PopCorns: Power Optimization using a Cooperative Network-Server approach for Data Centers

Bingqian Lu, Sai Santosh Dayapule, Fan Yao, Jingxin Wu, Guru Venkataramani, Suresh Subramaniam
Department of Electrical and Computer Engineering
The George Washington University
Email:{bqlu, saisantoshd, albertyao, jingxinwu, guruv, suresh}@gwu.edu

***Abstract*—Data centers have become a popular computing platform for various applications, and account for nearly 2% of total US energy consumption. Therefore, it has become important to optimize data center power, and reduce their energy footprint. With newer power-efficient design in data center infrastructure and cooling equipment, active components such as servers and the network consume most of the power with emerging sets of workloads. Most existing work optimizes power in servers and networks independently, and do not address them together in a holistic fashion that can achieve greater power savings. In this paper, we present PopCorns, a cooperative server-network framework for power optimization. We propose power models for switches and servers with low-power modes. We also design job scheduling algorithms that place tasks onto servers in a power-aware manner, such that servers and network switches can take effective advantage of low-power states. Our experimental results show that we are able to achieve more than 20% higher power savings compared to a baseline strategy that performs balanced job allocation across the servers.**

***Keywords*-Power Optimization, Data Center, Low Power States, Job scheduling algorithms**

I. Introduction

Data centers have spurred rapid growth in computing, and an increasing number of user applications have continued to move to cloud settings in the past few years. With this growing trend, data centers now account for about 2% of US energy consumption [1]. Many public cloud computing environments have power consumption on the order of several Gigawatts. Therefore, power optimization is a key challenge in data centers.

Data center servers are typically provisioned for peak performance to always satisfy user demands. This, however, also translates to higher power consumption. Hardware investments have resulted in power saving mechanisms, such as dynamic voltage and frequency scaling (DVFS) and low-power or idle states [2]. Through using such mechanisms, as servers continue to become more energy-proportional, other active data center components such as the data center network are growing to dominate overall power [3] [4]. Therefore, it is important to holistically address the power consumed by both data center servers and networks for an effective solution to this problem.

We note that power reduction strategies in network switches and routers have been studied in large-scale network settings. Gupta et al. [5] proposed a protocol-level support for coordinated entry into low-power states, where routers broadcast their sleep states for routing decisions to be changed accordingly. Adaptive link rate (ALR) for Ethernet [6] allows the network links to reduce their bandwidth adaptively for increased power efficiency. Such approaches may not be very effective in data center settings where application execution times have a higher dependence on network performance and Quality of Service (QoS) demands by the users.

In this paper, we propose PopCorns, a framework to optimize data center power through a cooperative network-server approach. We propose a power model for data center servers and network switches (with support for low-power modes) based on power measurements in real system settings and memory power modeling tools from MICRON [7] and Cacti [8]. We then study job placement algorithms that take communication patterns into account while optimizing the amount of sleep periods for both servers and switch line cards. Our experimental results show that we can achieve more than 20% energy savings compared to a baseline server load-balancing strategy.

We note that further power savings can be obtained at the application level through careful tuning for usage of processor resources [9] [10] or through load-balancing tasks across cores in multicore processor settings to avoid keeping cores unnecessarily active [11]. Such strategies can complement our proposed approach, and boost further power savings in data center settings.

In summary, the contributions of our paper are:

1. We propose a power model for data center servers and network switches based on available power measurements in real system settings and memory power modeling tools. We formulate a power optimization problem that jointly considers both servers and switches.

2. We propose an algorithm that considers servers and networks to co-ordinate server task placement while considering power drawn by network components. Transition between power states in switches is controlled by buffer sizes and certain traffic thresholds.

3. We compare our approach against a typical server load-balancing mechanism for task placement in terms of overall data center power consumption, and job latency. Our experimental results show that we are able to achieve more than 20% power savings.

II. System Model

In this section, we describe our power models for switches and servers, as well as models for jobs and the whole system.

A. Modeling Switch Power

1) Switch Port: To tackle the energy consumption in network equipment, the IEEE 8022.3az standard introduces the Low Power Idle (LPI) mode of Ethernet ports, which is used when there is no data to transmit, rather than letting the port be in active state all the time. The idea behind LPI is to refresh and wake up the port when there is data to be transmitted; the wakeup duration is usually small. Also, the energy consumption of a physical layer device (PHY) in LPI mode is significantly lower than when it is in the active mode [12].

We assume that there are three power states for each switch port: active, LPI, and off, and the corresponding average power consumption for each state under different link rates is shown in Table I. These power numbers are based on measurements on Intel's 82573L Gigabit Ethernet Controller [13].

Link Rate (Mbps)	**Active (mW)**	**LPI (mW)**
0	0	0
10	504	194
100	483	314
1000	1217	1010

Table I: Average port power consumption in active and LPI states.

2) Network Switch: Due to lack of detailed power models for commercial switches, we propose and derive our switch power model based on available data and memory power modeling tools from Micron [7]. A switch consists of several components such as ASICs, TCAMs, DRAM memory, and ports. Our power model for each component is explained below:

1. ASICs/Network processors: Operations such as parsing the packet contents to read the header, looking up routing tables, and forwarding to the corresponding destination port are performed by the network processor. According to Wobker's report [14], this consumes 52% of the total power in enterprise Cisco line cards. Accordingly, the ASIC/network processor's power consumption is computed to be 165 W. Based on studies done by Iqbal et al. [15] and Luo et al. [16], we assume a 60% power reduction due to power gating of ALU and micro engines in the network processor.

