. 2880–2899, 2013.