Received October 18, 2020, accepted October 28, 2020, date of publication November 4, 2020, date of current version November 17, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3035777

Co-Adjusting Voltage/Frequency State and Interrupt Rate for Improving Energy-Efficiency of Latency-Critical Applications

KI-DONG KANG, HYUNGWON PARK, GYEONGSEO PARK, AND DAEHOON KIM, (Member, IEEE)
Department of Information and Communication Engineering, DGIST, Daegu 42988, South Korea

Corresponding author: Daehoon Kim (dkim@dgist.ac.kr)

This work was supported in part by the National Research Foundation of Korea (NRF) grants funded by the Korean Government, Ministry of Science and ICT (MSIT) under Grant NRF-2020R1C1C1013315 and Grant NRF-2018R1A5A1060031, and in part by the Institute for Information and Communications Technology Promotion (IITP) funded by the Korean Government, Ministry of Science and ICT (MSIT), Resilient Cyber-Physical Systems Research, under Grant 2014-0-00065.

ABSTRACT As the power/energy consumption is one of the major contributors to the Total Cost of Ownership (TCO), improving power/energy efficiency is crucial for large-scale data centers where latency-critical applications are commonly accommodated while computing resources are usually under-utilized. For improving the power/energy efficiency of processors, most of the commercial processors support Dynamic Voltage and Frequency Scaling (DVFS) technology that enables to adjust Voltage and Frequency state (V/F state) of the processor dynamically. In particular, for the latency-critical applications, many prior studies propose power management policies using the DVFS for the latency-critical applications, which minimizes the performance degradation or satisfies the Service Level Objectives (SLOs) constraints. Meanwhile, although the interrupt rate also affects the response latency and energy efficiency of latency-critical applications considerably, those prior studies just introduce policies for V/F state adjustment while not considering the interrupt rate. Therefore, in this article, we investigate the impact of adjusting the interrupt rate on the tail response latency and energy consumption. Through our experimental results, we observe that adjusting interrupt rate along with V/F state management varies the performance and energy consumption considerably, and provides an opportunity to reduce energy further by showing latency overlap between different V/F states. Based on the observation, we show the quantitative potential in improving energy efficiency of co-adjusting V/F state and interrupt rate with a simple management policy, called `Co-PI`. `Co-PI` searches the most energy-efficient combination of the V/F state and interrupt rate from the latency and energy tables that we obtain through offline profiling, and reflect the combination to the core and NIC. `Co-PI` reduces energy consumption by 34.1% and 25.1% compared with performance and ondemand governors while showing the almost same tail response latency with the performance governor that operates cores at the highest V/F state statically.

INDEX TERMS Power management, dynamic voltage and frequency scaling, latency-critical applications, interrupt coalescing.

I. INTRODUCTION

In large-scale data centers, power/energy efficiency is essential since it mostly contributes to the Total Cost of Ownership (TCO). In particular, since current data centers do not always fully utilize their computing resources while showing a low average resource utilization (i.e., 10-50% on average [1]–[4]), there is an opportunity to improve the power/energy efficiency by turning off a part of hardware resources or throttling their performance. Consequently, sophisticated power management policies, while not degrading performance or satisfying the target Service Level Objectives (SLOs), become necessary [5].

For improving energy efficiency of the processor that is one of major contributors to the total power consumption of the computing system, most commercial processors support a power management technology, called Dynamic Voltage and Frequency Scaling (DVFS) [6], which can allow the

The associate editor coordinating the review of this manuscript and approving it for publication was Stavros Souravlas.

processor to adjust its Voltage and Frequency state (i.e., V/F state) dynamically at run-time. Besides, Operating Systems (OSes) also provide power management policies, called governors (e.g., `ondemand` governor [7]), which exploits the DVFS technology for improving energy efficiency. Although the DVFS technology can reduce power and energy consumption, it may increase response latency, especially tail response latency, of latency-critical application substantially [8], [9]. For example, the power management governors can determine a low V/F state that is not enough to handle the incoming requests quickly for the processor due to their periodic decision even though a quite number of requests arrives at the server for a very short period. To address the problem, many prior studies propose power management policies exploiting the DVFS technology for latency-critical applications to improve energy efficiency while minimizing performance degradation or not violating SLO constraints [10]–[14].

Meanwhile, in addition to the V/F state of the processor, interrupt rate also affects the response latency of the latency-critical applications since interrupt handling that includes handler invocation, supervisor mode switching, and more cache misses along with the packet delivery consumes the processor considerably as the network load increases [15]. Furthermore, the frequent interrupt delivery can even lead to the receive live lock problem that does not make any progress of tasks except for the interrupt handling [16]. Therefore, current Network Interface Cards (NICs) support interrupt coalescing technology [17] that can adjust the interrupt rate dynamically. However, adjusting the interrupt rate for latency-critical applications is challenging since it can affect response latency positively or negatively. Increasing interrupt rate can improve the response latency by delivering packets to the cores quickly, but it can degrade the latency by putting more load on the processor. On the contrary, decreasing interrupt rate can degrade the response latency by making packets wait longer at the network queues, but it can improve the latency by reducing the load on the processor due to interrupt handling. The current policy for the interrupt coalescing offered by the NIC drivers adjusts the interrupt rate based on the network load.

Since the interrupt rate shows the considerable impact on the response latency of latency-critical applications, adjusting the interrupt rate provides an opportunity to decrease the V/F state of cores without performance degradation, improving energy efficiency. For example, if the application still shows the same tail response latency after decreasing the V/F state of the processor by adjusting the interrupt rate, we can reduce the energy consumption without performance degradation. Furthermore, since the interrupt rate determines the load on the core, it also affects the energy consumption of the processor.

Therefore, in this article, we experimentally analyze the impact of the interrupt management at each V/F state of the processor on the response latency and energy consumption with two representative latency-critical applications, `memcached` [18] and `nginx` [19]. Our experimental results show that adjusting the interrupt rate makes the performance overlaps among V/F states, allowing changing the V/F state without performance degradation by adjusting the interrupt rate. Furthermore, we observe that adjusting the interrupt rate leads to increase/decrease in the energy consumption at the same V/F state. Consequently, adjusting the interrupt rate along with the V/F state management can improve the energy efficiency of the latency-critical applications considerably.

To demonstrate the potential of co-adjusting the V/F state and interrupt rate in improving the energy efficiency, we propose a simple policy, called `Co-PI`, which co-adjusts the V/F state and interrupt rate based on the network load. `Co-PI` first obtains latency and energy tables for each combination of the V/F state and interrupt rate through offline profiling; we discuss the offline profiling in Section IV in detail. Under the given load, `Co-PI` first searches combinations that show the same latency with the target latency that can be set by the user. Next, among the combinations, `Co-PI` chooses the combination that shows the lowest energy consumption. By choosing the most energy-efficient combination, `Co-PI` reduces the energy consumption by 34.1% and 25.1% compared with performance and ondemand governors, respectively, while showing the almost same or even shorter tail response latency than performance governor that operate cores at the highest V/F state statically.

To the best of our knowledge, this is the first study to demonstrate the potential of co-adjusting the V/F state and interrupt rate in improving energy efficiency for latency-critical applications. The main contributions of this article are as follows.

1) We demonstrate that the interrupt rate management affects the tail latency and energy consumption of the latency-critical applications.
2) We demonstrate that co-adjusting V/F state and interrupt rate provide an opportunity to improve energy efficiency further.
3) We demonstrate that the potential in improving the energy efficiency of co-adjusting the V/F state and interrupt rate with the proposed simple policy on the real-system setup.

The remainder of this article is organized as follows. Section II describes background. Section III analyzes the impact of adjusting the interrupt rate for each V/F state on the tail response latency and energy consumption of latency-critical applications. Section IV proposes a simple policy, `Co-PI`, which can show the potential of co-adjusting the V/F state and interrupt rate for power management. Section V shows the experimental results. Section VI and Section VII discusses related work and concludes this article, respectively.