2. DRAM memory: The active power consumption of DRAM depends on the frequency of accesses, and leakage/static power depends on the transistor technology. Micron Power calculators [7] for RLDDR3 (Reduced latency DRAM) show a power consumption of 1571 mW per 1.125 GB of memory when active and 314 mW when in sleep. We assume the line card to integrate 8 such memory chips.

3. TCAM (Ternary content addressable memory): A typical 4.5 Mb TCAM structure which is used to offload high-speed packet lookup, consumes 15 W of power [17]. We model the static leakage power for a 4.5 Mb CAM structure using Cacti [8], which estimates the power consumed during the idle sleep period when memory is not accessed.

4. Line card interconnect fabric: The line card interconnect fabric consumes 23W during active power state [18].

5. Host processor: Each line card includes a host processor which is used in line card boot and initialization process for copying routing table information from the switch fabric card. The processor is kept running in sleep mode to keep the routing tables synchronized and to wake up the line card on packet arrival. We assume a 30% power reduction due to dynamic frequency scaling during line card sleep [19].

6. Ports: Combining the port LPI model in Section II-A1, for a typical line card with 48 ports at 1 Gbps, we calculate the total port power.

7. Local power supply: Based on Liu et al. [19], we assume a 50% reduction in the power loss due to reduced power consumption during the sleep state.

Table II shows the power consumption used in our experiments.

Component	Active Power (W)	Sleep Power (W)	Sources
ASICs/NW processor	165	66	[18] [15]
TCAM/Fwd. Engines	15	1.26	[17] [8]
DRAM	12.5	2.4	[7]
Interconnect Fabric	23	23	[18]
Host Processor	29	20	[18] [19]
Ports	48 x 1.21	48 x 1.01	[13]
Local Pwr supply	30	15	[18] [19]
Total	332	176	

Table II: Power model for a 48-port switch.

Apart from the line cards, we model a constant baseline power of 120 W for the rest of the switch in ON state, which includes switch supervisor module, the backplane, cooling systems, and switch fabric card, based on Pan et al. [18]. We consider a 25 µs wakeup latency from sleep state for the line card [18].

B. Modeling Server Power

We use the power consumption model of an Intel Xeon E5-based server from our prior work [20]. For simplicity, we model servers that can run a single task at a given time and utilize C6 (package sleep) state when the processor is idle. Our server power model is shown in Table III.

Active State	C6 Sleep state	Wakeup Latency	Source
92W	37W	1ms	[21] [20]

Table III: Server power model.

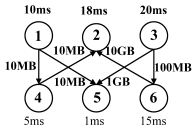


Figure 1: Example of a job DAG. Nodes represent tasks, with execution latency showed next to them. Edges represent flows and their sizes.

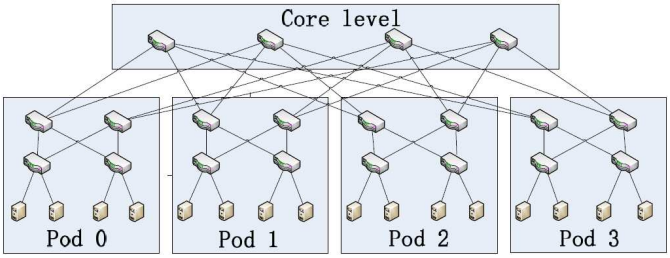


Figure 2: Fat tree topology.

C. Modeling Job

We model the execution of jobs at the server side as follows. A job consists of multiple inter-dependent tasks that include both spatial and temporal inter-dependence. Application tasks are typically executed by specific server types. For example, a web service request will first be processed by an application or web server, and a search request is processed by a database server, and this kind of task relationship is called spatial inter-dependence. In terms of temporal inter-dependence, a task cannot start executing until all of its 'parent' tasks have finished their execution, and until after their results have been communicated to the server assigned to the task. A job is considered to have finished when all of its tasks finish execution. A server core can only process one task at a time; we assume single-core processors in this paper and leave multicore processors for future work.

Each job j can be represented as a directed acyclic graph (DAG) $G^j(V^j, E^j)$, where V^j is the set of tasks of job j. In the DAG, if there is a link from task i to task r, then task i^j must finish and communicate its results to task r^j before r^j can start processing. Each task $v^j \in V^j$ has a workload requirement, namely task size or execution time requirement w_v^j for the core. For each link in E^j, there is a network flow with size D_l^j assigned to it, which denotes the transmission of results over link l (from the task at the head of DAG link to the task at the tail). Figure 1 shows an example of a job DAG.

D. Data Center System

Figure 2 shows the classic fat tree topology used in our network-server system [22]. In our system, each switch consists of a number of distributed cards plugged into the backplane, which provides the physical connectivity [18].

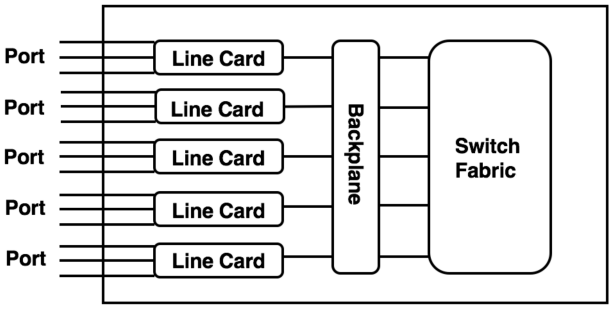


Figure 3: Schematic of switch, line card, and its ports.

