Impact of OS Design and Hardware Configuration on the Power Performance Tradeoff

Abstract

Energy proportional computing is a challenging feat when faced with the management of modern I/O intensive datacenter workloads with strict service level agreements (SLAs). We focus on saving power in the context of key-value stores that have 99% tail latencies in the hundreds of microseconds. Typically, additional power savings can be gained when server nodes are underutilized during periods of low to medium traffic by using CPU power limiting or frequency tuning methods. In this work, we tackle power consumption in relation to the dual operation of system software stacks and network interface cards (NICs). We first explore manipulating a time-based interrupt throttling mechanism on modern NICs to tune interrupt rates and control the batching of packet processing. By exploiting this feature and artificially increasing packet processing delay, a system is able to buffer more packets and process less interrupts. This method was able to achieve over 22% power savings compared to using CPU power limiting features when used in Linux.

We also compare and contrasted CPU and NIC tuning in the context of a baremetal library OS specialized for Memcached in order to understand their behavior in the performance-power tradeoff. Our baremetal library OS was able to achieve 2.5X higher peak throughput than Linux while using 3X lower power. Moreover, we find that our library OS is more than 2-3X more efficient in number of instructions per watt used, and therefor is more sensitive to CPU frequency changes. Our results demonstrate that optimizing the dataplane code path in system software stacks paired with the right hardware tuning can scale up I/O intensive workloads while effectively lowering power consumption.

1 Introduction

There is a fundamental tension between performance guarantees and energy consumption in modern datacenter workloads. They are often single purpose [4, 7, 16, 25] and I/O intensive with strict service level agreements (SLA) ranging from hundreds of microseconds to milliseconds [9, 40]. Kernel bypass [8, 11, 21, 36], and kernel specialization [3, 31, 46] methods have been proposed to address the performance problems. However, datacenters are already under increasingly constrained energy budgets [15, 48]. This complicates achieving their performance targets and puts pressure on general purpose system software stacks to be efficient in order to provide the throughput and latency guarantees.

The class of online, data-intensive (OLDI) workloads, one of which is Memcached [12], have been well studied from this perspective [27, 30, 33, 37]. Memcached workloads typically have a high fan-out deployment on many user facing servers with stringent SLAs in hundreds of microseconds. The Memcached service must also sustain high request rates and many incoming connections while maintaining the SLA specified low tail latencies. In addition, these workloads can operate in low to medium utilization levels due to diurnal patterns in user traffic [4, 33]. One way to improve energy efficiency of OLDI workloads under stringent SLAs is to scale a node's power consumption down under low to medium utilizations while satisfying the current SLA. Prior works [30, 37] have tackled this specific problem by using features on modern processors to modulate power consumption of CPUs, namely Running Average Power Limit (RAPL) [17] and Dynamic Voltage Frequency Scaling (DVFS) [10].

While the approach of exploiting hardware mechanisms to provide just enough performance to meet the required SLA is compelling, is limiting the power or frequency of the processor the most effective technique for I/O intesive workloads? We hypothesize that a hardware mechanism that directly correlated to the delay of requests could be more effective. To test this hypothesis, we explore using a hardware feature available on most modern network interface cards (NICs) that limit maximum interrupt rates. The Intel 82599 datasheet [20] defines a time-based interrupt throttling mechanism to limit maximum interrupt rates. We refer to this as *ITR-Delay*, it controls the time delay between packet reception and firing of the interrupt on a particular core. The effect of setting a higher interrupt delay can cause more receive packets to be buffered

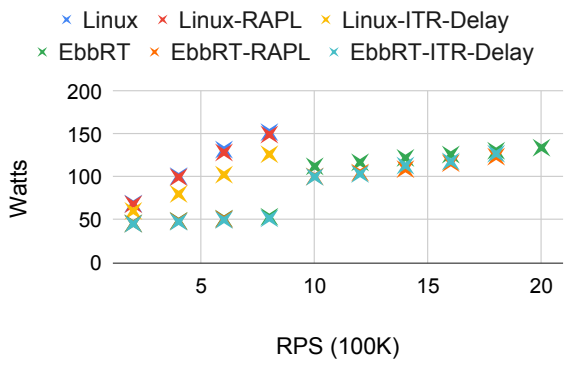


Figure 1: Performance vs Watts used

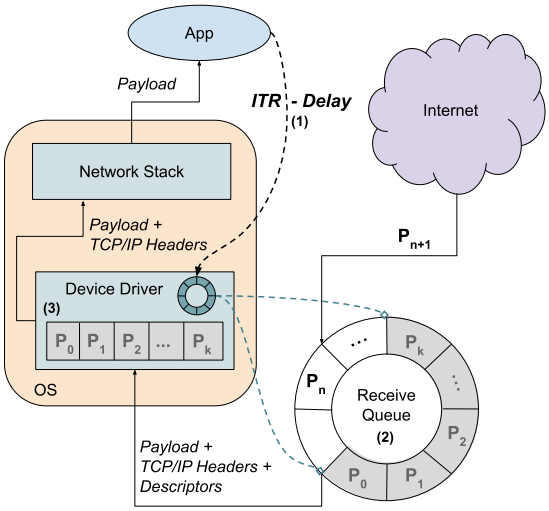


Figure 2: *ITR-Delay* mechanism.

and increase packet processing efficiency at a potential cost to response time. By default, the IXGBE device driver [19] updates the interrupt delay value dynamically after every packet receive to the interrupt delay value such that it can better handle the current traffic pattern. This value can also be tuned statically, which results in a interrupts being fired statically every X microseconds, where X is a configurable value. We take advantage of this ability by setting interrupt delay values statically for different request loads while maintaining SLA tail latency objectives. We find that controlling the batching of packet processing that happens underneath a complex system software stack can help find a sweet spot in performance to energy consumption ratios. Under a general purpose OS, we demonstrate a reduction in energy consumption by up to 22% compared to using *RAPL* power limiting in running Memcached.