II. BACKGROUND

A. INTERRUPT COALESCING FOR NETWORK I/O

I/O devices (e.g., Network Interface Card (NIC), storage, Graphic Processing Unit (GPU)) typically notify the processor of the completion of I/O events using interrupts.

Each interrupt from a different device has a unique ID along with its software handler, called Interrupt Service Routine (ISR). The interrupts temporarily stop the current execution of the processor, and the processor invokes the ISR corresponding to the number of the interrupt, which incurs switching to supervisor mode and more cache misses. However, as the advent of high-performance network technologies (e.g., 100 Gigabit Ethernet, InfiniBand [20]), the frequency of interrupt delivery to the processors increases substantially, which can even lead to receive live lock problem [16] where the processor cannot execute any tasks except for processing interrupts.

To mitigate the performance overheads by handling interrupts, current NICs support the interrupt coalescing technique that adjusts the frequency of interrupt delivery as the network load changes, reducing the load on the processor. However, although coalescing more interrupts mitigate the number of interrupts and performance overhead by handling the interrupts, it delays the packet delivery, increasing the response latency of network requests. For the interrupt coalescing, NICs support a register that maintains the period of interrupt delivery and the software driver that manages the value of the register. For example, the Intel 82599 NIC provides an Interrupt Throttle Register (ITR) and the software driver (i.e., ixgbe driver) that updates the ITR based on the number of processed network packets and size; the software driver's default ITR management adjusts the ITR value in the range of 40 (100K interrupts/sec.) to 336 (12K interrupts/sec.) [21]. In addition, the software driver also allows users to update the ITR value in the range of 24 (166K interrupt/s) to 4088 (1K interrupts/sec.).

The modern multi-queue NICs supporting multiple hardware queues for network packets manage the interrupt rate separately for each queue. Since each queue is typically mapped to a different core [22], packets and interrupts are distributed across cores with the multi-queue NICs, processing packets in parallel. For example, the Intel 82599 NIC supports up to 64 hardware queues, and activates the same number of queues as the number of cores. Consequently, the software driver coalesces interrupts separately depending on the load on each queue for each core.

B. POWER MANAGEMENT WITH DYNAMIC VOLTAGE AND FREQUENCY SCALING

For the power/energy efficiency of processors, most commercial processors support the Dynamic Voltage and Frequency Scaling (DVFS) technology. The DVFS technology allows the processor to dynamically adjust its current V/F state, varying performance and power consumption of the processor. Most of the desktop and mobile processors support the chip-wide DVFS that adjusts the V/F state of all cores on the same processor package to the same V/F state. Furthermore, current high-end multi-core processors for servers (e.g., Intel Xeon processor) support the different V/F state for each core (i.e., per-core DVFS).

Linux OS provides several software governors for power management based on the DVFS technology, such as performance, powersave, userspace, ondemand, conservative [23]. As static governors, performance and powersave governors statically operate cores at the highest and lowest V/F state, respectively. Consequently, performance governor shows the highest performance and power consumption while powersave governor shows the lowest performance and power consumption. The userspace governor allows users to statically set a particular V/F state for the processor. Meanwhile, dynamic governors, such as ondemand, conservative, adjust the V/F state based on the periodically measured CPU utilization (e.g., every 10 ms). The ondemand governor sets V/F state in proportion to the measured CPU utilization while the conservative governor gradually adjusts the V/F state by increasing/decreasing the V/F state adjacent to the current V/F state (e.g., from P1 to P0 or from P1 to P2). Since each core can deploy a different governor, cores of the processor supporting per-core DVFS can operate at the different V/F state. However, the chip-wide DVFS sets the V/F state of all cores to the highest V/F state among the V/F states determined by the governor of each core.

III. IMPACT OF CO-ADJUSTING V/F STATE AND INTERRUPT RATE ON RESPONSE LATENCY AND ENERGY

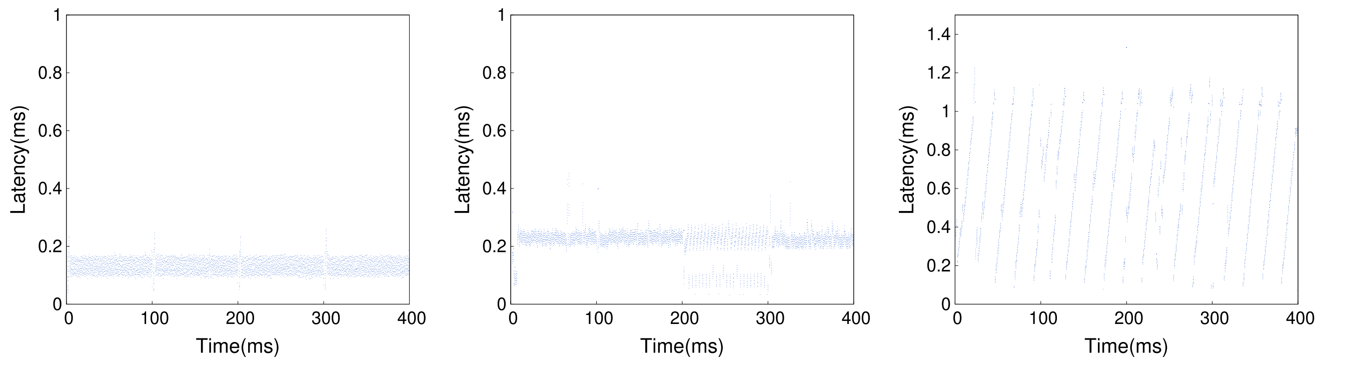
In this section, we investigate the effects of V/F states and interrupt rate on the tail response latency and energy consumption of the latency-critical application.

TABLE 1. System specification of experimental environment.

Processor	Intel Xeon Silver 4108
NIC	Intel 82599
Network Switch	D-Link DXS-1210-12SC 10 GbE
Network Driver	Intel ixgbe 5.6.1
OS	Ubuntu 16.04
Kernel	Linux 4.16.1

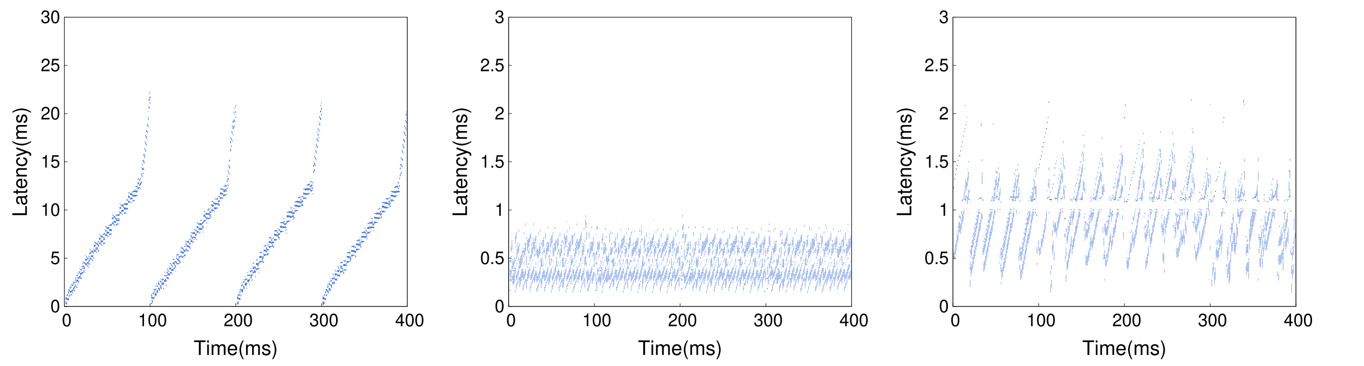
A. EXPERIMENTAL METHODOLOGY

We build a client-server environment consisting of two separated machines. Table 1 specifies the system specifications of the server. As a latency-critical application, we run `memcached`, which is a representative in-memory key-value store application. For the server, we run eight `memcached` threads on an eight-core Intel Xeon processor (Silver 4108) supporting 11 different V/F states ranging from P0 (1.8GHz) to P10 (0.8GHz) for each core (i.e., per-core DVFS). For performance evaluation, we measure tail response latency (e.g., 95th percentile latency (P95)) of `memcached` server, as most of the prior datacenter studies for latency-critical applications do. We use multi-queue NICs (Intel 82599) for both server and client, which distribute packets across all cores with Receive Side Scaling (RSS) technology, and connect the client and sever using D-Link DXS-1210-12SC 10 Gigabit Ethernet switch. We experiment with Linux 4.16.1 kernel of Ubuntu 16.04, and ixgbe 5.6.1 NIC driver. The client



(a) 166.6K interrupts/sec. (b) 5K interrupts/sec. (c) 1K interrupts/sec.