Among these cards, there are multiple line cards for forwarding packets or flows, and can be in active, sleep, or off state. In turn, each line card contains several ports connecting to external links, which can also be in active, LPI, or off state. A typical schematic of switch, line card, and port is shown in Figure 3.

III. Problem Statement

In this section, we formulate joint server and data center network power optimization as a constrained optimization problem using the switch power model and job model defined above. First, to obtain the power consumption of a switch k, assume the number of active line cards and ports are ζ_k^{active} and ρ_k^{active} respectively, and the number of line cards in sleep state and ports in LPI mode are ζ_k^{sleep} and ρ_k^{LPI} respectively. Since the total power of a switch is the sum of base power (P_k^{base}), power of ports, and power of line cards, we have $P_k^{switch} = P_k^{base} + \zeta_k^{active} * P_{linecard}^{active} + \rho_k^{active} * P_{port}^{active} + \zeta_k^{sleep} * P_{linecard}^{sleep} + \rho_k^{LPI} * P_{port}^{LPI}$. To calculate the power consumption of server i, since our system considers the core as the basic processing unit in a server, the total power of a server is the sum of idle power P_i^{idle} and dynamic power, which is linear in the number of active cores C_i^{on}. Then, we have $P_i^{server} = P_i^{idle} + C_i^{on} * P_{core}^{on}$, where P_{core}^{on} denotes the power consumed by an active core.

Then the joint power optimization problem can be formulated as: minimize $\sum_{k=1}^{N_{switch}} P_k^{switch} + \sum_{i=1}^{N_{server}} P_i^{server}$ under both network-side and serve-side constraints, such as link capacity, computation resources, etc.

IV. Solution Approach

Given a set of jobs, modeling the joint power optimization problem as an Integer Linear Programming (ILP) formulation is a solution, and optimization tools like MathProg can be used to provide a near-optimal result. However, the computation complexity increases exponentially with the number of servers and switches [23]. In a typical data center with tens of thousands of servers and hundreds of switches, it is computationally prohibitive to solve the optimization

Symbol	Description
T_s	Traffic threshold for waking up a switch port from LPI state
T_f	Traffic threshold for waking up a switch port from off state
T_a	Traffic threshold for a port to enter LPI state
Q_{iL}	Current traffic load of port i in line card L
τ_{wakeup}^{port}	Port wakeup latency from LPI state
τ_{wakeup}^{LC}	Line card wakeup latency from sleep state
τ_{off}^{port}	Port wakeup latency from off state
τ_{off}^{LC}	Line card wakeup latency from off state
d_{wakeup}^{port}	Port wakeup delay
d_{wakeup}^{LC}	Line card wakeup delay
d_{LPI}^{port}	Delay for a port to enter LPI state
d_{sleep}^{LC}	Delay for a line card to enter sleep state
d_{off}^{LC}	Time threshold for a line card to enter off state

Table IV: Notations in PopCorns switch state transition algorithm.

problem. We therefore propose a computationally-efficient heuristic algorithm in this work.

A. Heuristic Algorithms

In this section, we first present a power management algorithm for line card and port power management, a simple power transition algorithm for servers, and then propose a joint job placement and network routing algorithm for solving the optimization problem efficiently.

1) Switch State Transition Algorithm: As not all switches need to be active all the time, if we can intelligently control the transitions to active and low-power states for ports and line cards, then DCN power consumption could be reduced. Therefore, we propose the Switch State Transition Algorithm 1 to implement network power management. The notations are elaborated in Table IV. An overview of our approach is shown in Figure 4. We assume that there is a global controller that keeps record of all the line cards, ports, and server status, including their power state and queue size. The global controller also monitors the current traffic load (number of pending flows or packets) at each port and decides whether the current line card can enter low-power state. In our design, we consider the line card and port power states and their transition as follows: if a line card is in sleep or off state, then all the ports are also in LPI or off state; if a line card is active, then its ports can be in LPI or active state, and ports in LPI state can be woken up to become active with a small latency τ_{wakeup}^{port}. When a flow arrives at port i of line card L, if L is in sleep state and current traffic load Q_{iL} of i exceeds threshold T_s, then both i and L will be woken up, with wakeup latency τ_{wakeup}^{port} and τ_{wakeup}^{LC} respectively. In contrast, if Q_{iL} is below T_s, then the flow is buffered in the queue of port i, and i and L are scheduled to be woken up after wakeup delay d_{wakeup}^{port} and d_{wakeup}^{LC} respectively. However, if other flows arrive during this wakeup delay and cause Q_{iL} to exceed T_s, the scheduled wakeup is canceled and i and L begin waking up immediately. When a flow is transmitted from i of L, if Q_{iL} is below T_a, then i will enter LPI mode after delay d_{LPI}^{port}. If all the ports of L are in LPI state, L enters sleep state after d_{sleep}^{LC}. Once the sleeping period of L exceeds d_{off}^{LC}, L together with its ports enter the off state immediately. For a port i in off state, once its Q_{iL} accumulates to T_f, both i and L begin waking up with latency τ_{off}^{port} and τ_{off}^{LC}.

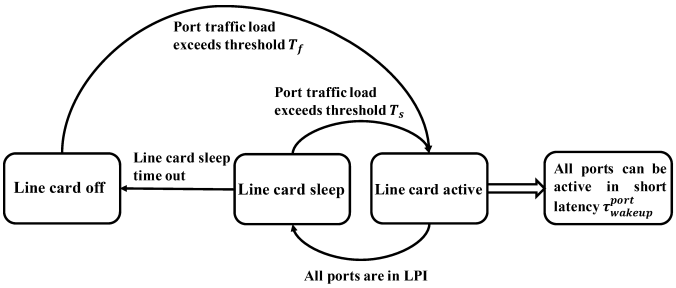


Figure 4: Power state transition overview.

