A Dual Delay Timer Strategy for Optimizing Server Farm Energy

Fan Yao, Jingxin Wu, Guru Venkataramani, Suresh Subramaniam
Department of Electrical and Computer Engineering
The George Washington University
Washington, DC, USA
Email: {albertyao, jingxinwu, guruv, suresh}@gwu.edu

***Abstract*—Server farms are becoming increasingly energy-hungry with the growing popularity of web-based applications and services. Servers consume nearly 60% of peak power even when operating at relatively low server utilization levels of around 30%. Unfortunately, most server farms are generally provisioned to accommodate the peak load, and wasteful energy is often spent on unnecessarily keeping the servers active. Recent work on utilizing processor sleep states has mitigated the energy problem, but more opportunities to optimize energy remain to be explored. In this paper, we explore techniques that make smart use of the processor deep sleep states through augmenting them with dual delay timers for more effective energy management in the multi-server environment. We find that our exploratory studies on smarter use of processor sleep states with dual delay timers show good promise in achieving higher energy savings on different kinds of synthetic and real workloads. Our experimental results show that our techniques achieve up to 71% savings in energy over naive energy management without the use of low-power sleep states, and up to 31% energy savings over a relatively smarter energy management mechanism with just a single delay timer to enter the sleep state. We also show that the normalized latency of jobs on a server farm with our dual delay timer strategy is almost similar to the one that is always ready to accept incoming jobs.**

Keywords*—*Server Farm; Energy Optimization; Processor Sleep States; Dual Delay Timer

I. Introduction

Energy footprint of large-scale server farms, especially data centers, has grown rapidly, accounting for nearly 2% of the US domestic energy consumption [1], [2]. Increasing demand from users for personalized and contextual retrieval of large volumes of data and the associated computations have exacerbated the energy issues [3]. Most server farms have traditionally been provisioned for peak demand, and configured to operate at capacities much higher than necessary. Studies by Barroso et al. [4] have shown that the servers in data center environments are typically utilized at only 30% of their potential while drawing almost 60% of the peak power. This disproportionality in server utilization versus energy consumption occurs largely as a result of ineffective system-wide energy management techniques and the over-provisioning of servers without understanding or even considering the workload characteristics. Therefore, new and effective system-wide energy management techniques, that are aware of workload characteristics, are needed.

While several prior studies have considered DVFS (Dynamic Voltage Frequency Scaling) to achieve dynamic power savings, at relatively low processor utilization levels seen in many data center environments, utilizing processor *sleep states* or *low-power states* to conserve system energy can be even more effective. This is because, using the sleep states, the processor could halt the operation of (i.e., de-activate) various units on the chip or the motherboard to achieve both static and dynamic power savings. However, we note that waking up a processor to transition from a low-power state to active state is not instantaneous and incurs high performance overheads. Therefore, a smart use of processor sleep states is necessary to achieve higher energy savings in the server farm environment.

Recent work (e.g., [5]) has explored the use of sleep states in mitigating the energy issues; yet many more opportunities exist for energy optimization. In this paper, we propose the use of processor idle and deep sleep states combined with *dual delay timers* to orchestrate the entry and exit from these low-power states and maximize system energy savings across different types of workloads. We also study the effectiveness of our techniques for various job arrival patterns and real-world workloads such as Wikipedia trace [6].

We note that the energy savings can be further enhanced when our approach is used in conjunction with other features such as DVFS. For clarity purposes and to minimize the implementation complexity, we will limit our exploratory studies in this work to primarily using just the processor sleep states.

Exploring *autonomous* frameworks that perform system-wide energy management are essential for effective energy management in server farms. As users increasingly turn to using modern computing environments such as cloud computing, manual energy optimization or policy construction becomes an impractical task. Also, while server-level energy optimization strategies already exist, global energy management can provide much higher benefit than locally optimal energy saving policies. Therefore, we envision that our framework will have a direct benefit to data center operations.

In summary, the contributions of our paper are:

1) We propose a Dual Delay Timer based algorithm that makes smart use of the existing processor and platform sleep states to achieve higher energy savings in comparison to existing approaches that simply use a single delay timer strategy to enter and exit sleep states.

2) We investigate the relationship between server utilization and job service time combined with the use of processor low-power states and delay timers. We explore the effects of

these various parameters on overall system energy management.

3) We evaluate our energy-saving techniques for different job arrival patterns – random job arrivals modeled using *Poisson Process*, and bursty job arrivals modeled using *Markov Modulated Poisson Process (MMPP)*. We also show our framework's benefit with jobs of different sizes.

4) We show our experimental results with four synthetic workloads and a real system job trace from Wikipedia servers [6] that has a random (unknown) job arrival distribution. Our experimental results show that our techniques achieve up to 71% savings in energy over naive energy management without the use of low-power sleep states, and up to 31% energy savings over a relatively smarter energy management mechanism with just a single delay timer to enter the sleep state. We also show that the normalized job latency with our Dual Delay Timer strategy is similar to the latency in the case when the servers are always active and ready to execute jobs.

5) We explore the scalability of our proposed techniques in server farms by varying the number of servers from 20 to 100. Our results show good promise for scaling as we increase the number of servers.