FIGURE 1. Response latency of each packet at low load (200KRPS).



(a) 166.6K interrupts/sec. (b) 5K interrupt/sec. (c) 1K interrupts/sec.

FIGURE 2. Response latency of each packet at high load (900KRPS).

generates and sends 200, 700, and 900 Kilo Requests Per Second (KRPS) to the server, denoted as low, medium, and high load, respectively. We disable hyper-threading and turbo-boost technology for both server and client, and use Model Specific Register (MSR) to obtain energy consumption by the processor package.

B. IMPACT OF INTERRUPT RATE ON THE RESPONSE LATENCY

We first analyze the impact of interrupt rate on the response latency of the latency critical service. Figure 1 and Figure 2 plot the response latency of every packet for 400ms at the low and high loads, respectively, To eliminate the impact of the V/F state, we set the V/F state of all cores to the highest state (i.e., P0). Figure 1a, Figure 1b, and Figure 1c show the response latency of the requests at low load when the Interrupt Throttle Rate (ITR) is 24 (166.6K interrupts/sec.), 800 (5K interrupts/sec.), and 4088 (1K interrupts/sec.), respectively; note that 24 and 4088 are the minimum and maximum values supported by the NIC. At the low load, the 24 ITR shows the lowest response latency on average. However, as the ITR increases (i.e., as the interrupt rate decreases), the response latency increases since packets wait for the delivery to the core at the network queues longer.

However, at the high load, increasing ITR shows completely different results. As plotted in Figure 2a, the low ITR increases the response latency substantially. This is because the frequent interrupt delivery interferes with request processing of application threads running on the core. As the interrupt handlers consume the CPU, the application threads consume the CPU less, affecting performance considerably, especially at the high load. Therefore, if we allow application threads to consume more CPUs by reducing the interrupt rate, the response latency is considerably improved at the high load as plotted in Figure 2b and Figure 2c. However, the excessive increase of the ITR rather increases the response latency. Compared with 4088 ITR, 800 ITR shows the lower response latency on average. Therefore, properly adjusting the ITR can be critical for providing low response latency for latency-critical applications.

C. PERFORMANCE AND ENERGY ANALYSIS OF THE V/F STATE AND INTERRUPT RATE

To investigate the impact of the V/F state and interrupt rate on the tail latency and energy consumption of the latency-critical application, we measure the P95 and energy consumption as the combination of V/F state and interrupt throttle rate (ITR) changes.

Tail Response Latency: Figure 3 shows the range of P95 as the ITR changes by plotting the lowest and highest latency for each V/F state (from P0 to P10) at three different loads, low, medium, and high. For the experiments, we gradually

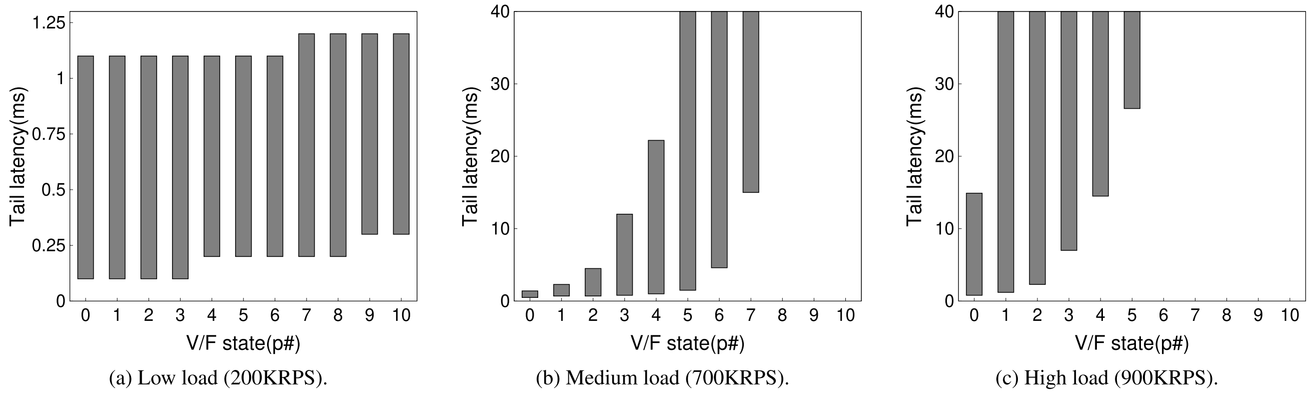


FIGURE 3. The range of 95th percentile latency with `memcached` **by adjusting ITR for each V/F state.**

increase the ITR value by 100 from the minimum to maximum (i.e., 24 to 4088) for each V/F state. We do not plot when the server fails to show the throughput as much as the requests per second generated by the client since the tail latency will increase substantially up to several seconds in that case; we represent response latency of the cases as 40ms in figures.

As plotted, adjusting ITR changes the tail response latency of the latency-critical application considerably. At the low load, the ITR adjustment varies P95 by about 1ms even at the same V/F state. In addition, at the medium and high loads, the ITR can substantially increase/decrease the P95 by up to several seconds. For example, when the cores operate at P4 at the medium load, the minimum P95 by the ITR adjustment is 1ms while the maximum P95 is 22.2ms.

In many cases, adjusting the ITR shows the greater impact on the tail latency than adjusting the V/F state. In particular, the impact of adjusting ITR becomes greater as the V/F state decreases at the medium load. This is because the lower V/F states reduce the performance of cores, increasing the CPU utilization at the same load, thus the interrupt rate affects the response latency more by stealing CPU share of application threads. In addition, at P5, P6, and P7, the ITR adjustment can lead to lower throughput than the generated requests per second from the client, increasing the P95 up to several seconds; at P8, P9, and P10, all ITR configurations show the less throughput than the generated requests per second from the client. At the high load where the core is highly utilized by application threads, except for P0, the improper ITR adjustment can increase the tail latency by up to several seconds.

Since the ITR adjustment shows a considerable impact on the P95, it can allow the same performance at the lower V/F state, providing an opportunity to improve energy efficiency. As plotted in Figure 3, the ranges of P95 are overlapped among all V/F states at the low load. This means that P10 can show the similar or the same P95 with P0 by the ITR adjustment. At the medium and high loads, there is also the latency overlap between different V/F states. Consequently, Figure 3 shows that there are many combinations of the V/F states and ITR values that show the same tail latency.

Energy: Figure 4 shows the ranges of the energy consumption by the ITR adjustment at each V/F state for three different loads, low, medium, and high. All results are normalized to the worst energy consumption at each load. As expected, increasing/decreasing the V/F state increase/decrease the energy consumption. The energy consumption varies among the V/F states by up to 23.5%. In particular, when decreasing the V/F state from P0, we can considerably reduce energy consumption. Since different V/F states can show the same performance by the ITR adjustment, we can improve energy efficiency by decreasing V/F state along with the ITR adjustment while not degrading performance.