Furthermore, there has been a lot of renewed interest in different specialized software stacks such as Unikernels and Library OSes to satisfy demand of modern datacenter workloads [31, 41]. They have shown benefits by optimizing the dataplane code path in order to scale up workloads such as Memcached. To understand how the unikernel approach compares to our hardware based one, we ported an existing library OS, EbbRT [46], that was developed for virtualized environments, to run on bare metal directly using the same physical NIC. While EbbRT virtualized was able to achieve 1.5X higher peak throughput than Linux native, Figure 1 shows that baremetal EbbRT was able to achieve more than 2.5X higher peak throughput.

We also explored the effects of tuning the same set of hardware knobs (*ITR-Delay*, *RAPL*, and *DVFS*) in the context of a baremetal library OS for Memcached. The goal is to understand the performance and power implications when an entire systems software stack is different. Figure 1 shows that by applying power limiting features and interrupt delay mechanisms, we are able to achieve an additional 16% in power savings when the CPU frequency is set at the highest frequency. It also shows that baremetal EbbRT is able to sustain the same request loads of running Memcached in Linux by using up to 3X less power when CPU frequency is set at the lowest frequency. This is partly due to the fact that EbbRT uses up to 3X fewer instructions than Linux to handle the same SLA objective in a multi-core Memcached experiment.

In the rest of the paper, we first explain the interrupt delay mechanism on the NIC and how it interacts with a general purpose OS stack such as Linux. We also shown how its dynamic interrupt delay is functioning during a Memcached workload. Next, we go into detail about how we set up the hardware, in terms of CPU configurations via the Intel Model Specific Registers (MSRs), how it is replicated in EbbRT and the multi-node experimental setup. We then describe two simple methods, *RAPL-MIN* and *ITR-MAX*, which are two iterative methods at finding peak power limit and maximum interrupt delay limit such that they maximize power while ensuring 99% tail latencies are still under the SLA objective. Finally, we present our experimental results in Section 5. We also discuss future directions.

2 Network Device Drivers

In this work, we use the Intel 82599ES 10GbE NIC in our experiments for *ITR-Delay* tuning. Figure 2 illustrates this mechanism of *ITR-Delay* in the context of a general system. *ITR-Delay* is a hardware knob that is exposed as a set of bits within a hardware register in the Intel 82599 datasheet [20]. In Linux, this hardware knob can be accessed via the *rx_usecs* parameter in *ethtool*. This parameter is tuned soley within the IXGBE device driver via a policy to improve the performance of packet processing. Figure 2 shows the overall process of how interrupt delays impact packet processing. As *ITR-Delay* (1) is turned up, this causes the NIC's Receive Queue (2) to

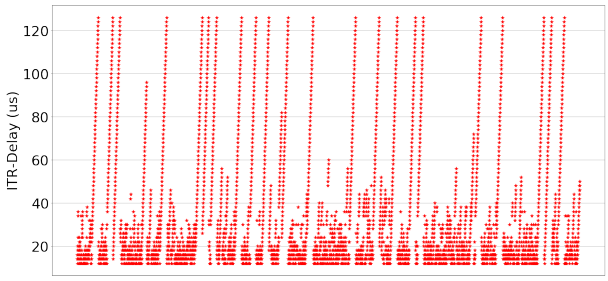


Figure 3: Updated interrupt delay values during memcached.

buffer more incoming packets as the NIC cannot fire new interrupts until the delay value has been reached. Once the interrupt is fired, Linux's NAPI polling mechanism kicks in and starts polling in new packets (3) to be processed until it either reaches its current work budget or if there is no more data to be processed. The 82599 NIC enables us to set interrupt delay values at a granularity of 2 microseconds and it can range from 0 to 1024 microseconds, where 0 indicates that an interrupt is fired for every packet receive, while 1024 means the device will wait for 1024 microseconds before initiating the next interrupt.

In Linux's IXGBE device driver, there is a dynamic algorithm that seeks to tune *ITR-Delay* such that it better reflects the current workload. It achieves this by using packets and bytes received from the last interrupt and classifies them broadly into a set of ranges that are pre-computed based off theoretical maximum wire speeds. Based off these ranges, they are then `switched` to a new delay value for the next interrupt. To demonstrate its behavior, we instrumented a simple logging tool into the IXGBE device driver. Figure 3 shows a small snapshot of its values while running a Memcached workload. Each marker represents a new *ITR-Delay* value. In its current implementation, it can only seek values between the range of 2 to 126 microseconds. The reason for this is that it was never designed towards a use case of aggressively delaying packet receive interrupts to take advantage of energy proportionality under stringent SLAs, which can potentially result in setting *ITR-Delay* values in the hundreds of microseconds.

It is also possible to disable this dynamic algorithm through the flip of a bit inside the device driver. This requires a rebuild of the driver module as it was not a configurable parameter. After flipping this bit, we can use Linux's *ethtool* to set new interrupt delay values statically to fire every X microseconds, where X is a configurable value. In our experimentation, we take advantage of this ability by setting interrupt delay values statically for different request loads while maintaining SLA tail latency objectives.

EbbRT's IXGBE device driver is written in C++ and totals over 3000 lines of code. It interfaces with a multicore TCP/IP network stack. We were careful to ensure that both EbbRT and Linux were configured similarly in terms of NIC features: receive-side coalescing (RSC) disabled, direct-cache injection (DCA) disabled, receive-side scaling (RSS) enabled to distribute packets for multi-core processing, hardware checksum offloading enabled. We also ensured the same values for parameters such as number of NIC transmit, receive descriptors and write-back thresholds for packet transmissions. A main difference in EbbRT is that it inherently does not have a dynamic policy for updating interrupt delay values since its implementation in Linux relies on other assumptions such as jiffies and NAPI polling budgets.