II. System Model and Simulation Platform

A. Processor Low Power Modes

Emerging from embedded devices, low-power states are now an important feature targeted for power management in modern computer systems. The Advanced Configuration and Power Interface (ACPI) [7] provides a standardized specification for platform-independent power management. ACPI-defined interfaces have been adopted by several major operating system vendors [8], [9] and supported by various hardware vendors such as Intel and IBM [10], [11]. ACPI uses global states, *Gx*, to represent states of the entire system that are visible to the user. Within each *Gx* state, there is one or more system sleep states, denoted as *Sx*. For instance, *S0* is the working state and *S1* is the low-latency sleep state. When a computer system is in the *S0* state, the processor is allowed to reside in a set of *C* states such as *C0*, *C1* and *C2*. A higher-numbered C state typically indicates more aggressive energy savings but also corresponds to longer wake-up latency. Modern processors generally provide high parallelism by integrating multiple cores within one package. Typically, low-power sleep states are supported at both core level and package level. The package *C* state is automatically resolved to the shallowest sleep state among all the cores.

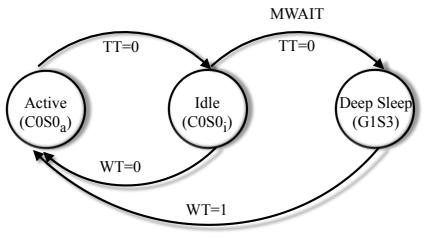

Fig. 1: Server power states and their corresponding Transition (TT) and Wakeup (WT) times (in seconds).

Components	Active $C0S0_a$	Idle $C0S0_i$	Deep Sleep $G1S3$
Cores [10], [5]	130 W	75 W	16 W
Chipset [5]	7.8 W	7.8 W	7.8 W
RAM [5]	23.1W	10.4 W	3.0 W
HDD [5]	6.2 W	4.6 W	0.8 W
NIC [5]	2.9 W	1.7 W	0.5 W
PSU [5]	70 W	35 W	1 W
Cooling [5]	10 W	1 W	0 W
Total Power	250 W	135.5 W	29.1 W

TABLE I: Power breakdown for an Intel Xeon-E5 based server.

In this paper, we use the low-power states from the Intel Xeon E-5 processor [10] (shown in Figure 1). Table I shows the power model illustrating the power consumption by various units in a given low-power mode. Note that the *C* states mentioned in our power model refer to package level *C* states. We conservatively assume that the processor consumes peak power in active state regardless of the actual number of busy cores. Similar assumptions are also adopted in [12], [13].

B. Server, Job, and Workload Model

We model the server farm as a multi-server system of homogeneous servers. Each server is equipped with an Intel Xeon processor that can process four jobs at a time. In this paper, we use four synthetic workloads with average workload execution times shown in brackets next to them: Google search (4.2 ms), Apache (75 ms), Mail (92 ms) and DNS query (194 ms) jobs [14], and one real system trace, Wikipedia, with an average workload execution time of 3.5 ms [6]. The term utilization factor ρ is defined as the fraction of time the server is expected to be busy executing jobs.

We assume that a system-wide load balancer dispatches jobs to the servers. The *job latency* is defined as the time elapsed from when a job arrives to when the job completes its execution and departs the server farm. In this paper, we monitor the average job latency (normalized with respect to the average expected execution time) and ensure that the worst case job latency is within bounds to meet Quality of Service constraints.

C. Simulation Platform

We have built an in-house event-driven simulator to analyze the energy savings of our approach. Even though several well-known data center and cloud simulators exist (such as [15], [16]), they do not fit our needs: we need a simulator that provides fine-grained modeling of sleep states and control of sleep state transitions as well as a basic framework to manipulate server power states in a centralized manner. Moreover, to analyze a wide range of workloads and applications, the simulator also needs to accept both synthetic workloads and realistic workloads from system traces. Our simulator has three major components: *workload generator*, *server power state/performance manager*, and *server farm job handler/load balancer*. The workload generator injects jobs into the system based on either a stochastic process (for synthetic workloads such as Google search, Apache, Mail, and DNS) or system job arrivals/service time traces collected from realistic data centers (for Wikipedia workload). The *power state manager* models various sleep states and is responsible for coordinating

the servers to enter or wake up from a specific sleep state. For example, the *power state manager* can request a server to enter deep sleep (*G1S3* state) immediately or after a certain delay timer value. Our simulator reports detailed statistics including the breakdown of server energy and performance measures such as the job latency characteristics.

D. Simulation Setup

To exploit the use of sleep states for energy optimization, we simulate a server farm with 50 four-core servers. In the rest of the paper, unless otherwise noted, we assume this to be our default configuration. We configure the duration of our simulations to be long enough for the type of workload – for example, the simulation length for DNS workload (with largest job size) to be 20,000 seconds, which means roughly 40,000 jobs would be processed per server on average at server utilization of 0.1 and an even higher number of jobs at higher utilization levels. We then scale the execution time for other workloads based on their job sizes. As a result, Mail, Apache, and Google search would have simulation times of 10000, 8000, and 1000 seconds, respectively. These simulation lengths are also configured to ensure that enough number of bursty and non-bursty phases would be observed in our experiments where we model non-uniform job arrivals (Section IV-C). Also, in all of our experimental results, we report the steady state statistics by disregarding the warm-up time period during the first 10000 jobs arrivals.

E. Workload Generation

We use three types of workload arrival models. By default, we use Poisson Process for job arrivals, which is widely used in prior works to model data center workloads [12], [14]. To model bursty patterns in job arrivals that are also typically seen in data center environments, we use Markov Modulated Poisson Process, a well studied model to simulate workload burstiness [17], [18], [19]. Aside from such analytical models, we also use Wikipedia workload, a realistic system trace available for public use [6]. For the first two cases, we simulate three different levels of system utilizations (0.1 for low utilization, 0.3 for average utilization [4], 0.6 for high utilization).