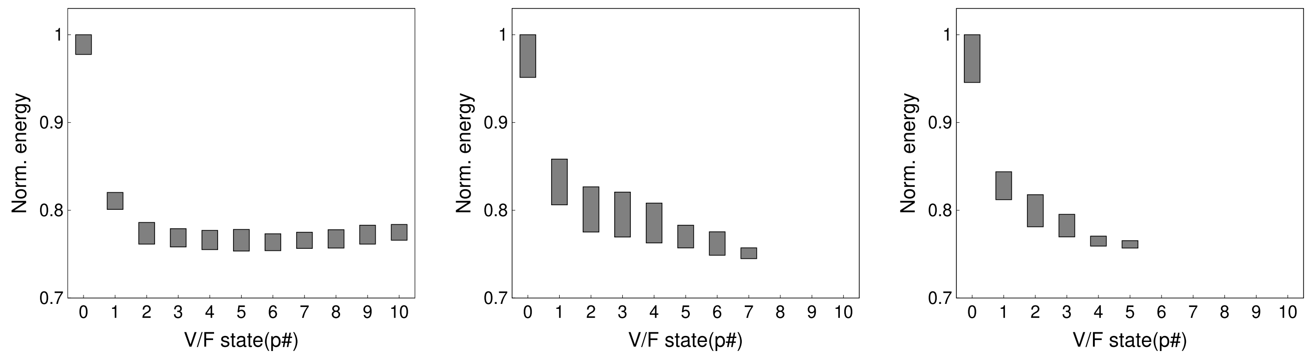
In addition to the V/F state, adjusting ITR also shows the different energy consumption by up to 5.8% (P0 at the high load) at the same V/F state. This is because the cores are more/less consumed as the interrupt rate changes. As shown in Figure 4a, at the low load, between P2 and P10, the ITR adjustment leads to greater change in the energy consumption than the V/F state adjustment. Therefore, adjusting ITR along with the V/F state can improve energy efficiency further.

Our experimental results show that co-adjustment of the V/F state and ITR provide an opportunity to improve energy efficiency further. In this article, we quantitatively demonstrate the potentials of the co-adjustment of the V/F state and ITR for improving the energy efficiency of latency-critical applications.

IV. CO-ADJUSTING V/F STATE AND INTERRUPT THROTTLE RATE FOR IMPROVING ENERGY EFFICIENCY

A. DYNAMIC CO-ADJUSTMENT OF V/F STATE AND INTERRUPT THROTTLE RATE

In Section III, we observe that the ITR adjustment along with V/F state management can improve energy efficiency considerably. To observe the quantitative potential of co-adjusting the V/F state and ITR in improving energy efficiency, we implement a simple power management policy, called `Co-PI`. `Co-PI` co-adjusts the V/F state and ITR for latency-critical applications as the measured network load rather than just adjusting the V/F state as prior power management studies. For each load, `Co-PI` determines the most energy-efficient combination of the V/F state and ITR while showing the same tail response latency with the target latency that can be specified by users.



(a) Low load (200KRPS). (b) Medium load (700KRPS). (c) High load (900KRPS).

FIGURE 4. **The range of energy consumption with** `memcached` **adjusting ITR for each V/F state.**

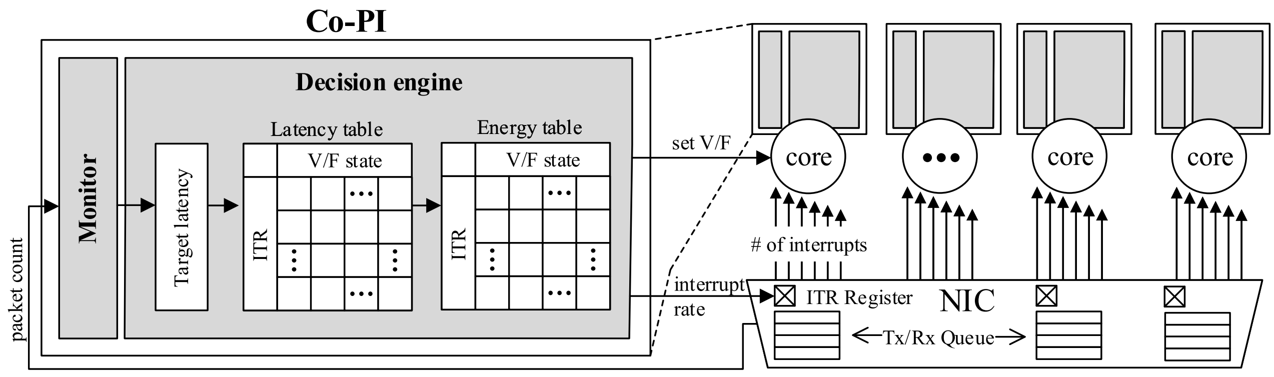


FIGURE 5. **The overall architecture of** `Co-PI`**.**

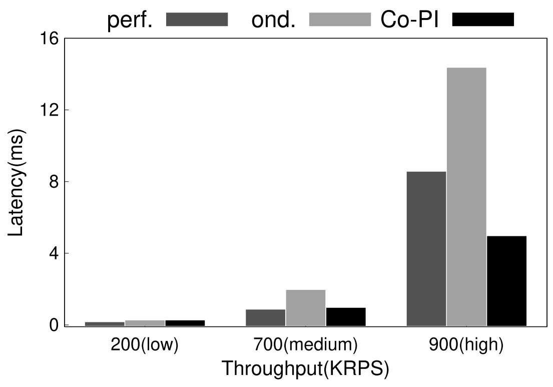
Figure 5 illustrates the overall architecture of `Co-PI`. `Co-PI` consists of two major components, the monitor and decision engine. The monitor counts the number of packets processed by each core to figure out the current network load; note that the network load on each core can be different with the multi-queue NIC since the multi-queue NIC maps each queue to a different core, and distributes packets across queues by default. The monitor delivers the counted value to the decision engine.

The decision engine uses the target tail response latency when searching combinations of the V/F state and ITR. To set the target latency aggressively, although we use P95 when the core operates at P0 statically (i.e., the same with `performance` governor) with the default ITR policy offered by the Intel NIC driver, the target tail latency can be set to a particular value by the user. The decision engine has two tables for latency and energy that maintain P95 and energy consumption of combinations of V/F state and ITR; the tables need to be filled through offline-profiling as discussed in Section IV-B. `Co-PI` first searches possible combinations of V/F state and ITR that show the same latency with the target latency under the given load by looking up the latency table. Next, `Co-PI` chooses the most energy-efficient combination that shows the lowest energy consumption among the combinations searched by the first step; with the multi-queue NIC and multi-core processor supporting per-core DVFS, `Co-PI` chooses the most energy-efficient combination of V/F state and ITR for each core. Lastly, `Co-PI` reflects the V/F state and ITR of the chosen combination to the core and NIC queue.

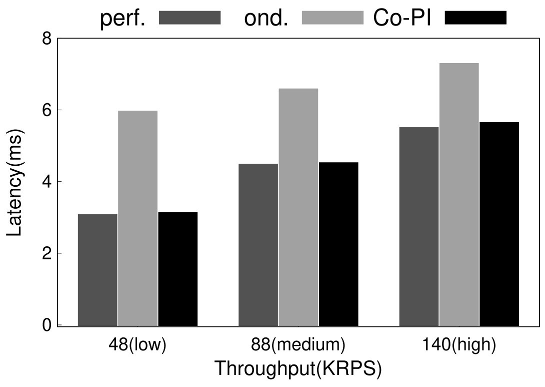
B. OBTAINING THE ENERGY-EFFICIENT COMBINATION OF V/F STATE AND ITR THROUGH OFFLINE PROFILING

For offline profiling, there are tremendous number of cases since we have three dimensions, the V/F states, ITR, and loads. The number of V/F states supported by the processors is typically ranging from 10 to 20, but there are thousands of ITR configurations; in our experimental environments, the processor supports 11 V/F states, and the NIC supports 4065 different ITR configurations ranging from 24 to 4088. Consequently, there are 44968 (11×4088) cases for each load, making the profiling costly. However, if the target tail response latency is millisecond scale, we do not need to profile all cases by gradually increasing the ITR by 1 (e.g., 24, 25, 26,...) since increasing/decreasing the ITR value by 1 leads to the difference of the interrupt period in microsecond scale. If we increase the ITR by 4, it delays the interrupt delivery to the processor by $1\mu s$. Consequently, the fine-grained profiling is not necessary for the applications that target millisecond scale latency; in this article, we use `memcached` and `nginx` that target millisecond scale tail response latency.