3 System Configuration

In our experiments, we use static configurations of *DVFS*, *RAPL*, and *ITR-Delay* in the following ways:

- Initially, *DVFS* is used to configure both Linux and EbbRT to the same CPU frequency. Linux is configured by overwriting each CPU's *cpufreq/scaling_governor* into various states. We explored two frequency states: *performance* and *powersave*, which represent the lowest and highest frequency avaiable on the processor. The main reason was to highlight the extremities in configuration to better understand differences when we apply additional power limiting and interrupt delay mechanisms. We validated that the above methods are correct in that new updated values appear in both IA32_PERF_CTL and MSR_PERF_STATUS registers [18]. In EbbRT, the same updated values are written to the corresponding MSRs to toggle between low and high CPU frequencies.

- *ITR-Delay* exists in two modes, the first is with the default policy that dynamically updates its value per receive interrupt (*ITR-DYNAMIC*), while the second is a static interrupt value. In Linux, both modes are explored in conjunction with the aforementioned *DVFS* states. In the static mode, a simple method of exploration is to repeatedly rerun the same workload but incrementally increase the interrupt delay value such that for a particular request load, the tail latencies lie just under the SLA. We indicate this as *ITR-MAX* in our results section.

- *RAPL* is applied in Linux in conjunction with *DVFS* states using the same approach as *ITR-MAX* but by iteratively decreasing the CPU's power limit until either the SLA objective is violated, the Memcached process fails to launch, or Memcached fails to create the necessary number of connections. The method used is indicated as *RAPL-MIN* in the results below. We did not explore *RAPL* and *ITR-Delay* together as the search space would have been too large.

In addition, we ensured that the processor for both Linux and baremetal EbbRT was setup as similarly as possible by carefully configuring both IA-32 Architectural MSRs (Table

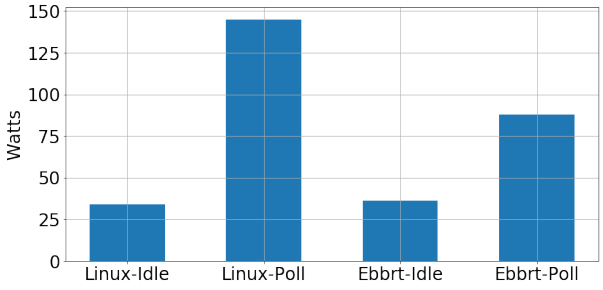


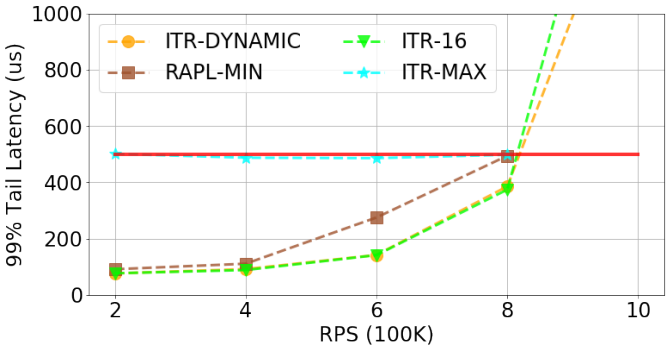
Figure 4: Power usage in poll and idle modes.

35-2) and processor specific MSRs (Table-35-18) in Intel's system programmer's manual [18]. One of EbbRT's design point is an event-driven model, its event loop is designed such that an interrupt that is continously fired after every static period of time has passed. To implement a similar CPU idle function as Linux, we added a MONITOR and MWAIT instruction in the event loop code to force it always to go into the C7 sleep state prior to an execution of a HLT instruction. Given this, Figure 4 illustrates the base efficiency of a specialized kernel. We use the Intel *RAPL* provided MSR_PKG_ENERGY_STATUS register was used to gather CPU power usage numbers. In polling mode, EbbRT results in a power savings of 39% over Linux.

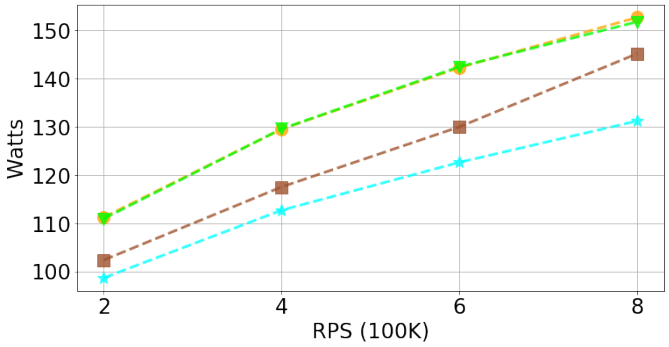
4 Experimental Approach

Our experimental cluster consists of a total of 7 nodes with a mix of 16 core Intel(R) Xeon(R) CPU E5-2690 @ 2.90GHz and 12 core Intel(R) Xeon(R) CPU E5-2630L v2 @ 2.40GHz processors and NICs with a mix of Solarflare Communications SFC9120 10G Ethernet Controller, and Intel Corporation 82599ES 10-Gigabit SFI/SFP+. All the processors have hyperthreads and Turbo Boost feature disabled. All workloads are pinned to physical cores to reduce potential background noise. In addition, *irqbalance* is disabled and packet receive interrupts are affinitized to their respective cores. The nodes have a mix of 125 GB RAM and 251 GB RAM configurations. We run memcached-1.5.6 on top of a Ubuntu LTS 18.0.4 installation with Linux 4.15.1, the other nodes either run Ubuntu LTS 18.0.4 with a mix of Linux 4.15.1 and 5.0.0 or Red Hat Enterprise Linux 7.7 with Linux 3.10.0.