Poisson-based job arrivals: The job service times are modeled as a uniform distribution with a mean service time, $1/\mu$, where μ is the service rate of a server. Uniform distribution, rather than the usual exponential distribution, is assumed to prevent the workload generator from producing very short jobs that can severely deteriorate job latencies at the tail end. In a multi-core based server farm, the relation between system utilization ρ and job arrival rate λ is: $\rho = \frac{\lambda}{\mu * nServers * nCores}$, where $nServers$ is the number of servers and $nCores$ is number of cores per server.

MMPP-based bursty job arrivals: *MMPP* uses a continuous-time Markov chain to model different stages or states of the workload. Each state x corresponds to a Poisson Process with job arrival rate λ_x. By orchestrating the transitions among various states with high and low λs, *MMPP* is able to model workload *burstiness* at a finer-grain level. In our experiments, we use a 2-state *MMPP* model, in which one state has a high job arrival rate λ_h representing periods of bursty arrivals, and the other state has a low arrival rate (λ_l) and models non-bursty periods of operation. There are two approaches to tune the levels of burstiness – increasing the ratio of job arrival rates between bursty and non-bursty state, $R_a = \lambda_h/\lambda_l$, or decreasing the proportion of time the process stays in bursty state. Detailed exploration of workload burstiness modeling is a rich area of study [17]. In our experiments to characterize burstiness, we define the ratio between λ_h and λ_l, as well as the ratio between process durations. The job arrival rates are then translated to different utilization factors. λs in both states are computed and set so that the bursty workloads generated would have an average utilization factor of 0.1, 0.3, and 0.6, respectively. This is done to compare our results to Poisson-based workloads with the same system utilization factors. Table II illustrates the high ρ and low ρ (corresponding to the two states of the *MMPP* model) associated with different average utilization levels.

Utilization levels	High ρ	Low ρ	Window length
0.1	0.4	0.025	30 seconds
0.3	0.7	0.2	30 seconds
0.6	0.8	0.4	30 seconds

TABLE II: *MMPP* ρ values for bursty and non-bursty periods to achieve a certain *overall* system utilization level.

F. Job Handler

Our job handler uses the following algorithm: First, we check if an active server has an idle core. If so, the job handler schedules the job on a server with least number of idle cores. If multiple such servers exist, the handler picks one of them randomly and schedules the job on one of its cores. Second, if none of the active servers have any available cores to accommodate an incoming job, we check if an idle server exists. If so, we randomly wake up one of the idle servers and schedule the job on one of its cores. Third, if there are no active or idle servers to accommodate an incoming job, we check if a sleeping server exists. If so, we randomly wake up one of the sleeping servers and schedule the job on one of its cores. Fourth, if there are no available servers, the job is buffered until a server becomes available.

III. Exploiting Sleep States and Delay Timers for Energy Optimization

Sleep states have been implemented in most modern processors to reduce energy consumption, especially when the processor utilization levels are low. For example, when the processor operates at 10% of its peak capacity, the processor should ideally stay active for 10% of the time. For the remaining 90% of the time, the system should enter one of the low-power states or sleep states (that consumes significantly less power compared to the active mode), and ultimately approach energy proportionality. While sleep states are designed to lower the system energy consumption, one has to use them judiciously to effectively take advantage of their energy-saving benefits. To illustrate this, we perform motivational experiments that study the energy consumption of server farms under different workloads using three different server power configurations described below.