Furthermore, we observe that the ITR change shows the much greater impact on the response latency when the ITR is low (i.e., delivering interrupts frequently). This is because



(a) 95th percentile response latency with `memcached`.



(b) 95th percentile response latency with `nginx`.

FIGURE 6. Comparison of 95th percentile response latency among `performance`**,** `ondemand`**, and** `Co-PI`**.**

that a considerable portion of CPU time is consumed by handling interrupts when the ITR is low, thus increasing ITR (i.e., reducing interrupt rate) shows a notable impact on the performance. However, as the ITR value increases, the performance difference between configurations decreases gradually. Therefore, we do not need to profile in the fine-grained manner in the high ITR range. Lastly, due to the run-time characteristics of latency critical applications where the load usually fluctuates, the fine-grained profiling for each load that can rather lead to inefficient V/F state and ITR decisions is not necessary.

V. EVALUATION

A. EXPERIMENTAL METHODOLOGY

We use the same experimental environment with Section III-A. In addition to `memcached` (in-memory key value store), we run `nginx`, which is a representative web server application. To evaluate `Co-PI`, we use the P95 as the target latency when cores operate at the P0 (i.e., the highest V/F state) statically with the default ITR management policy offered by the Intel NIC driver. The monitor of `Co-PI` measures the network load every 100ms, thus the decision engine of `Co-PI` determines the combination of the V/F state and ITR every 100ms. We use `ondemand` and `performance` governors to compare the tail response latency and energy consumption with `Co-PI`. With our setup, `ondemand` governor determines the V/F state every 10ms based on the CPU utilization for the past 10ms; the minimum sampling period of `ondemand` governor is 10ms.

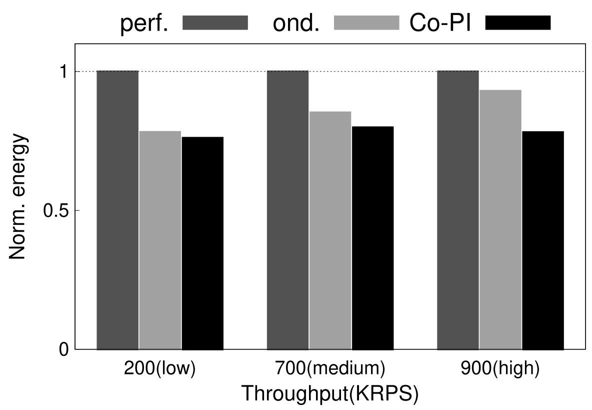
B. EXPERIMENTAL RESULT

1) COMPARISON OF THE TAIL RESPONSE LATENCY AND ENERGY CONSUMPTION

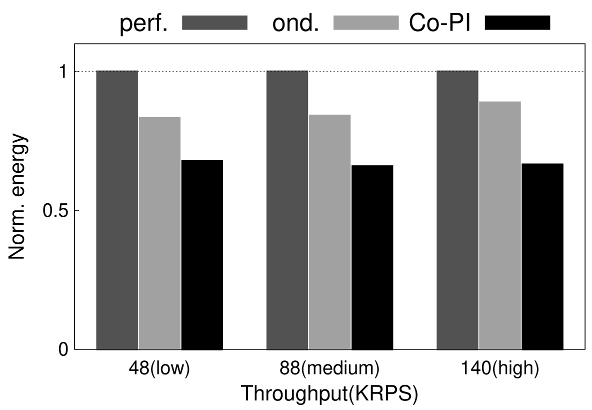
Figure 6 shows the comparison of P95 among `performance` governor, `ondemand` governor, and `Co-PI`. Figure 6a and Figure 6b plot the P95 with `memcached` and `nginx` applications, respectively. The `ondemand` governor shows the longest P95 for all loads while `performance` governor shows the much shorter latency compared with `ondemand` governor. With `nginx`, `Co-PI` shows the P95 almost same with `performance` governor for all loads. With `memcached`, at the low and medium loads, `Co-PI` shows the P95 almost same with `performance` governor while `Co-PI` shows even shorter P95 by 42.3% compared with `performance` governor by co-adjusting the V/F state and ITR at the high load. The reason for showing much shorter latency even than `performance` governor at the high load is that `Co-PI` can show higher performance by ITR adjustment even though it chooses lower V/F states than P0 while the default ITR driver with `performance` governor usually fails to choose the efficient ITR; note that improper ITR leads to worse performance even with higher V/F states as discussed in Section III-C. In particular, `memcached` requires more CPU-shares for processing interrupt due to small size data than `nginx`. Consequently, the impact of increasing ITR (i.e., reducing interrupt rate) that allows application threads to consume more CPU-share on the tail response latency is considerable, especially at the high load.

Figure 7 shows the comparison of the energy consumption among `performance` governor, `ondemand` governor, and `Co-PI`. All results are normalized to the energy consumption of `performance` governor. Figure 7a and Figure 7b show the energy consumption with `memcached` and `nginx`, respectively. As plotted, the `performance` governor shows the highest energy consumption while the `ondemand` governor consumes much less energy than the `performance` governor. For all cases, `Co-PI` shows the lowest energy consumption by co-adjusting the V/F state and ITR. With `nginx`, `Co-PI` shows further energy reduction than with `memcached`. With `memcached`, compared with `performance` governor, `Co-PI` reduces the energy consumption by 23.9%, 20.1%, and 21.9% for the low, medium, and high loads, respectively. With `nginx`, compared with `performance` governor, `Co-PI` reduces the energy consumption by 32.3%, 34.14%, and 33.47% for the low, medium, and high loads, respectively.

In particular, as the load increases, `Co-PI` shows further energy reduction compared with the `ondemand` governor. This is because the `ondemand` governor increases the V/F

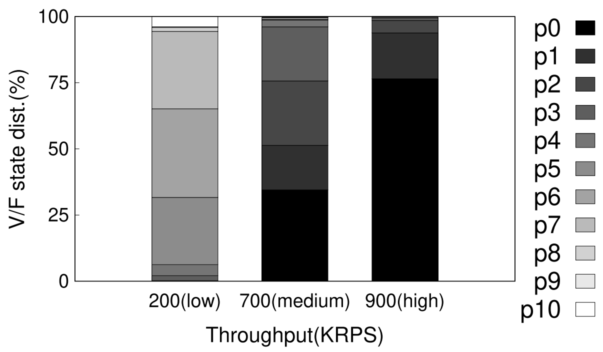


(a) Energy consumption with memcached.

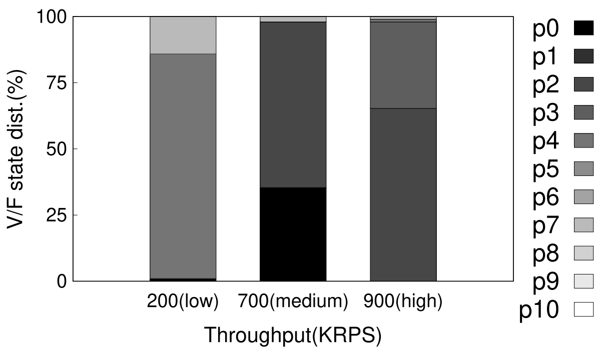


(b) Energy consumption with nginx.

FIGURE 7. Comparison of the energy consumption among performance**,** ondemand**, and** Co-PI**.**



(a) V/F state distribution by ondemand governor.



(b) V/F state distribution by Co-PI.

FIGURE 8. Comparison of V/F state distribution with memcached**.**

state as the load increases while Co-PI increases the ITR rather than the V/F state. With memcached, compared with ondemand governor, Co-PI reduces the energy consumption by 2.7%, 6.3%, and 16% for the low, medium, and high loads, respectively. With nginx, compared with ondemand governor, Co-PI reduces the energy consumption by 18.63%, 21.7%, and 25.12% for the low, medium, and high loads, respectively.