2) Server State Transition Algorithm: As mentioned in Section II-B, we simply model two power states for the server: active and C6 sleep state. Algorithm 2 describes our policy on managing the server power states. We assume that each server has a local queue to buffer the tasks. When dispatching tasks, to select a pool of available servers, we first consider active servers with queue size below threshold T_s^{server}. If all the active servers have a queue size larger than T_s^{server}, then these servers will be considered available too. However, if all the servers are in sleep state, then some of them must be woken up to execute incoming tasks. Algorithm 3 explains which servers to choose in this case. After selecting available servers (S_{ava} in Table V), specific task placement policy is also described in Algorithm 3.

3) Cooperative Network-Server Algorithm: The main idea of our algorithm is to jointly consider the status of the pool of servers and the network before assigning jobs. To be more specific, for a job consisting of several pairs of interdependent tasks, if we place task pairs based on their interdependence, and choose the core pair with the minimum routing cost (in terms of power), then the placement together with its corresponding routing path can be more energy-efficient.

Based on this idea, we propose the Cooperative Network-Server (CNS) algorithm shown in Algorithm 3. The notations are shown in Table V. As mentioned in the previous section, we have a global controller to keep track of all the line cards, ports, and server status. Thus, when a job consisting of a set of interdependent tasks arrive, we first check the servers to select all server pairs whose power states are active and local queue sizes do not exceed a threshold. If no server satisfies these requirements, then servers with full local queue and servers in C6 sleep state will be selected. The set of eligible servers is called as S_{ava}. Note that if a task is assigned to a server in sleep state, then the server will be woken up immediately and will enter active state after a

Algorithm 1: Switch State Transition Algorithm

Input: T_s, T_a, Q_{iL}
Output: Line card and port power state transition

Initialization: All line cards are in sleep state, all ports are in LPI state;
while *there are jobs to be executed* **do**
  if *a flow arrives at port i of line card L* **then**
    if *L is in sleep state* **then**
      if $Q_{iL} > T_s$ **then**
        L begins waking up from sleep state;
        i begins waking up from LPI state;
      end
      else
        L begins waking up after d^{LC}_{wakeup};
        i begins waking up after d^{port}_{wakeup};
      end
    end
  end
  if *a flow is transimitted from port i of line card L* **then**
    if $Q_{iL} < T_s$ **then**
      i enters LPI state after d^{port}_{LPI};
      if *when i enters LPI state, all the ports of L are in LPI state* **then**
        L enters sleep state after d^{LC}_{sleep};
        if *sleeping period of L exceeds* d^{LC}_{off} **then**
          L and all its ports enter off state immediately;
        end
      end
    end
  end
end

Algorithm 2: Server State Transition Algorithm

Input: server wakeup latency τ^{server}_{wakeup}, local queue of server S
Output: Server power state transition

if *a task arrives at a server S* **then**
  if *S is in sleep state* **then**
    S enters active state after τ^{server}_{wakeup};
  end
  else
    task is put in the local queue of S;
  end
end
if *a task is finished executing at server S* **then**
  if *local queue of S is empty* **then**
    S enters sleep state immediately;
  end
end

Symbol	Description
Q_s	Local queue size of server s
T^{server}_s	Local queue size threshold for server s
S	All the servers in DCN
S_{ava}	All the available servers for executing tasks
$P_{x,y}$	Routing path between node x and y in DCN
PC^{port}_p	Power consumption of an arbitrary port p
PC^{LC}_{lc}	Power consumption of an arbitrary line card LC
PC^{LC}_{active}	Power consumption of an active line card
PC^{port}_{active}	Power consumption of an active port

Table V: Notations in PopCorns Cooperative Network-Server (CNS) algorithm.

wakeup latency.

For each server pair in S_{ava}, we compute the shortest routing path in terms of the *additional* power consumption incurred in the current system state, if that server pair is selected for executing a pair of interdependent tasks. Note that, along each eligible path, line cards could be active, sleeping, or off, and the corresponding ports could be active, LPI, or off. Upon assigning a path to the server pair, all the line cards and corresponding ports should become active. In other words, we need to wake up the inactive line cards and ports on the path, which requires extra power consumption. Based on this reasoning, we set the weight of each link as the additional power needed for the tail node on this link to be active. In other words, for link l from a server (or a switch) to switch sw, via line card lc and port p in sw, the weight of l is $w(l) = (PC^{LC}_{active} - PC^{LC}_{lc}) + (PC^{port}_{active} - PC^{port}_{p})$. Based on Table II, PC^{LC}_{active} = 273.92W, PC^{port}_{active} = 1.21W, PC^{LC}_{lc} = 127.52W if lc is in sleep state, PC^{LC}_{lc} = 0 if lc is in off state, PC^{port}_{p} = 1.01W if p is in LPI state, and PC^{port}_{p} = 0 if p is in off state. For a link from a switch to a server, the weight is obviously 0. After assigning link weights as described, the CNS algorithm finds the shortest path $P_{x,y}$ for every pair of servers (x,y) in S_{ava}. Then, it selects the server pair that has the minimum shortest path length, and assigns it to the task pair.