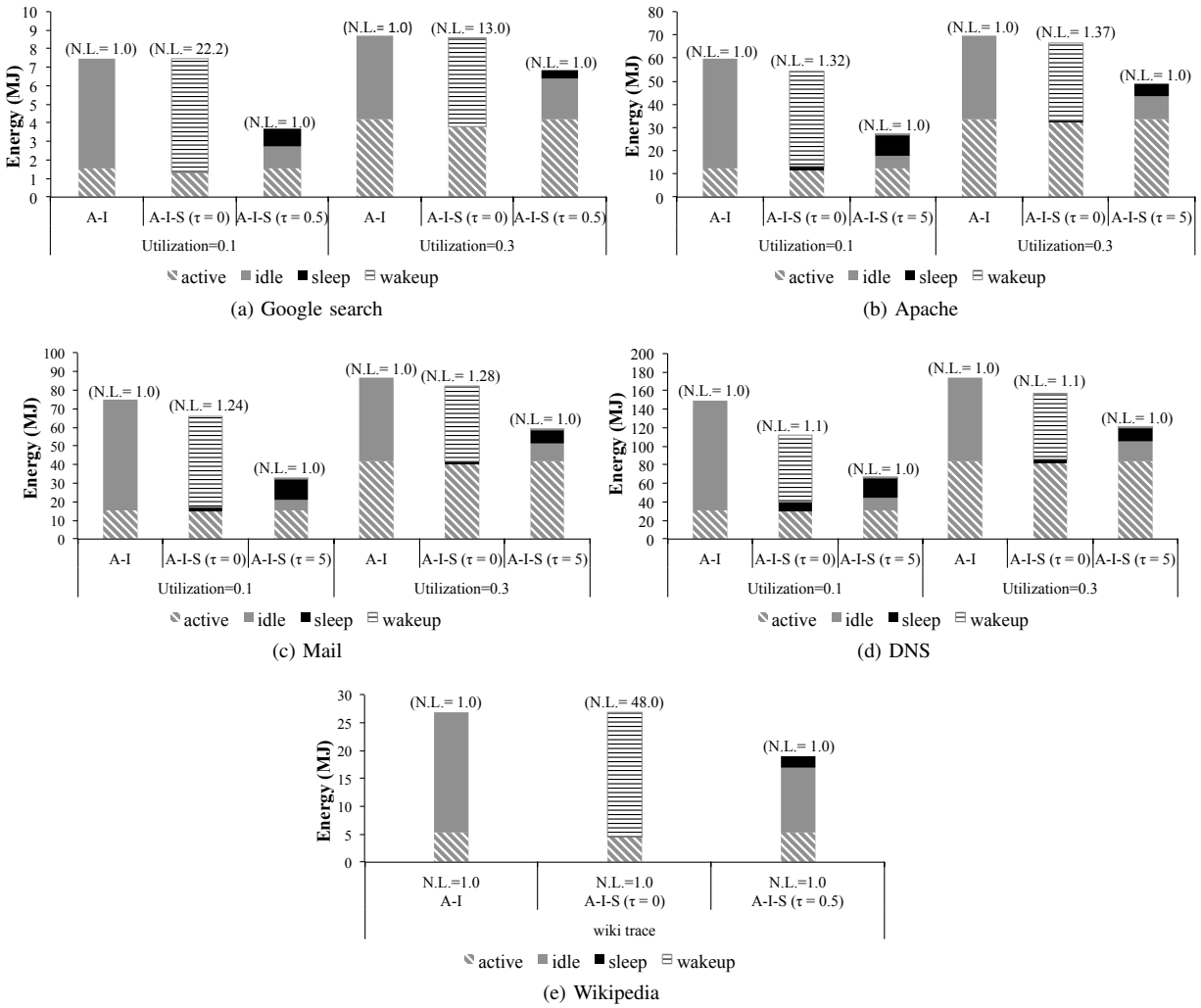

Fig. 2: Energy breakdown for various workloads using three different server power configurations. Average Normalized Latency (N.L.) is shown above the bars normalized to the workload execution times.

1) A-I is a power configuration where the server alternates between active and idle states. A server is active when at least one of the four cores within the server has a job to process. The server enters idle state if none of its cores has a job to process.

2) A-I-S ($\tau = 0$) is a power configuration where the server transitions between three states – active, idle, and sleep. τ denotes the *delay timer* for transition from processor idle to sleep state. The server is active when at least one of the four cores within the server has a job to process. The server enters the idle state when none of the cores within the server have jobs to process, and then, it immediately goes to deep sleep since the delay timer τ is set to zero. In other words, the server goes to deep sleep immediately whenever there are no jobs to be processed by the server.

3) A-I-S ($\tau = c$) is a power configuration that is identical to the above, except that the server goes to deep sleep from idle after a delay timer expires. In other words, the server waits for $\tau = c$ seconds before transitioning to deep sleep state after entering the idle state. If a job arrives before the delay timer reaches zero, the server gets back to active state. The optimal τ value is chosen based on an exponential sampling of τ values and the associated energy savings. We note that prior studies [13] and [20] have considered timer-based strategies to conserve energy. In [20], the authors used delay timers to transition between hypothetical sleep states, and in [13], the servers are completely switched off to conserve energy. As we show later in the paper, our new dual timer strategy significantly improves the energy savings over the single delay timer approach.

Figure 2 shows the results of our experiments for various workloads. In each workload, we study the A-I, A-I-S ($\tau = 0$) and A-I-S ($\tau = c$) configurations for two different processor utilization levels of 0.1 (low server utilization in data centers) and 0.3 (average server utilization in data centers observed by Barroso et al. [4]). The corresponding energy consumption (in millions of Joules) are shown on the y-axis with breakdown between processor active, idle, sleep, and wakeup cycles. Average normalized job latency is shown above each power configuration at a certain server utilization level. We note that in a majority of cases, A-I-S ($\tau = 0$) configuration does not significantly improve energy, and in fact, the idle energy spent in A-I configuration is simply translated into wakeup energy

for the processor to transition from sleep to active state in A-I-S ($\tau = 0$). This phenomenon actually diminishes the effectiveness of sleep state to save energy. Worse still, the latency impact of using A-I-S ($\tau = 0$) is extremely high, especially for workloads with short execution times such as Google search where we observe $22\times$ performance slowdown at a utilization level of 0.1, $13\times$ performance slowdown at a utilization level of 0.3, and Wikipedia with a $48\times$ performance slowdown.

For A-I-S ($\tau = c$) configuration where c is the optimal τ value, we find the following optimal values for τ: $\tau = 0.5$ seconds for short latency jobs such as Google search and Wikipedia, and $\tau = 5.0$ seconds for long latency jobs including Apache, Mail and DNS. In general, we note that A-I-S ($\tau = c$) configuration significantly reduces the overall energy consumption while keeping the average normalized job latency to be almost the same as that for A-I configuration. Through our experiments, we measured the normalized job latency to be 1.0 in A-I-S ($\tau = c$).

1) At average server utilization levels of 0.1, we observe as much as 55.4% energy reduction for Apache and up to 55.7% energy reduction in Mail workloads compared to their corresponding A-I configuration. We note that such significant energy savings are possible because a majority of servers now enter deep sleep state (consuming 88.4% less power than active mode). Most sleeping servers are rarely woken up and remain in deep sleep mode due to the incoming jobs being serviced by the servers in the idle/standby mode with a delay timer $\tau = c$.

2) At average server utilization levels of 0.3, we observe less energy savings compared to utilization level of 0.1 due to a higher rate of incoming jobs. We measure about 32.4% energy reduction in Mail and 30.1% energy reduction in DNS workloads compared to the corresponding A-I configuration. Beyond active energy (that is spent on actually servicing the jobs), we note that servers remain idle most of the time, thus leading to overall energy reduction.

3) In Wikipedia workload trace, we observe 29.2% reduction in A-I-S ($\tau = 0.5$) compared to A-I configuration.