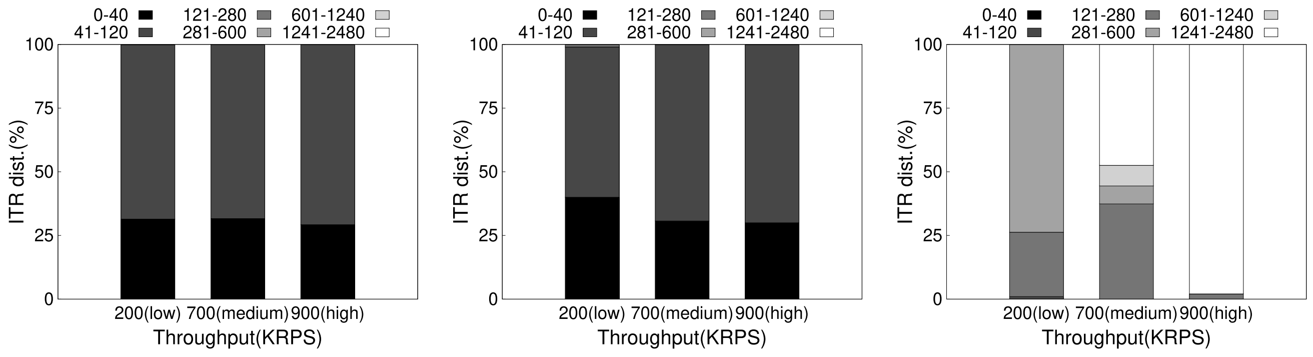
In summary, as a result of co-adjusting the V/F state and ITR for latency-critical applications, we show a considerable energy reduction by up to 23.9% and 34.1% compared with the performance governor while showing the energy reduction by up to 16% and 25.1% compared with ondemand governor, respectively. Along with the energy reduction, Co-PI shows almost the same or even shorter P95 compared with the performance governor.

2) COMPARISON OF THE V/F STATE AND ITR DISTRIBUTION

To figure out where the performance and energy difference come from, we plot the V/F state and ITR distribution of the performance governor, ondemand governor, and Co-PI. Figure 8 shows comparison of the V/F state distribution between the ondemand governor and Co-PI; note that the performance governor operates cores at the highest V/F state (i.e., P0) statically. Figure 8a and Figure 8b show the V/F state distribution of the ondemand governor and Co-PI, respectively. Figure 9 shows a comparison of the ITR distribution between the performance governor, ondemand governor, and Co-PI. Figure 9a, Figure 9b, and Figure 9c show the ITR distribution of the performance governor, ondemand governor, and Co-PI, respectively.

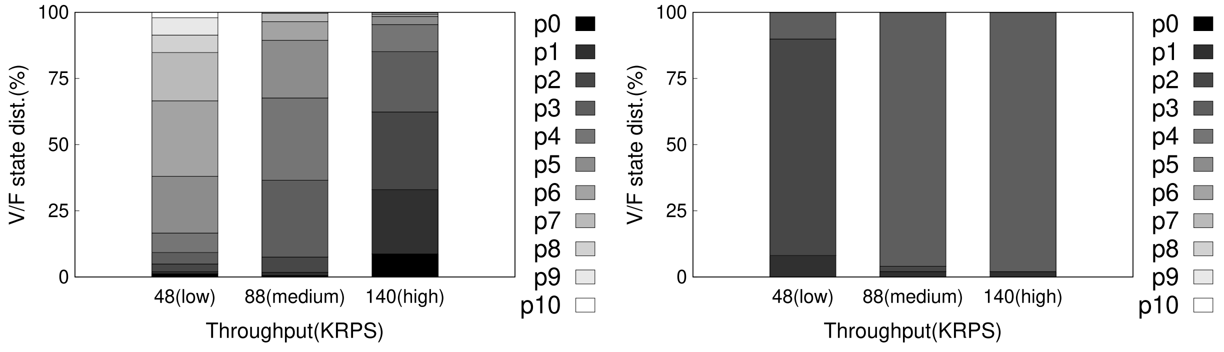
As shown in Figure 8a, as the load increases, the ondemand governor operates cores more at the higher V/F states since it determines the V/F state based on the CPU utilization. At the high load, the ondemand governor operates cores at the P0 for 76.5% of the total run-time. On the other hand, as shown in Figure 8b, Co-PI operates core at lower V/F states than the ondemand governor on average even at the high load since it does not determine the V/F state based on the CPU load.

However, with memcached, when the load is 200 KRPS, although the average V/F state by Co-PI (i.e., P4) is higher than the average V/F state by the ondemand governor (i.e., P6), Co-PI shows the energy reduction by 2.7%. This means that adjusting ITR affects the energy consumption more than the V/F state. As plotted in Figure 9b and 9c, Co-PI reduces the interrupt rate by setting higher ITR values than the ondemand governor, reducing the load on the cores (i.e., reducing the performance overheads by handling interrupts). As shown in Figure 8a and 8b, when the load is 700KRPS,



(a) ITR distribution with `performance` governor. (b) ITR distribution with `ondemand` governor. (c) ITR distribution by `Co-PI`.

FIGURE 9. **Comparison of ITR distribution with** `memcached`**.**



(a) V/F state distribution by `ondemand` governor. (b) V/F state distribution by `Co-PI`.

FIGURE 10. **Comparison of V/F state distribution with** `nginx`**.**

`Co-PI` and `ondemand` governor show a similar V/F state distribution while `Co-PI` shows less energy consumption by 6.3%. This is also because `Co-PI` sets higher ITRs than the `ondemand` governor as shown in Figure 9c.

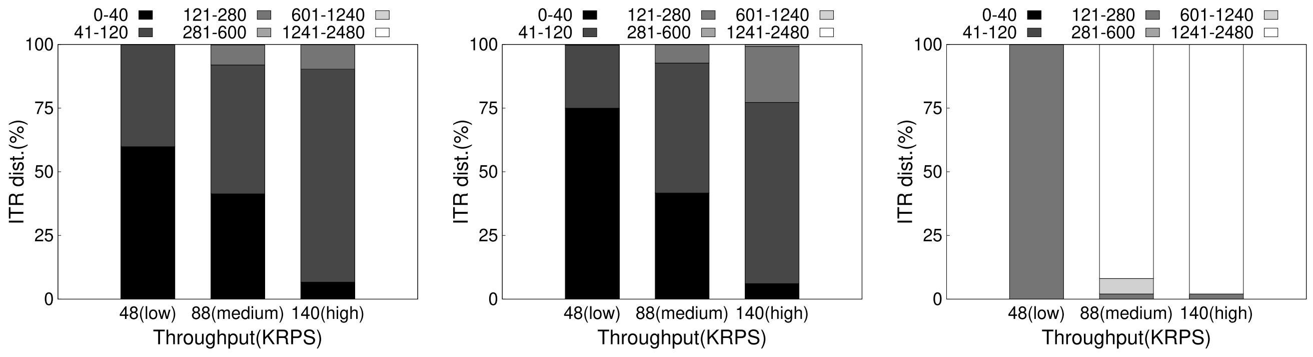
Lastly, as shown in Figure 8a and 8b, when the load is 900KRPS, `Co-PI` operates cores at P2 for 65.3% of the runtime while the `ondemand` governor mostly operate cores at P0 (i.e., for 76.5% of the runtime). Although `Co-PI` operates cores at the lower V/F states than the `ondemand` governor, it does not lead to performance degradation by increasing the ITR (i.e., reducing interrupt rate) as plotted in Figure 9a and Figure 9b; note that the Intel default ITR management adjusts the ITR in the narrow range as plotted in Figure 9a and Figure 9b. Consequently, `Co-PI` reduces the energy consumption by 16% compared with the `ondemand` governor while also improving tail response latency.