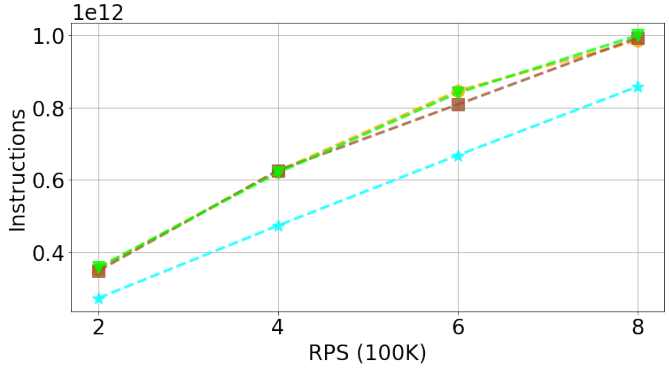
The same server node is also used to boot into baremetal EbbRT. It contains a 16 core Intel(R) Xeon(R) CPU E5-2690 @ 2.90GHz, with 125 GB of RAM and a Intel Corporation 82599ES 10-Gigabit SFI/SFP+ NIC. In contrast to Linux, EbbRT uses a re-implemented version of memcached written to EbbRT interfaces. It is a multicore application that supports the standard memcached binary protocol. To alleviate lock contention, a RCU hashtable is used to store key-value pairs. EbbRT's implementation does lack some additional features such as authentication and other commands but it is functional



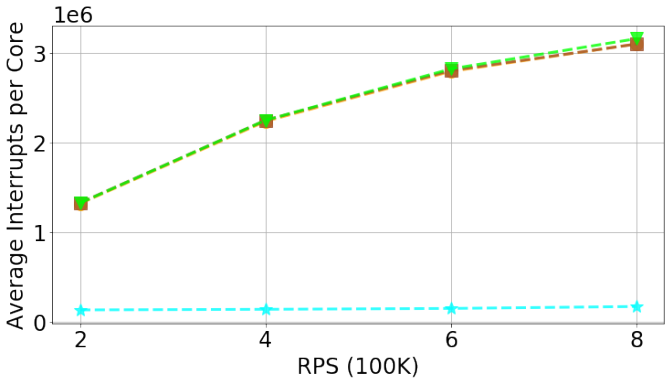
(a)



(b)



(c)



(d)

Figure 5: Linux Memcached ETC workload in *maximum* CPU frequency.

enough to support the benchmark tool.

We run Memcached on all 16 cores on the server node, the SLA objective in our experiments is to maintain tail latency of 99% requests under 500 us. There is another unloaded client node running *mutilate* as the master node for controlling and monitoring the experiment, it has two responsibilities: 1) coordinates with five other *mutilate* slave nodes to generate requests to the Memcached server and, 2) it measures tail latencies of all requests made. Two of the five nodes are 12 core machines while the rest are all 16 core machines. Each core creates 16 connections, for a total of 1152 connections amongst 5 nodes for a single memcached server. This setup is able to saturate the single 16 core server. We configured *mutilate* to pipeline up to four connections to further increase its request rate. We run a representative load from Facebook [4] (ETC), which represents the highest capacity deployment, it uses 20 Byte - 70 Byte keys, and 1 Byte to 1 KByte values, and contains 75% GET requests.

Given a static configuration listed above, we use *mutilate* to generate loads at different request per second (RPS) rates, each for a constant period of time while collecting additional system metrics. In Linux, *perf* is used to gather these metrics at a per second timeslice since the MSR_PKG_ENERGY_STATUS for reporting CPU power usage has a limited wrap around time. EbbRT is able to collect the same metrics as Linux as it contains a inbuilt *Perf* class which reads directly from Intel's PMC registers and a *Rapl* class which reads from Intel's RAPL registers. In EbbRT, an event is triggered to fire every second in order to log the corresponding data.

5 Results

5.1 Linux

In this section, we examine the power savings achievable in Linux when running Memcached at different RPS loads. Two methods are used, namely *RAPL-MIN* and *ITR-MAX* and the CPU frequency is at the highest. We also captured other metrics such as number of instructions and number of interrupts fired per core during each experiment to better understand efficiency gains. Figure 5a illustrates the 99% tail latency as a function of different request loads for Linux when configured with maximum CPU frequency. *ITR-MAX* is able to maximize SLA gap to drastically increase tail latency of requests such that they are consistently shifted just under the SLA objective, as indicated by the red line. In contrast, power limiting in *RAPL-MIN* is not able to achieve the same level of precision across the various request loads compared tuning NIC hardware registers. One reason is that *RAPL* power limiting is applied across the entire CPU die, which is a coarser-grained method that impacts all system components, therefore it is difficult to fully exploit the opportunity gaps during low to medium utilization of request loads while meeting SLA objectives. By aggressively delaying packet receive interrupts, *ITR-MAX* can enable more efficient packet processing by interacting with the NAPI polling algorithm in the IXGBE device driver. After every receive interrupt, NAPI polling will poll the receive descriptors for a period of time to process as many packets as it can within its given budget. This results in better coalescing of incoming packets and fewer wasted instructions on handling interrupts and other system work that needs to be done to handle every interrupt received. The immediate effect of this can be seen in Figure 5d where *ITR-MAX* uses more than 10X fewer interrupts per core to satisfy the same request load. With fewer interrupts to process, *ITR-MAX* also uses between 14%-22% fewer instructions as shown in Figure 5c. Compared to the power usage of default Linux (*ITR-DYNAMIC*), Figure 5b shows *RAPL-MIN* was able to achieve power savings between 5%-9% whereas *ITR-MAX* was able to achieve power savings between 11%-14%.