IV. Dual Delay Timers: A Novel Strategy for Further Energy Optimization

In Section III, we observed that A-I-S ($\tau = c$) is able to achieve significant energy savings over A-I while having similar performance in terms of job latency. However, when the system utilization is low (say 0.1), short latency workloads such as Google search still consume around 30% of the peak energy. This is due to many servers still remaining in the idle mode after they are done servicing their jobs.

To explore the possibility of further savings in system energy consumption, we devise *Dual Delay Timers* or *Dual* τ. Dual τ is based on the following intuition: instead of keeping τ values to be the same for all the servers, further energy reduction could potentially be had by grouping the servers into two pools: one pool of servers with relatively high τ that offers to be the standby machines for the incoming jobs, and the second pool of servers with small τ values which can quickly go to deep sleep state. The best case scenario is when a small number of servers with high τ continue to stay in the standby mode to maximize the chances of accepting all of the incoming jobs, and the rest of servers with low τ quickly go to sleep resulting in optimizing the overall energy consumption.

A. Dual Delay Timer Algorithm

The power management policy for Dual Delay Timer augments the A-I-S configuration by designating a small fraction of the server pool with high τ values, while others have low τ values.

Symbol	description
V_{ai}	set of servers in active or idle state
V_s	set of servers in deep sleep
τ_h	high τ value
τ_l	low τ value
t_w	threshold for waking up a sleeping server

TABLE III: Notations in the Dual Delay Timer Algorithm.

Algorithm 1: Dual Delay Timer Algorithm

```
Input: t_w, n (total number of servers)
Initialization: V_ai = {s_1, s_2, ..., s_n};
/* By default, all servers are placed in
   V_ai and set into idle state                    */
for i in [1, n] do
    power state of s_i ← idle
end
/* arrived jobs are first placed in the
   queue                                           */
while there are unfinished jobs in queue do
    pick a new job j from the head of the queue;
    if at least one server with a free core exists in V_ai then
        find set of servers S in V_ai with highest utilization;
        if multiple servers exist in S then
            give preference to a server with τ_h;
            randomly pick a server, s_sch in S for scheduling;
        end
        schedule job j on a free core from s_sch;
        continue;
    end
    else
        compute the number of pending jobs in queue, p;
        if p > t_w then
            pick a server s_sch from V_s;
            give preference to a server with τ_h;
            add s_sch to V_ai;
            set s_sch to active state;
        end
    end
end
```

Algorithm 1 presents our Dual Delay Timer approach that accepts the incoming jobs and assigns them to servers. The corresponding notations are listed in Table III. Our dual delay timer approach designates a small set of servers with high τ and the rest of the servers with a low τ. The candidate servers in the two pools are rearranged periodically based on load balancing and fairness among all of the servers. When a new job arrives, the job handler will first try to assign this job to one of the schedulable servers, which could either be an active server with available cores or a server in idle state, denoted as V_{ai}. The algorithm will choose a server from V_{ai} with the highest utilization. This is done to favor the less utilized servers to enter idle state rapidly. If there are multiple servers with high

		Dual-τ							
Workload	Utilization	Energy Reduction over A-I	Energy Reduction over A-I-S (opt τ)	50%-ile N.L.	90%-ile N.L.	95%-ile N.L.	High τ	Low τ	Num. servers with High τ
Google	0.1	+61.05%	+21.49%	1.00	1.12	1.23	0.50	0	6
Apache		+63.90%	+16.71%	1.00	1.18	1.30	1.0	0	6
Mail		+63.90%	+15.35%	1.00	1.18	1.29	1.0	0	6
DNS		+63.90%	+15.37%	1.00	1.18	1.29	10.0	0	6
Google	0.3	+39.86%	+23.74%	1.00	1.00	1.03	5.0	0	18
Apache		+41.48%	+14.40%	1.00	1.07	1.13	5.0	0	17
Mail		+41.49%	+12.89%	1.00	1.07	1.13	5.0	0	17
DNS		+39.86%	+10.05%	1.00	1.03	1.13	5.0	0	18
Google	0.6	+11.15%	+10.08%	1.00	1.10	1.16	5.0	0	32
Apache		+18.81%	+11.75%	1.00	1.10	1.16	5.0	0	32
Mail		+18.95%	+11.11%	1.00	1.10	1.16	5.0	0	32
DNS		+17.89%	+9.82%	1.02	1.29	1.40	5.0	0	32

TABLE IV: Energy Reduction for Dual τ in various workloads and dual delay timer values compared with A-I and A-I-S (opt τ: lowest energy). Job arrivals are modeled as Poisson Process, and Normalized Latencies (N.L.) are calculated with workload execution times as baselines.

		Dual-τ							
Workload	Utilization	Energy Reduction over A-I	Energy Reduction over A-I-S (opt τ)	+50%-ile N.L.	90%-ile N.L.	95%-ile N.L.	High τ	Low τ	Num. servers with High τ
Google	0.1	+62.33%	+23.95%	1.00	1.05	1.10	5.00	0.05000	2.00
Apache		+62.26%	+17.30%	1.00	1.14	1.24	5.00	0.20	2.00
Mail		+62.17%	+17.74%	1.00	1.18	1.31	5.00	0.20	2.00
DNS		+61.96%	+15.64%	1.00	1.11	1.19	5.00	0.50	2.00
Google	0.3	+38.92%	+21.94%	1.00	1.03	1.07	5.00	0.05	8.00
Apache		+40.18%	+13.00%	1.00	1.20	1.36	5.00	0.20	7.00
Mail		+39.84%	+13.78%	1.00	1.29	1.57	5.00	0.20	7.00
DNS		+40.36%	+12.64%	1.00	1.15	1.27	5.00	0.50	8.00
Google	0.6	+15.95%	+15.54%	1.00	1.01	1.05	5.00	0.05	18.00
Apache		+18.32%	+9.05%	1.00	1.12	1.18	5.00	0.20	18.00
Mail		+18.23%	+9.08%	1.00	1.15	1.24	5.00	0.20	18.00
DNS		+18.19%	+8.46%	1.00	1.18	1.28	5.00	0.50	18.00