Figure 10 and Figure 11 show comparison of V/F state and the ITR distribution with `nginx`, respectively. Figure 10a and Figure 10b show the V/F state distribution of the `ondemand` governor and `Co-PI`, respectively. Figure 11a, Figure 11b, and Figure 11c show the ITR distributions of the `performance` governor, `ondemand` governor, and `Co-PI`, respectively.

As shown in Figure 10a, with `nginx`, as the load increases, the `ondemand` governor also operates cores more at the high V/F state as it does with `memcached`; the residence time at P0 decreases considerably compared with the `memcached`s results since `ngix` shows the relatively lower CPU utilization with large size data. On the other hand, as shown in Figure 10b, `Co-PI` does not notably show the increase of average V/F state as the load increases. When the load is 48KRPS with `nginx`, `Co-PI` mostly operates cores at P2 while the `ondemand` governor mostly operates cores at the lower V/F states (i.e., P5, P6, P7). However, `Co-PI` shows less energy consumption by 18.63% compared with the `ondemand` governor; note that `Co-PI` reduces the energy consumption by 16% with `memecached` compared with the `ondemand` governor. With `nginx`, `Co-PI` shows more energy reduction since adjusting ITR leads to the energy reduction while handling requests faster with the higher V/F states, making cores enter the idle state quickly; `ngix` shows more idle periods while running than `memcached`.

Lastly, as shown in Figure 10b and Figure 11c, when the load is 88KRPS and 140KRPS, `Co-PI` shows similar V/F state and ITR distributions while the `ondemand` governor operate cores at the higher V/F states and ITRs as the load increases. At the 88KRPS and 140KRPS, `Co-PI` reduces the energy consumption by 21.7% and 25.1% compared with the `ondemand` governor, respectively.

In summary, with `memcached` shows the short service time per request with the small size data, the impact of



(a) ITR distribution with `performance` governor. (b) ITR distribution with `ondemand` governor. (c) ITR distribution by `Co-PI`.

FIGURE 11. Comparison of ITR distribution with `nginx`.

adjusting ITR is greater than `nginx`. Consequently, along with the energy reduction by up to 23.9%, `Co-PI` also improves the tail response latency by 42.3% compared with the `performance` governor at the high load. Compared with the `ondemand` governor, `Co-PI` improves tail response latency by up to 65.7% while improving energy consumption by up to 16%. With `nginx`, `Co-PI` shows almost the same tail response latency with the `performance` governor while reducing the energy consumption by up to 34%. Compared to the `ondemand` governor, `Co-PI` improves energy consumption and tail response latency by up to 25.1% and 22.7%, respectively.

VI. RELATED WORK

A. POWER MANAGEMENT

Many existing studies propose power management policies for V/F states or idle states (i.e., C states) to reduce energy consumption while ensuring the target latency of latency-critical applications.

V/F State Management for Latency-Critical Applications: `Rubik` improves the energy efficiency by setting the optimal V/F based on the length of the queuing of network packets so that requests for latency-critical applications do not show longer latency that the target latency [10]. `Rubik` uses the table that maintains the optimal V/F for each queue length, which is pre-trained based on the statistical model. `PEGASUS` adjusts the power supply when the target latency is violated or the latency is much lower than the target while monitoring the power of servers running latency-critical applications and the latency of requests [11]. `Adrenaline` identifies latency-critical requests among requests and adjusts the V/F state according to the load [12]. `Hipster` determines a core to schedule latency critical service and the V/F state of the core in the heterogeneous architecture through reinforcement learning using the latency of the request [13]. `NMAP` adjusts the V/F state for latency-critical applications based on the transitions of packet processing modes between interrupt and polling [14]. `NMAP` raises the V/F state when it detects the ratio of polling to interrupt increases rapidly. Kumar *et al.* [24] propose a technique to adjust V/F state according to the queueing of requests and analyzes the performance impact of the queue-core mapping configuration.

C-State Management for Latency-Critical Applications: `PowerNap` proposes an approach to enter a deep C-state quickly by handling requests at the highest V/F [25]. `CARB` improves energy efficiency by controlling the number of active cores for latency-critical applications based on the load so that in-active cores enter the C-state [26]. `DynSleep` predicts the maximum idle time that does not violate the target SLO based on request arrival time, and decide the C-state according to the predicted idle time [27]. `Yawn` also predicts the maximum idle time according to predictor based on machine learning without the target SLO violation, and update machine learning model parameters with the difference between predicted idle time and measured idle time [28].

V/F State and C-State Management for Latency-Critical Applications: `SleepScale` profiles the idle time of the processor every epoch (e.g., 1 minute) and predicts the idle time to determine the optimal V/F and processor idle state that does not violate target latency [29]. `NCAP` wakes up the cores from the idle state, maximizes the V/F, and disable the idle state before the packets are delivered to the core when it detects the burst of latency-critical requests at the NIC [30]. `uDPM` manipulates packet delivery rate at the SmartNIC to maximize the residence time at the idle state while setting energy-optimized V/F when the packets are delivered [31].

However, all of these power management studies for the latency-critical applications overlook the impact of adjusting interrupt rate even though it affects the tail response latency and energy consumption considerably while providing an opportunity to reduce energy by allowing cores to decrease the V/F without performance degradation. By co-adjusting V/F state and interrupt rate, `Co-PI` shows the considerable energy reduction of latency-critical applications while showing the same P95 with performance governor.

VII. CONCLUSION

In this article, we show that the impact of adjusting interrupt rate on performance and energy consumption of latency-critical application. Through the experimental analysis, along with the V/F state management, we demonstrate that the adjustment of the interrupt rate provides an opportunity to improve energy efficiency considerably without performance degradation. This is because the the adjustment of the interrupt rate allows cores to decrease the V/F state without performance degradation. Furthermore, we observe that adjusting the interrupt rate shows more impact on the tail response latency than adjusting the V/F state in many cases. Consequently, we demonstrate that co-adjusting the V/F state and interrupt rate lead to substantial improvement in the energy efficiency of the latency-critical applications. Based on the observation, we propose `Co-PI` to observe the quantitative potential of co-adjusting the V/F state and interrupt rate in improving energy efficiency, which co-adjusts the V/F state and interrupt load based on the measured load. Under given load, `Co-PI` searches the most energy-optimal combination of the V/F state and the interrupt rate among the ones that does not show the longer latency than the specified target latency. In this article, we use the aggressive target latency that is the P95 with `performance` governor with the default ITR management offered by the Intel NIC. Our experimental results show that `Co-PI` reduces the energy consumption by up to 23.9% and 34.1%, respectively, with `memcached` and `nginx`, while not showing the notable degradation or even shorter latency compared with the aggressive target latency.

In this article, although we show the quantitative potential of co-adjusting the V/F state and interrupt rate in improving energy efficiency, our proposed `Co-PI` requires offline profiling and operates with a simple policy based on the network load. Therefore, for more efficient and dynamic management, we plan to extend our work to co-adjust the V/F state and interrupt rate based on other architectural behaviors related to the tail response latency and energy consumption in addition to the network load. To propose more practical solution, we also plan to improve the efficiency of offline training by avoiding the unnecessary profiling. Furthermore, the idle state of the processor is also another type of power states that allows core to reduce the energy consumption when the core are idle. In the future work, we also plan to analyze the interrupt rate on the idle state and the energy efficiency, and propose management techniques that can maximize the energy reduction without performance degradation. Lastly, we plan to integrate with policies for the idle state management with our work.

ACKNOWLEDGMENT

(Ki-Dong Kang and Hyungwon Park contributed equally to this work.)

REFERENCES

[1] C. Delimitrou and C. Kozyrakis, "Quasar: Resource-efficient and qos-aware cluster management," in *Proc. ACM Int. Conf. Archit. Support Program. Lang. Oper. Syst. (ASPLOS)*, 2014, pp. 127–144.