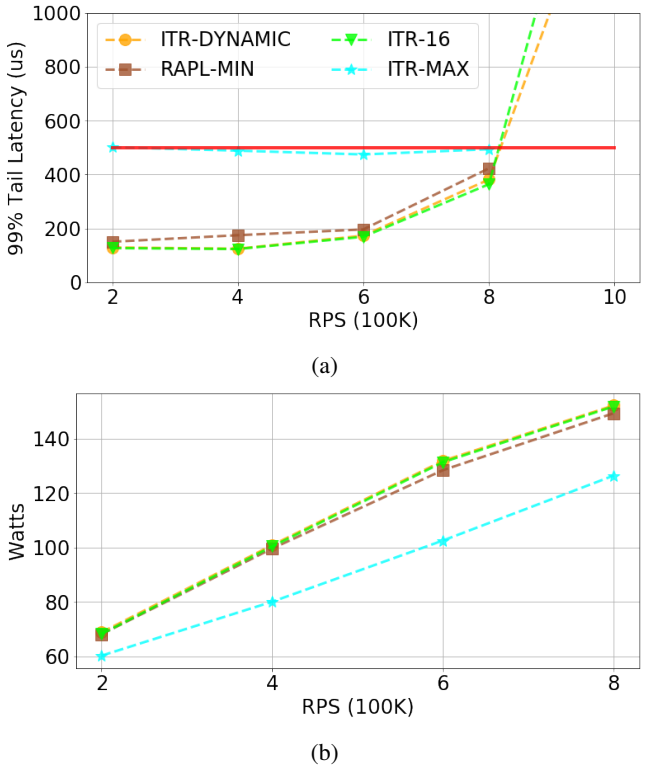


Figure 6: Linux Memcached ETC workload in *minimum* CPU frequency.

As an aside, we found a static interrupt delay of 16 microseconds (*ITR-16*) was able to approximate the same performance and power used as *ITR-DYNAMIC*, it is also prominently featured in the denser regions of Figure 3 under 20 microseconds. We wanted to highlight this in our graphs as it shows that the current dynamic policy in the IXGBE device driver is not necessarily targetted towards the benefit of an application such as Memcached, when one wants to delay

packet processing until tail latencies are just barely meeting SLA objectives. In contrast, what we've shown is that by catering the policy in a way such that it aggressively increases interrupt delay values while still being able to maintain the same request load under the SLA, we are able to do a more precise job in tackling this performance and power tradeoff.

The same experiment as above is conducted with the CPU frequency is set at the lowest in Figure 6. We can see that the peak performance of Memcached has not changed much compared to running at maximum frequency. To examine whether there is a real difference between maximum and minimum peak performance, we conducted further tests by slowly increasing RPS at smaller granularities and found that maximum frequency was only able to reach a higher peak throughput of 0.1% over the processor set at minimum frequency. This result suggests that Linux contains other system overheads that impede its ability to take advantage of the higher clock frequency to improve peak performance. In terms of comparing relative performance levels, the peak power savings using *ITR-MAX* in Figure 5b and Figure 6b, Memcached is able to maintain the same request loads while using 4%-40% less power at lowest CPU frequency. However, using *RAPL-MIN* in this setting had a minimal effect on power savings. At a RPS of 600K, Figure 6b shows *RAPL-MIN* managed an additional power savings of only 2.5% with whereas using *ITR-MAX* we were able to achieve 22.2% additional power savings. *RAPL* power limiting functions by setting a specific power limit during some average period of time that the processor can draw, therefore throttling its performance. This power limit is typically the Thermal Design Power (TDP) set by the manufacturer. This result suggests that when the processor is already running at minimum frequency (drawing the least amount of power), *RAPL* is not as effective a method to save additional power.

5.2 EbbRT

5.2.1 Maximum Processor Frequency

RPS(100K)	2	4	6	8
Linux-ITR-16	3.2e+09	4.8e+09	5.9e+09	6.6e+09
EbbRT-ITR-16	1.2e+09	2.1e+09	2.9e+09	3.6e+09

Table 1: Instructions/Watt comparison at different request loads for static interrupt delay at 16 microseconds.

There have been plenty of works in the past that seek to improve Memcached performance in Linux through systems software specialization methods [8, 22, 29, 36, 46]. While they seek the peak performance that could be gained, we are also curious about their implications in power usage and the tension in the power-performance tradeoff. Figure 7 compares a baremetal library OS, EbbRT, with Linux when the processor is set at highest frequency. Given that EbbRT does