TABLE V: Energy Reduction for Dual τ in various workloads and dual delay timer values compared with A-I and A-I-S (opt τ: lowest energy). Job arrivals are modeled as *MMPP*, and Normalized Latencies (N.L.) are calculated with workload execution times as baselines.

utilization, a server with τ_h is prioritized in order to favor the servers with τ_l to enter idle state state rapidly. If none of the servers in V_{ai} has an idle core, the incoming job is queued and a server in sleep state is woken up to enter active state. However, waking up a sleeping server every time when a job is queued can be suboptimal since the transition penalty for an asleep server is high. To address this issue, we add a simple threshold t_w. Only when the current number of pending jobs in queue is greater than t_w would a server in deep sleep be woken up. The t_w threshold is a tunable parameter that guides how conservatively the job handler wakes up a server. Throughout our experiments, t_w is set to be the product of the number of cores per server and the number of τ_h servers; This effectively avoids unnecessary server wake-ups. Note that our algorithm assumes that the servers are capable of processing one job per core at a time without loss of generality. If the core is capable of running multiple jobs concurrently due to techniques like Simultaneous Multi-Threading, we could assign multiple jobs per core until it is fully occupied.

B. Exploration of Dual Timers for Poisson Job Arrivals

The parameter exploration space for Dual τ is large due to combinational exploration of two τ values and the partitioning of servers into two τ categories. To reach the solution faster, we first conduct a uniform sampling of the three major parameters: high τ, low τ, and the number of servers with high τ. Due to space constraints, we are unable to present all of our results. We summarize a few important observations from our experiments below:

1) The number of high τ servers that maximizes energy savings is approximately ρ * total number of servers. This finding confirms our intuition that when using dual τ, the system is able to utilize a minimal number of active/standby servers while putting the majority of servers to deep sleep.

2) Having relatively large high τ values and setting low τ to zero (the server immediately goes to sleep state when idle) maximizes energy savings at all utilization levels relative to A-I and A-I-S ($\tau = c$) configurations.

Table IV summarizes the results of our experiments. We show the energy reduction with Dual Delay Timers compared to A-I and A-I-S along with the normalized percentile job latencies. We note that dual τ can achieve further energy savings of up to 16.7% beyond the A-I-S (optimal τ) especially at server utilization levels of 0.1.

C. Exploration of Dual Timers for MMPP Job Arrivals

We conduct experiments for *MMPP*-based workloads and utilize a Markov chain-based predictor for burstiness detection.

We perform parameter space exploration for τ values and number of servers with high τ. Table V summarizes the results of our experiments demonstrating the savings in energy over A-I and A-I-S (optimal τ: lowest energy) and the corresponding τ values and the number of servers with high τ. From the table, we can see that our Dual Delay Timer is able to save nearly 25% energy over optimal A-I-S configuration with single τ. Also, the number of active servers (with high τs) are now much smaller than the expected ρ * number of servers. This is because of short periods of high server utilization levels after which even the active servers can go

		Dual-τ							
Workload	Utilization	Energy Reduction over A-I	Energy Reduction over A-I-S (opt τ)	50%-ile N.L.	90%-ile N.L.	95%-ile N.L.	High τ	Low τ	Num. servers with High τ
Wikipedia	$\star$	+71.20%	+31.36%	1.00	1.37	1.53	5	0.002	2

TABLE VI: Energy reduction for Dual-τ in Wikipedia trace for dual delay timer values compared with A-I and A-I-S (opt τ: lowest energy). Normalized Latency (N.L.) is calculated with workload execution time as baseline.

to deep sleep without affecting performance. Unlike Poisson-based job arrivals, *MMPP* workloads have a small, non-zero low τ value to service the jobs during bursty phases.

D. Exploration of Dual Delay Timer for Wikipedia trace

We performed exploration of dual delay timer values τ for real world workload traces from Wikipedia. Table VI shows that setting low $\tau = 0.002$ seconds and high $\tau = 5$ seconds reduces system energy consumption by 31% over A-I-S (optimal τ) configuration. Note that the Wikipedia trace is a slowly varying workload with ultra low utilization (ranging from 5% to 10%). Detailed statistics in our experiments showed that under such scenario Dual Delay Timer is able to maintain the exact number of servers to be active without waking up even a single server prematurely after a short warm-up period. The results again show that Dual Delay Timer is especially effective under low system utilization levels.

V. Scalability with number of servers

In this section, we study the energy reduction benefits of Dual Delay Timer strategy by varying the number of servers. We adopt the same procedure for parameter exploration as we did in Section IV while exploring the optimal dual τ values that minimize energy consumption. We repeat the experiments for three different numbers of servers: 20, 50, and 100. Figure 3 shows energy savings of our Dual Delay Timer strategy compared with the corresponding A-I configuration. For each server farm size, the four synthetic workloads are simulated under three utilization levels of 0.1, 0.3, and 0.6. From the scalability trend, we can see that when the server farm size increases from 20 to 50, the relative energy savings for all workloads increase at the scale of 10% to 20%. With 100 servers, the energy saving benefits are slightly better in comparison to 50 servers, and the benefits are significantly higher against the corresponding A-I configuration. For example, at server utilization of 0.1, energy saving benefits of dual delay timer is about 50%, and at server utilization level of 0.3, the corresponding energy savings are well over 30% for all of the workloads. In summary, about 45% to 63% energy is saved at average server utilization 0.1 depending on the size of the server farm, 28% to 40% energy is saved at server utilization of 0.3, and 10% to 20% energy is saved at server utilization of 0.6. Our mechanism shows good scalability and energy saving potentials as server farm size increases.