V. Evaluation

A. Experimental Setup

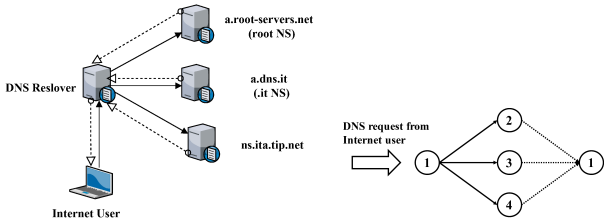
Our experiments are performed using an event-driven simulator [20], [24]. We simulate a data center cluster with 16 servers and 20 switches. The network is configured using a fat tree-topology as shown in Figure 2. We simulate three classes of real applications: *web service* (small-sized workload), *web search* (medium-sized workload), and *DNS service* (large-sized workload). For each of the representative workloads, we generate synthetic job arrivals with different utilization levels (10% for low, 30% for average, and 60%

Algorithm 3: Cooperative Network-Server Algorithm

Input: Q_s, T_s^{server}, line cards and ports power state, task dependency within a job

Output: Job placement and corresponding routing path

while *job j consisting of task set* T^j *arrives* **do**
  for *each pair of interdependent tasks* (T_m^j, T_n^j) *in* T^j **do**
    select S_{ava} from S;
    for *each pair of available servers* (x, y) *in* S_{ava} **do**
      compute shortest path $P_{x,y}$ for server pair (x, y);
    end
    choose the least-weight path among all the $P_{x,y}$, together with its corresponding server pair for task pair (T_m^j, T_n^j);
  end
end



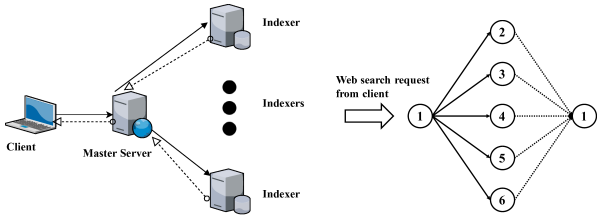
(a) Task workflow for DNS resolution. (b) DAG of a DNS job.

Figure 5: DNS service communication patterns (left) and DAG of DNS jobs (right). Each job consists of 4 tasks, and there are interdependence and communication between task 1 and 2, 3, 4.

for high utilization [25]). Random job arrivals are modeled by Poisson process. The communication patterns and the DAG for the three workloads are illustrated in Figures 5, 6, and 7. The switch and server powers under different sleep states are based on Tables II and III. Besides, we set the server wakeup latency as 1ms and local queue size threshold T_s^{server} (for Algorithm 2) as 10 (number of tasks). For the traffic thresholds and latency values in Table IV, we set T_s, T_a, and T_f as 10, 2, and 20 (in terms of the number of pending flows) respectively; τ_{wakeup}^{LC} as 25µs, τ_{wakeup}^{port} as 18.5µs, τ_{off}^{LC} as 2s, τ_{off}^{port} as (2s+2ms), d_{wakeup}^{LC} as 2ms, d_{wakeup}^{port} as (2ms+2µs), d_{LPI}^{port} as 1µs, d_{sleep}^{LC} as 1µs, and d_{off}^{LC} as 2s.

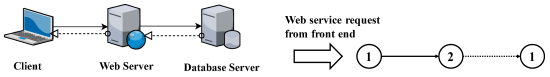
B. Evaluation Methodology

There are two major objectives: first, to demonstrate that the smart use of line card sleep state combined with port LPI state (shown in Algorithm 1), does reduce power consumption compared to no power management policy on the switches; second, our proposed Cooperative Network-Server Algorithm 3 can further save power, compared to other job placement algorithms that do not consider network



(a) Task workflow for a web search request. (b) DAG of a web search job.

Figure 6: Web search communication patterns (left) and DAG of a web search job (right). Task 1 is executed by the master server, while tasks 2 - 6 are processed by different indexers, and master server communicates with all the indexers.



(a) Task workflow for a web service request. (b) DAG of a web service job.

Figure 7: Web service communication patterns (left) and DAG of a web service job (right). Task 1 is executed by application server, while task 2 is assigned to database server.

and server status jointly. The baseline comparison policy in our experiment is the Server-Balanced (SB) Algorithm for task placement, which dispatches tasks uniformly to all the servers (i.e., n_{th} task allocated to server with index $n\%N$), but does include our proposed switch power management (Algorithm 1). After tasks are scheduled on servers, the Server-Balanced Algorithm uses single shortest path routing (e.g., using Dijkstra's algorithm) for interdependent task communication in the network. Thus, as long as there is an inactive line card or port on the routing path, it needs to be woken up. Note that since tasks of the same job are placed on adjacent servers, the communication between interdependent tasks would mostly be within a pod (see Section II-D), which involves only 2-4 hops. Therefore, this baseline algorithm might already perform better than random job placement in terms of DCN power consumption.

In our study, we break the power savings into two parts: network and server power saving. We calculate the power consumption of the network part with our network power management policy shown in Algorithm 1, and compare it with the case when all the line cards and ports are kept active all the time. To study the effectiveness of joint server-network power optimization, we set up two different job placement configurations: PopCorns framework (shown using the suffix of $-CNS$ to denote PopCorns' core Cooperative Network-Server Algorithm) and Server-Balanced Algorithm (shown using the suffix of $-SB$).

C. Evaluation Results

1) DNS Service: Figure 5a and Figure 5b show the task workflow for DNS service and its corresponding job DAG.