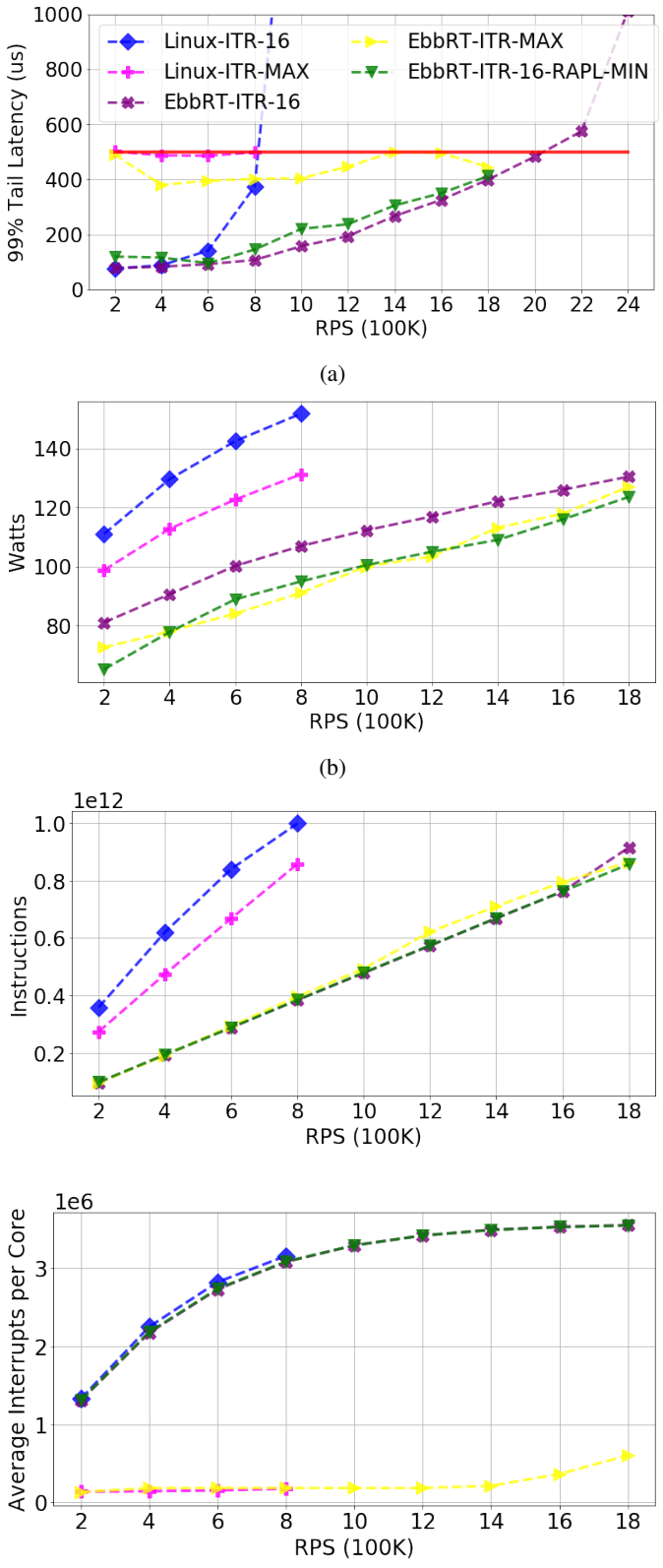


Figure 7: EbbRT Memcached ETC workload in *maximum* CPU frequency.

not have a dynamic way of setting interrupt delay values, we used a static interrupt delay of 16 microseconds (ITR-16) as a representative comparison point against Linux. Memcached written to EbbRT's interfaces was able to achieve 2.5X higher peak throughput compared to Linux. The benefits of EbbRT's implementation was demonstrated previously in a virtualized environment and achieved 1.5X higher throughput running in a VM versus baremetal Linux [46]. Therefore, running baremetal EbbRT enabled us to achieve even further performance as we bypass additional system overheads inside the VM. Being a specialized unikernel for Memcached, it also uses 2.1X-2.8X fewer instructions to support the request rate in Linux. In contrast to the NAPI polling policy, EbbRT is not limited by a polling budget per receive interrupt, it will process as many receive descriptors as the hardware allows in a synchronous manner, moreover as interrupts are disabled during this entire process, EbbRT potentially could process more packets than Linux which must balance workload with processing budgets. This point is reflected in the amount of interrupts fired per core where on average EbbRT *ITR-16* used 3% fewer interrupts per core compared to Linux. From a efficiency perspective, Figure 7b shows EbbRT was able to satisfy the same request load as Linux but use 30% less power on average.

We are also interested in comparing and contrasting the effects of tuning the same set of hardware knobs (*ITR-Delay*, *RAPL*, and *DVFS*) in the context of a baremetal library OS for Memcached. Figure 7b shows the power savings achieved in EbbRT by using *ITR-MAX*, similarly to Linux under maximum processor frequency 5b, *ITR-MAX* was able to further increase power savings compared to *EbbRT-ITR-16* from 3% to 17%. In contrast to Linux, using *RAPL-MIN* in EbbRT was able to achieve similar power savings as using *ITR-MAX*, we believe this is partly due to the prototype implementation of the IXGBE device driver in EbbRT as it is not fully optimized yet, further, Table 1 indicates that EbbRT uses less than of the Instructions per Watt at supporting the same workload as Linux, therefore power limiting has more of an impact as compared to Linux because EbbRT instructions are used more directly in executing the Memcached workload. Table 1 also helps explain why at peak throughput of 1800K RPS, EbbRT is able to achieve 2.25X higher throughput while using similar power as Linux did at peak throughput of 800K.

5.2.2 Minimum Processor Frequency

RPS(100K)	2	4	6	8
Linux-ITR-16	1.1e+10	1.3e+10	1.4e+10	1.5e+10
EbbRT-ITR-16	5.4e+09	8.6e+09	1.1e+10	1.3e+10

Table 2: Instructions/Watt comparison at different request loads for static interrupt delay at 16 microseconds.