VI. Related Work

Bridging the gap between data center server utilization and relatively high power/energy is a widely studied topic. Prior work [21], [22], [23] has used DVFS-based mechanisms; however, at low server utilization, static power dominates and DVFS is not effective. Also, due to device scaling, the headroom for voltage scaling is largely shrunk. As a result, prior work [14], [24] has proposed architectural support to

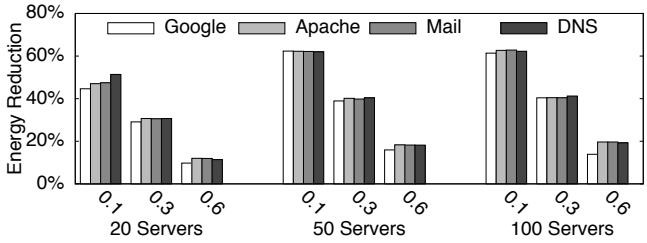

Fig. 3: Energy reduction of A-I-S Dual τ compared to A-I in various workloads and server utilization levels for different numbers of servers – 20, 50, and 100.

facilitate sleep state management on multi-core servers that include scheduling policies to delay, pre-empt and execute requests, and artificially create common idle and busy periods across cores of a server. Lo et al. [3] leverage *Running Average Power Limit* to dynamically adapt the runtime power of data center according to job latency feedback. The trends in server energy proportionality are analyzed by Ryckbosch et al. [25]. Sleepscale [5] utilizes speed scaling and server sleep states jointly to reduce the average power for single server systems while satisfying the QoS constraints of normalized request latency. We note that other approaches such as Knightshift [26] have explored more specialized approaches such as exploiting heterogeneity of processor cores to improve energy. In Knightshift, two execution modes are utilized – one providing high performance while consuming higher power, the other being an active low power mode for low-utilization periods to save power. The model is extended in [27] to provide cluster-wide energy proportionality. However, to preserve generality of our solution and study the applicability of our techniques on many current warehouse-scale systems, we model homogeneous servers and cores with same capability. We note that when we combine our proposed approach with energy improvement solution approaches on heterogeneous servers, we can further boost energy savings.

Gandhi et al. propose a delayed-off mechanism, AutoScale, in [13] that turns off a server after it is idle for a preset period of time. AutoScale reduces power consumption of the multi-server system by controlling the number of on servers while satisfying the response time SLA. In [20], the authors study the effectiveness of utilizing hypothetical sleep states with a power management policy, *SoftReactive*, which is similar to delayed off. In our work, the baseline approach A-I-S is a variant of the *SoftReactive* policy except that we use hardware-supported sleep states and realistic power profiles.

Other prior works have utilized job scheduling to deal with the interference between co-located workloads that might result in performance loss [28], [29]. Through accurately predicting the performance degradation between co-located workloads, the utilization of the cluster can be improved within the QoS constraints. Delimitrou et al. [30] use classification techniques to find the impact of server heterogeneity and interference between co-located workloads for resource assignment

while satisfying the performance requirements. A cluster management mechanism that maximizes resource utilizations while meeting the QoS constraints for each workload is presented in [31]. The authors of [32] design scheduling algorithms to minimize power consumption for bag-of-tasks applications with deadline constraints, while [33] studies the tradeoff between power management and performance in datacenters. The comparison in [33] indicates the need for a comprehensive sleep state control algorithm, which is what we have presented in this paper.

VII. Conclusion

In this paper we presented a novel Dual Delay Timer technique to judiciously use the processor sleep states to optimize server farm energy consumption. We evaluate the effectiveness of our approach using five different workloads including four synthetic (Google search, Apache, Mail, and DNS) and one real workload trace (Wikipedia). We study the workloads using two different job arrival patterns – Poisson for non-bursty arrivals and Modulated Markov Poisson Process for bursty arrivals. Our experimental results show that our techniques achieve up to 71% savings in energy over naive energy management without the use of low-power sleep states, and up to 31% energy savings over a relatively smarter energy management mechanism with just a single delay timer to enter the sleep state. We also show that the normalized latencies of jobs on a server farm with our Dual Delay Timer strategy are almost the same as the job latencies in the case when servers are always ready to accept incoming jobs.

Acknowledgment

This material is based upon work supported in part by the National Science Foundation under CAREER Award CCF-1149557 and CNS-1320226.

References

[1] R. Brown *et al.*, "Report to congress on server and data center energy efficiency: Public law 109-431," *Lawrence Berkeley National Laboratory*, 2008.

[2] J. Koomey, "Growth in data center electricity use 2005 to 2010," *A report by Analytical Press, completed at the request of The New York Times*, 2011.

[3] D. Lo, L. Cheng, R. Govindaraju, L. A. Barroso, and C. Kozyrakis, "Towards energy proportionality for large-scale latency-critical workloads," in *Proceeding of IEEE International Symposium on Computer Architecuture*, 2014.

[4] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *ACM SIGARCH Computer Architecture News*, vol. 35, no. 2, 2007, pp. 13–23.