[2] L. A. Barroso and U. Hölzle, "The case for energy-proportional computing," *Computer*, vol. 40, no. 12, pp. 33–37, Dec. 2007.

[3] P. Bohrer, E. N. Elnozahy, T. Keller, M. Kistler, C. Lefurgy, C. McDowell, and R. Rajamony, "The case for power management in Web servers," in *Proc. Power Aware Comput.*, 2002, pp. 261–289.

[4] L. A. Barroso and U. Hölzle, "The datacenter as a computer: An introduction to the design of warehouse-scale machines," *Synth. Lectures Comput. Archit.*, vol. 4, no. 1, p. 108, Jan. 2009.

[5] J. Dean and L. A. Barroso, "The tail at scale," *Commun. ACM*, vol. 56, no. 2, pp. 74–80, Feb. 2013.

[6] P. Macken, M. Degrauwe, M. Van Paemel, and H. Oguey, "A voltage reduction technique for digital systems," in *Proc. 37th IEEE Int. Conf. Solid-State Circuits*, 1990, pp. 238–239.

[7] D. Brodowski and N. Golde. (2013). *Linux Cpufreq Governors*. [Online]. Available: https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt

[8] D. Meisner, C. M. Sadler, L. A. Barroso, W.-D. Weber, and T. F. Wenisch, "Power management of online data-intensive services," in *Proc. 38th Annu. Int. Symp. Comput. Archit.*, 2011, pp. 319–330.

[9] S. Kanev, K. Hazelwood, G.-Y. Wei, and D. Brooks, "Tradeoffs between power management and tail latency in warehouse-scale applications," in *Proc. IEEE Int. Symp. Workload Characterization (IISWC)*, Oct. 2014, pp. 31–40.

[10] H. Kasture, D. B. Bartolini, N. Beckmann, and D. Sanchez, "Rubik: Fast analytical power management for latency-critical systems," in *Proc. 48th Int. Symp. Microarchitecture*, 2015, pp. 598–610.

[11] D. Lo, L. Cheng, R. Govindaraju, L. A. Barroso, and C. Kozyrakis, "Towards energy proportionality for large-scale latency-critical workloads," in *Proc. ACM/IEEE 41st Int. Symp. Comput. Archit. (ISCA)*, Jun. 2014, pp. 301–312.

[12] C.-H. Hsu, Y. Zhang, M. A. Laurenzano, D. Meisner, T. Wenisch, J. Mars, L. Tang, and R. G. Dreslinski, "Adrenaline: Pinpointing and reining in tail queries with quick voltage boosting," in *Proc. IEEE 21st Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2015, pp. 271–282.

[13] R. Nishtala, P. Carpenter, V. Petrucci, and X. Martorell, "Hipster: Hybrid task manager for latency-critical cloud workloads," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2017, pp. 409–420.

[14] K.-D. Kang, G. Park, N. S. Kim, and D. Kim, "Network packet processing mode-aware power management for data center servers," *IEEE Comput. Archit. Lett.*, vol. 19, no. 1, pp. 1–4, Jan. 2020.

[15] L. Schaelicke, A. Davis, and S. A. McKee, "Profiling I/O interrupts in modern architectures," in *Proc. 8th Int. Symp. Modeling, Anal. Simul. Comput. Telecommun. Syst.*, 2000 pp. 115–123.

[16] J. C. Mogul and K. K. Ramakrishnan, "Eliminating receive livelock in an interrupt-driven kernel," *ACM Trans. Comput. Syst.*, vol. 15, no. 3, pp. 217–252, Aug. 1997.

[17] K. Salah, "To coalesce or not to coalesce," *AEU Int. J. Electron. Commun.*, vol. 61, no. 4, pp. 215–225, Apr. 2007.

[18] B. Fitzpatrick, "Distributed caching with memcached," *Linux J.*, vol. 124, p. 15, Dec. 2004.

[19] W. Reese, "Nginx: The high-performance Web server and reverse proxy," *Linux J.*, vol. 17, p. 2, Sep. 2008.

[20] *Infiniband Trade Association*. Accessed: Jul. 16, 2020. [Online]. Available: http://www.infinibandta.org/

[21] I. Intel, "82599 10 gbe controller datasheet," 2014.

[22] (2009). *Assigning Interrupts to Processor Cores Using an Intel 82575/82576 or 82598/82599 Ethernet Controller*. [Online]. Available: https://www.intel.com/content/www/us/en/ethernet-controllers/82575-8257%6-82598-82599-ethernet-controllers-interrupts-appl-note.html

[23] D. Brodowski and N. Golde. (2002). *Cpu Frequency and Voltage Scaling Code in the Linux Kernel*. [Online]. Available: https://www.kernel.org/doc/Documentation/cpu-freq/governors

[24] S. K. Shukla, D. Ghosal, and M. Farrens, "Tuning network I/O processing to achieve performance and energy objectives of latency critical workloads," in *Proc. IEEE 21st Int. Conf. High Perform.*, Aug. 2019, pp. 1499–1508.

[25] D. Meisner, B. T. Gold, and T. F. Wenisch, "Powernap: Eliminating server idle power," in *ACM Int. Conf. Architectural Support for Program. Lang. Operating Syst. (ASPLOS)*, pp. 205–216, 2009.

[26] X. Zhan, R. Azimi, S. Kanev, D. Brooks, and S. Reda, "CARB: A C-State power management arbiter for latency-critical workloads," *IEEE Comput. Archit. Lett.*, vol. 16, no. 1, pp. 6–9, Jan. 2017.

[27] C.-H. Chou, D. Wong, and L. N. Bhuyan, "DynSleep: Fine-grained power management for a latency-critical data center application," in *Proc. Int. Symp. Low Power Electron. Design - ISLPED*, 2016, pp. 212–217.

[28] E. Sharafzadeh, S. A. S. Kohroudi, E. Asyabi, and M. Sharifi, "Yawn: A CPU idle-state governor for datacenter applications," in *Proc. 10th ACM SIGOPS Asia–Pacific Workshop Syst.*, 2019, pp. 91–98.

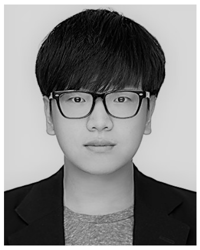
[29] Y. Liu, S. C. Draper, and N. S. Kim, "SleepScale: Runtime joint speed scaling and sleep states management for power efficient data centers," in *Proc. ACM/IEEE 41st Int. Symp. Comput. Archit. (ISCA)*, Jun. 2014, pp. 313–324.

[30] M. Alian, A. H. M. O. Abulila, L. Jindal, D. Kim, and N. S. Kim, "NCAP: Network-driven, packet context-aware power management for client-server architecture," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2017, pp. 25–36.

[31] C.-H. Chou, L. N. Bhuyan, and D. Wong, "DPM: Dynamic power management for the microsecond era," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2019, pp. 120–132.

KI-DONG KANG received the B.S. degree in computer science from Gachon University and the M.S. degree from the Department of Information and Communication Engineering, DGIST, in 2017, where he is currently pursuing the Ph.D. degree. His research interests include energy-efficient systems, network systems, and virtualization.

HYUNGWON PARK received the B.S. degree in computer science from Sahmyook University and the M.S. degree from the Department of Information and Communication Engineering, DGIST, in 2019, where he is currently pursuing the Ph.D. degree. His research interests include computer architecture, operating systems, virtualization, and cloud computing.



GYEONGSEO PARK received the B.S. degree in electrical and computer engineering from Ajou University, Suwon, South Korea, in 2016. He is currently pursuing the Ph.D. degree with the Department of Information and Communication Engineering, DGIST. His current research interests include computer architecture, network systems, and cloud computing.

DAEHOON KIM (Member, IEEE) received the B.S. degree in computer science from Yonsei University in 2008, and the Ph.D. degree in computer science from KAIST in 2014. He is currently an Assistant Professor with the Department of Information and Communication Engineering, DGIST. His research interests include computer architecture, operating systems, system virtualization, and cloud computing.