In the simulation, we set the flow size to be 100MB and the average job service time is randomly generated between 150ms and 200ms. Since our experiment is based on multi-task job, we define the service time as average task size (e.g., 75 - 100ms for a DNS task, based on job DAG), and the system utilization rate is defined as *service time∗task arrival rate*.

Figure 8a shows the power saving using the PopCorns framework for the execution of 2000 DNS jobs, compared to the baseline. In our experimental setup, if all the line cards and ports remain active, the switch power is 398.76W. We can see that for each configuration, the average power of PopCorns is 235 - 293W, and, PopCorns is able to achieve approximately 26 - 41% network power savings, while the baseline policy also saves 16 - 39% network power with smart use of both line card and port low-power states, compared to no power management on line cards and ports. Note that with increase in system utilization rates, the average power consumption of switches decreases. This is because larger system utilization means larger job arrival rate. Then, task pairs arriving at almost the same time will be assigned the same routing path, making the most of active line cards, while causing longer sleeping or off periods for the other line cards.

Additionally, we see that on the server side, PopCorns consumes approximately 26 - 30% less power than the baseline (Server-Balanced job scheduling). There is no power saving on the servers for the latter algorithm, as all the servers are kept active, and each consumes 92W. In this sense, we can conclude that intelligently scheduling tasks based on their interdependence and system status, together with the use of server C6 package sleep state further saves power.

Figure 8b, 8c, and 8d show the CDF of job latency for DNS Service jobs. We can see that the 90^{th} percentile job latency is the best under low system utilization, and gradually increases as utilization increases. WASP [20] shows that low system utilization is a more practical case in real scenarios. Thus, we note that our proposed power management policy together with job placement algorithm saves a large portion of power in DCN while maintaining the QoS demands (especially under low system utilization).

2) Web Search: Figure 6 shows the task workflow for a web search request and its corresponding job DAG. In the simulation, we set the flow size to be 100MB, and average job service time is randomly generated between 20ms and 60ms. Thus, the average web search task size is 10 - 30ms in our job model.

Figure 9a shows the power saving corresponding to the execution of 2000 DNS jobs, compared to the baseline. We can see that for each configuration, PopCorns is able to achieve about 60% network power savings under high system utilization, and even with low utilization, about 23% network power savings is achieved. In contrast, the baseline Server-Balanced job scheduling shows 46% and 23% power saving respectively, which demonstrates the benefits of smart use of low power states of switch components.

On the server side, PopCorns further saves approximately 18% power than the baseline, which has no power saving on the servers. The results are basically in accordance with the previous experiment.

Figures 9b, 9c, and 9d show the CDF of job latency in the Web Search experiment. We can see that the 90^{th} percentile job latency is also the best under low system utilization, and gradually increases with utilization rate getting higher.

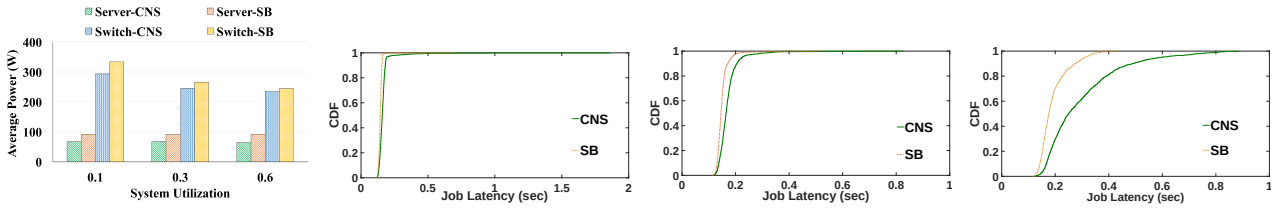
3) Web Service: Figure 7 shows the task workflow for a web service job and its corresponding DAG. In our simulations, we set the flow size to be 100MB, and average job service time is randomly generated between 2ms and 10ms. Therefore, the average web service task size is 1 - 5ms in our job model.

Figures 10 shows the power consumption for the execution of 2000 web service jobs, which reinforces our conclusions from prior experiments. We can see that for a DCN executing small-sized jobs like *web service*, smart use of server sleep states saves about 30% power compared to keeping servers active all the time. We observe about 23% network power saving under various system utilization levels. Additionally, compared to *DNS* and *web search* jobs, we can see that our framework achieves even higher power savings for larger-sized workloads.

VI. Related Work

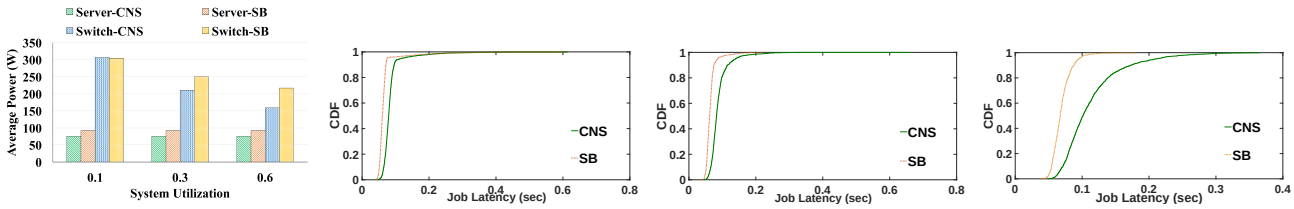
With the energy consumption of large data centers reaching Gigawatt scales, its energy saving techniques are increasingly being studied in recent years. Common techniques used for server energy reduction include DVFS to reduce the energy at the cost of server performance [26], co-ordinated DVFS and sleep states for server processors [27] [20] [21], and virtualization to consolidate VMs into fewer servers [28]. TS-Bat [27] demonstrates that, through temporally batching the jobs and by grouping them onto specific servers spatially, higher power savings can be obtained. WASP [20] shows that intelligent use of low power states in servers can be used to boost server power savings.