[5] Y. Liu, S. C. Draper, and N. S. Kim, "Sleepscale: runtime joint speed scaling and sleep states management for power efficient data centers," in *Proceeding of IEEE International Symposium on Computer Architecuture*, 2014.

[6] G. Urdaneta, G. Pierre, and M. van Steen, "Wikipedia workload analysis for decentralized hosting," *Computer Networks*, vol. 53, no. 11, pp. 1830–1845, 2009.

[7] Hewlett-Packard, Intel, Microsoft, Phoenix and Toshiba. Advanced Configuration and Power Interface Specification. http://www.acpi.info/.

[8] L. Brown, "ACPI in linux," in *Linux Symposium*, 2005.

[9] T. Watanabe, "ACPI implementation on freebsd." in *USENIX Annual Technical Conference, FREENIX Track*, 2002.

[10] Intel. (2012) Intel Xeon processor E5-1600/E5-2600/E5-4600 product families. http://tinyurl.com/d7ma5nf.

[11] M. Floyd, M. Allen-Ware, K. Rajamani *et al.*, "Introducing the adaptive energy management features of the Power7 chip," in *IEEE Micro*, 2011.

[12] A. Gandhi and M. Harchol-Balter, "How data center size impacts the effectiveness of dynamic power management," in *Proceedings of IEEE Conference on Communication, Control, and Computing*, 2011.

[13] A. Gandhi, M. Harchol-Balter, R. Raghunathan *et al.*, "Autoscale: Dynamic, robust capacity management for multi-tier data centers," *ACM Transactions on Computer Systems*, vol. 30, no. 4, p. 14, 2012.

[14] D. Meisner and T. F. Wenisch, "Dreamweaver: architectural support for deep sleep," *ACM SIGPLAN Notices*, vol. 47, no. 4, pp. 313–324, 2012.

[15] R. N. Calheiros, R. Ranjan, A. Beloglazov *et al.*, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.

[16] D. Meisner, J. Wu, and T. F. Wenisch, "Bighouse: A simulation infrastructure for data center systems," in *Proceeding of IEEE International Symposium on Performance Analysis of Systems and Software*, 2012.

[17] P. Bodik, A. Fox, M. J. Franklin *et al.*, "Characterizing, modeling, and generating workload spikes for stateful services," in *Proceedings of ACM Symposium on Cloud Computing*, 2010.

[18] G. Casale, N. Mi, L. Cherkasova, and E. Smirni, "How to parameterize models with bursty workloads," *SIGMETRICS Perform. Eval. Rev.*, vol. 36, no. 2, Aug. 2008.

[19] D. Perez-Palacin, J. Merseguer, and R. Mirandola, "Analysis of bursty workload-aware self-adaptive systems," in *Proceedings of ACM/SPEC International Conference on Performance Engineering*, 2012.

[20] A. Gandhi, M. Harchol-Balter, M. Kozuch *et al.*, "Are sleep states effective in data centers?" in *Green Computing Conference*, 2012.

[21] S. Herbert and D. Marculescu, "Analysis of dynamic voltage/frequency scaling in chip-multiprocessors," in *Proceedings of ACM/IEEE International Symposium on Low Power Electronics and Design*, 2007.

[22] S. Kaxiras and M. Martonosi, "Computer architecture techniques for power-efficiency," *Synthesis Lectures on Computer Architecture*, vol. 3, no. 1, pp. 1–207, 2008.

[23] D. C. Snowdon, S. Ruocco, and G. Heiser, "Power management and dynamic voltage scaling: Myths and facts," 2005.

[24] D. Meisner, B. T. Gold, and T. F. Wenisch, "Powernap: eliminating server idle power," in *ACM Sigplan Notices*, vol. 44, no. 3, 2009.

[25] F. Ryckbosch, S. Polfliet, and L. Eeckhout, "Trends in server energy proportionality," *Computer*, vol. 44, no. 9, pp. 69–72, 2011.

[26] D. Wong and M. Annavaram, "Knightshift: Scaling the energy proportionality wall through server-level heterogeneity," in *Proceedings of IEEE/ACM International Symposium on Microarchitecture*, 2012.

[27] D. Wong and M. Annavaram, "Implications of high energy proportional servers on cluster-wide energy proportionality," in *Proceedings of IEEE International Symposium on High Performance Computer Architecture*, 2014.

[28] J. Mars, L. Tang, R. Hundt, K. Skadron, and M. L. Soffa, "Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations," in *Proceedings of IEEE/ACM International Symposium on Microarchitecture*, 2011.

[29] H. Yang, A. Breslow, J. Mars, and L. Tang, "Bubble-flux: Precise online qos management for increased utilization in warehouse scale computers," *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3, pp. 607–618, 2013.

[30] C. Delimitrou and C. Kozyrakis, "Paragon: QoS-aware scheduling for heterogeneous datacenters," *ACM SIGARCH Computer Architecture News*, vol. 41, no. 1, pp. 77–88, 2013.

[31] C. Delimitrou and C. Kozyrakis, "Quasar: Resource-efficient and qos-aware cluster management," *ACM SIGPLAN Notices*, vol. 49, no. 4, pp. 127–144, 2014.

[32] K. H. Kim, W. Y. Lee, K. Jong, and R. Buyya, "SLA-based scheduling of bag-of-tasks applications on power-aware cluster systems," *IEICE TRANSACTIONS on Information and Systems*, vol. 93, no. 12, pp. 3194–3201, 2010.

[33] S. Kanev, K. Hazelwood, G.-Y. Wei, and D. Brooks, "Tradeoffs between power management and tail latency in warehouse-scale applications," in *Proceedings of IEEE International Symposium on Workload Characterization*, 2014.