In contrast to Linux, Figure 8 shows that changing proces-

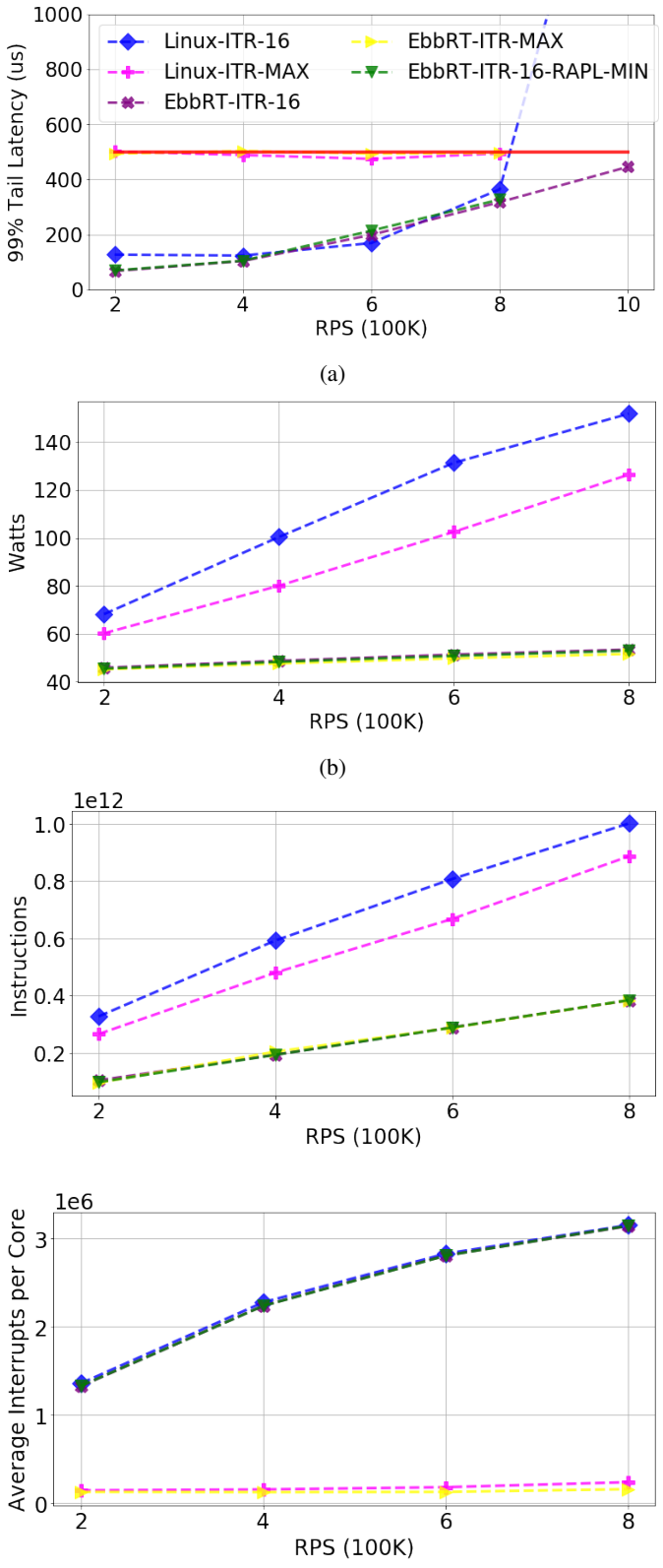


Figure 8: EbbRT Memcached ETC workload in *minimum* CPU frequency.

sor frequency from maximum to minimum resulted in a 2X performance penalty in EbbRT. Unsurprisingly, this result also demonstrate that by running a specialized kernel for a particular network workload, the optimized dataplane path resulted in pushing the processing bottleneck onto the CPU. Therefore resulting in a 2X performance penalty when we artificially lowered the CPU's frequency. Past dataplane operating systems research demonstrated packet processing bottlenecks that exists in traditional systems software stack [8,36,46], they were able to gain further performance advantages by optimizing the dataplane code path. In this example, we demonstrate that these systems will be more affected by CPU frequency tuning methods as their codebase is more optimized.

Even at the lowest CPU frequency, EbbRT was still able to achieve 25% higher peak throughput than Linux. Further, it is able to satisfy the specified RPS rates at the lowest power usage overall. At a RPS rate of 800K, *Linux-ITR-MAX* achieves a minimum power usage of 126 Watts while EbbRT was able to use 2.4X less power at 53 Watts. This result highlights the impact of processor tuning when used in conjunction with a more efficient systems software stack.

The efficiency comparison numbers between EbbRT and Linux in Section 5.2.1 shows that as EbbRT is more than 2X more efficient at using its instructions to do the same amount of work as Linux, therefore, changing the processor frequency had a much bigger impact on overall performance. This effect can also be seen in the relative reduction in power consumption between switching CPU frequency in Linux and EbbRT. In Linux, moving from a high frequency to a low frequency resulted in power savings between 4%-40% for a set of RPS loads under the SLA. For the same RPS loads in EbbRT, the power savings was between 56%-200%.

We also found that adding *ITR-DELAY* and *RAPL-MAX* to gain additional power savings in this setting had a minimal effect overall, with only additional power savings of 1%-2% on average. This phenomenon could be similar to results shown in Figure 6b as the CPU is already operating at such a lower frequency that doing additional coalescing of receive packets and power limiting have a negligible overall effect on power consumption. While EbbRT uses 2.6X-3.1X fewer instructions compared to Linux in Figure **??**, Table 2 shows that an efficiency measure in terms of Instructions per Watt was only between 1.14X - 2X as the system is already operating in a minimal power setting.

6 Related Work

There have been past works that looked at energy proportional computing under strict SLA objectives when applied to in-memory key-value stores [30, 37]. They developed dynamic algorithms such as fine tuning CPU frequency via *DVFS* or statically searching a large space and removing CPU cores in conjunction with *RAPL* to lower overall power usage while maintaing SLA objectives. Our work takes a look at the same problem from a different angle by going directly at source of packet processing by delaying when packets actually get processed. The authors in [37] also looked at energy savings in the context of *ix* a specialized library OS for Memcached, however, their measurements were taken in a virtualized environment while we used a baremetal library OS. In [33], the authors proposed wholistic system optimization of CPUs, RAM and storage towards this energy proportional goal. Our work leads credence to their proposal by demonstrating a new aspect of NIC tuning that can be used. Overall, we do believe our work can co-exist with these other approaches as another layer on top of enabling better system efficiency while meeting SLA objectives in OLDI workloads.