For the energy efficiency in the network, earlier works have looked at switches and routers for Internet-scale large area networking. Gupta et al. [5] first proposed the need for power saving in networks and pointed to having network protocol support for energy management. Adaptive Link Rate (ALR) [6] reduces link energy consumption by dynamically adjusting data link rates depending on traffic requirements. Other approaches include turning off switches when not required, or to put them in sleep mode depending on packet queue length [29] [30]. Prior works on reducing data center network power rely on DVFS and sleep states [15] to opportunistically reduce power consumption of individual switches. In these approaches, switches may



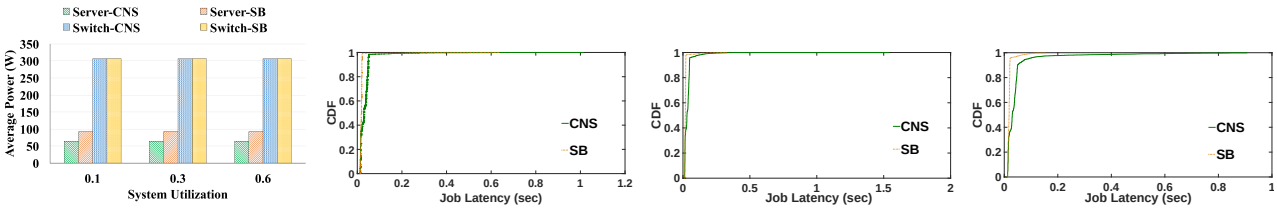
(a) DNS average power (b) DNS, Utilization=0.09 (c) DNS, Utilization=0.3 (d) DNS, Utilization=0.6

Figure 8: Average power and latency for 2000 Poisson-arrival DNS jobs under low, medium, and high system utilizations, respectively. Suffix $-CNS$ represents PopCorns framework and $-SB$ denotes baseline Server-Balanced job scheduling algorithm.



(a) Web search average power (b) Web search, Utilization=0.1 (c) Web search, Utilization=0.3 (d) Web search, Utilization=0.6

Figure 9: Average power and latency for 2000 Poisson-arrival web search jobs under low, medium, and high system utilization respectively. Suffix $-CNS$ represents PopCorns framework and $-SB$ denotes baseline Server-Balanced job scheduling algorithm.



(a) Web service average power (b) Web service, Utilization=0.1 (c) Web service, Utilization=0.3 (d) Web service, Utilization=0.6

Figure 10: Average power and latency for 2000 Poisson-arrival web service jobs under low, medium, and high system utilization respectively. Suffix $-CNS$ represents PopCorns framework and $-SB$ denotes baseline Server-Balanced job scheduling algorithm.

enter sleep states without knowledge of incoming server traffic and may be forced to wake up prematurely. Hence, a server job placement and network allocation co-ordination approach is required to optimize the amount of sleep time in saving network energy consumption.

There are existing works that combine server and network power savings. Mahadevan et al. [31] and Heller et al. [32] have proposed a heuristic algorithm for a coarse-grained load variation approach which dynamically allocates the servers required for the workload and powers off the unneeded switches for the server configuration. Other approaches consolidate VMs in fewer servers and in turn use fewer switches [33]. These approaches assume an unrealistically high amount of idle period to be able to completely turn off the servers and network components. To the best of our knowledge, our solution is the only one to consider network sleep states to target higher power savings in the data center.

VII. Conclusion

In this paper, we presented PopCorns, that makes smart use of low power states in line cards and ports of network switches. We combine them with network-aware task placement for more effective power management. Our experimental results show that smart management of low-power states achieves up to 40% power reduction over always active switches, and smart joint placement and routing saves up to 60% power while keeping the job latency reasonably low.

Acknowledgement

This material is based in part upon work supported by the National Science Foundation under Grant CNS-178133.

References

[1] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *ACM SIGARCH Computer Architecture News*, ACM, 2007.

[2] HP, Intel, Microsoft, Phoenix, Toshiba, "Advanced configuration and power interface specification." http://www.acpi.info/.

[3] D. Abts, M. R. Marty, P. M. Wells, P. Klausler, and H. Liu, "Energy Proportional Datacenter Networks," in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ISCA '10, (New York, NY, USA), pp. 338–347, ACM, 2010.

[4] G. Koutitas and P. Demestichas, "Challenges for energy efficiency in local and regional data centers," *Journal on Green Engineering*.

[5] M. Gupta and S. Singh, "Greening of the Internet," in *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '03, (New York, NY, USA), pp. 19–26, ACM, 2003.

[6] C. Gunaratne, K. Christensen, B. Nordman, and S. Suen, "Reducing the Energy Consumption of Ethernet with Adaptive Link Rate (ALR)," *IEEE Transactions on Computers*, vol. 57, pp. 448–461, Apr. 2008.

[7] Micron, "Micron rlddr3 memory power calculator."

[8] S. Li, K. Chen, J. H. Ahn, J. B. Brockman, and N. P. Jouppi, "Cacti-p: Architecture-level modeling for sram-based structures with advanced leakage reduction techniques," in *ICCAD: International Conference on Computer-Aided Design*, pp. 694–701, 2011.

[9] J. Chen, G. Venkataramani, and G. Parmer, "The need for power debugging in the multi-core environment," *IEEE Computer Architecture Letters*, vol. 11, no. 2, pp. 57–60, 2012.

[10] J. Chen, F. Yao, and G. Venkataramani, "Watts-inside: A hardware-software cooperative approach for multicore power debugging," in *2013 IEEE 31st International Conference on Computer Design (ICCD)*, pp. 335–342, Oct 2013.

[11] J. Oh, C. J. Hughes, G. Venkataramani, and M. Prvulovic, "Lime: a framework for debugging load imbalance in multi-threaded execution," in *International Conference on Software Engineering*, pp. 201–210, IEEE, 2011.

[12] P. Reviriego, V. Sivaraman, Z. Zhao, J. A. Maestro, A. Vishwanath, A. Sánchez-Macian, and C. Russell, "An energy consumption model for energy efficient ethernet switches," in *High performance computing and simulation (HPCS), 2012 International Conference on*, pp. 98–104, IEEE, 2012.

[13] A. Wertheimer and E. Mann, "Active/idle toggling with low-power idle."

[14] L. J.Wobker, "Power consumption in high end routing systems."

[15] M. F. Iqbal and L. K. John, "Efficient traffic aware power management for multicore communications processors," in *2012 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, pp. 123–133, Oct. 2012.

[16] Y. Luo, J. Yu, J. Yang, and L. Bhuyan, "Low power network processor design using clock gating," in *Proceedings. 42nd Design Automation Conference, 2005.*, pp. 712–715, June 2005.

[17] Q. Guo, X. Guo, Y. Bai, and E. İpek, "A resistive TCAM accelerator for data-intensive computing," in *2011 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 339–350, Dec. 2011.

[18] T. Pan, T. Zhang, J. Shi, Y. Li, L. Jin, F. Li, J. Yang, B. Zhang, X. Yang, M. Zhang, *et al.*, "Towards zero-time wakeup of line cards in power-aware routers," *IEEE/ACM Transactions on Networking (TON)*, vol. 24, no. 3, pp. 1448–1461, 2016.

[19] 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.

[20] F. Yao, J. Wu, S. Subramaniam, and G. Venkataramani, "Wasp: Workload adaptive energy-latency optimization in server farms using server low-power states," in *Cloud Computing (CLOUD), 2017 IEEE 10th International Conference on*, pp. 171–178, IEEE, 2017.

[21] F. Yao, J. Wu, G. Venkataramani, and S. Subramaniam, "A dual delay timer strategy for optimizing server farm energy," in *Cloud Computing Technology and Science (CloudCom), 2015 IEEE 7th International Conference on*, pp. 258–265, IEEE, 2015.

[22] F. Yao, J. Wu, G. Venkataramani, and S. Subramaniam, "A comparative analysis of data center network architectures," in *Communications (ICC), 2014 IEEE International Conference on*, pp. 3106–3111, IEEE, 2014.

[23] X. Wang, Y. Yao, X. Wang, K. Lu, and Q. Cao, "Carpo: Correlation-aware power optimization in data center networks," in *INFOCOM, 2012 Proceedings IEEE*, pp. 1125–1133, IEEE, 2012.

[24] D. Meisner, J. Wu, and T. F. Wenisch, "Bighouse: A simulation infrastructure for data center systems," in *Performance Analysis of Systems and Software (ISPASS), 2012 IEEE International Symposium on*, pp. 35–45, IEEE, 2012.

[25] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *ACM SIGARCH Computer Architecture News*, vol. 35, pp. 13–23, ACM, 2007.

[26] L. Wang, S. U. Khan, D. Chen, J. Ko\lOdziej, R. Ranjan, C.-Z. Xu, and A. Zomaya, "Energy-aware Parallel Task Scheduling in a Cluster," *Future Gener. Comput. Syst.*, vol. 29, pp. 1661–1670, Sept. 2013.

[27] F. Yao, J. Wu, G. Venkataramani, and S. Subramaniam, "Tsbat: Leveraging temporal-spatial batching for data center energy optimization," in *GLOBECOM 2017-2017 IEEE Global Communications Conference*, pp. 1–6, IEEE, 2017.

[28] C.-H. Hsu, K. D. Slagter, S.-C. Chen, and Y.-C. Chung, "Optimizing Energy Consumption with Task Consolidation in Clouds," *Inf. Sci.*, vol. 258, pp. 452–462, Feb. 2014.

[29] Q. Yu, T. Znati, and W. Yang, "Energy-efficient, Delay-aware packet scheduling in high-speed networks," in *2015 IEEE 34th International Performance Computing and Communications Conference (IPCCC)*, pp. 1–8, Dec. 2015.

[30] W. Si, J. Taheri, and A. Zomaya, "A distributed energy saving approach for Ethernet switches in data centers," in *37th Annual IEEE Conference on Local Computer Networks*, pp. 505–512, Oct. 2012.

[31] P. Mahadevan, P. Sharma, S. Banerjee, and P. Ranganathan, "Energy Aware Network Operations," in *Proceedings of the 28th IEEE International Conference on Computer Communications Workshops*, INFOCOM'09, (Piscataway, NJ, USA), pp. 25–30, IEEE Press, 2009.

[32] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown, "ElasticTree: Saving Energy in Data Center Networks.," in *Nsdi*, vol. 10, pp. 249–264, 2010.

[33] K. Zheng, X. Wang, L. Li, and X. Wang, "Joint power optimization of data center network and servers with correlation analysis," in *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, pp. 2598–2606, Apr. 2014.