There has been a large body of work in understanding complexity of device drivers [24] and various aspects of their reliability [5, 28, 43], configuration [42, 44, 47], and performance [13, 45]. The focus is mainly on device driver interface exposed to systems software, usually in order to provide a unified way to synthesize device drivers across different hardware and reduce code complexity while improving fault tolerance. While we do not address the inherent complexity in device drivers, our work shows the benefits if device driver interfaces are more accessibile towards application writers by exposing more hardware knobs that are easily tunable.

Similarly, there have been past projects to enable developers to make better use of NIC features by exposing NICs in a programmable way: FlexNIC [26] enables dynamic modification of RSS rules to better load balance in Memcached, Affinity-Accept [35] uses the Flow Director capability on IXGBE NICs in order to ensure TCP connections are always affinitized to the same core. Our work is inspired by these approaches in taking an advantage of another feature on the same Intel 82599 family of NICs and using it in a way that is more amenable towards an applications' energy goal.

There is a proliferation of new *smart* hardware such as NICs [1, 34], SSDs [14], and memory [2] to in order to bring computation closer to the data [6, 39] and potentially improve overall application energy efficiency. Subsequently, these devices are expected to join the existing plethora of programmable accelerators [23, 38] within a modern datacenter [32]. One potential problem posed by this diversity is the number of new knobs that are exposed to developers and how to better specialize towards their application. In this work, the benefits we show in tuning a single hardware register that exists on modern NICs stresses the importance in exposing more of these hardware knobs.

7 Discussion and Future Work

There are a few advantages and disadvantages to the *ITR-Delay* mechanism. As it can only be set to a maximum of 1024 microseconds, it remains to be seen how its benefits fare in settings with SLA's in the milliseconds range. Further, it must also interact with retransmit timeout policies in the TCP/IP

stack. Potentially if one delays the interrupt mechanism too long, it could cause redundant retransmissions and decrease overall throughput. *ITR-Delay* can also be modified on a per receive queue basis, which are typically mapped to a distinct core. Potentially, if different workloads with different SLA objectives are consolidated on a single server node, it could be possible to customize interrupt receive delays on a per application basis while maintaing different SLAs across the diverse set of applications. This would result in even better additional power savings over methods such as *RAPL* power limiting as it is a wholistic approach that applies to the entire processor.

One aspect not covered in this work is the combination *RAPL-MIN* and *ITR-MAX* tuning, we did not explore this mainly due to the larger potential search space. For example, one approach would be to use first use *ITR-MAX* to maximize interrupt delay and then apply *RAPL-MIN* after while ensuring SLA objectives are still met, another would be the reverse of this process. The benefit of this can be seen in the figures above where *RAPL-MIN* was able to lower power usage until a point where the application failed to launch, however, its 99% tail latency was still greatly under the SLA. This presents an opportunity to then add *ITR-MAX* to delay packet processing and enable additional power savings. Moreover, it is unknown if the best power savings actually derives from some intermediate combination of both methods. Given this, we are also interested in exploring the use of learning methods to bridge this large search space, the goal would be to develop a system module that can use application performance metrics combined with other metrics such as SLA in order to identify system configurations of CPU and the NIC that can optimize power savings as an objective.

Furthermore, there is an increasing need to readdress the complexity of modern NIC drivers **??**, examining the Intel 82599 datasheet [20] revealed over 5600 writable and read-only 32-bit registers and *ITR-Delay* only represented one of them. Other than *ITR-Delay* there are other hardware registered that have a direct on application processing of packets such as Receive Side Coalescing, as well as other tunable parameters in packet transmission. We are also interested in the impact that tuning these parameters have on a general purpose OS and a specialized OS.

8 Conclusion

We present our work in exploiting the interrupt delay mechanism that exist on modern NICs to lower power usage in Memcached while maintaining SLA objectives for different low to medium level request rates. We compare and contrast this method with power limiting features under two different CPU frequency settings. One of the contributions of this work is demonstrating the advantages gained by catering static interrupt delay values to a application specific manner. The mechanism to delay packet processing can be tuned in a fine-grained manner at the microsecond scale, this enables greater control over request latencies. When applied in Linux, we were able to achieve peak power savings of 22% over *RAPL* power limiting. We also examined the impact of hardware tuning in a barmetal library OS specialized for Memcached. In this setting, we found that since a specialized library OS is able to take better advantage of CPU frequency, changing CPU frequency was able to save over 2X the power as compared to operating in highest CPU frequency while sacrificing 2X peak throughput. Further, we found that a baremetal library OS was able to achieve 2.5X higher peak throughput than Linux while using 3X lower power.

References

[1] http://www.businesswire.com/news/home/20160425005805/en/Netronome-Brings-Efficient-Hardware-Accelerated-OpenS

[2] Agrawal, Sandeep R and Idicula, Sam and Raghavan, Arun and Vlachos, Evangelos and Govindaraju, Venkatraman and Varadarajan, Venkatanathan and Balkesen, Cagri and Giannikis, Georgios and Roth, Charlie and Agarwal, Nipun and Sedlar, Eric. A Many-core Architecture for In-memory Data Processing. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-50 '17, pages 245–258, New York, NY, USA, 2017. ACM.

[3] Antti Kantee, Justin Cormack. Rump Kernels: No OS? No Problem! https://www.usenix.org/publications/login/october-2014-vol-39-no-5.

[4] Atikoglu, Berk and Xu, Yuehai and Frachtenberg, Eitan and Jiang, Song and Paleczny, Mike. Workload Analysis of a Large-scale Key-value Store. In *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '12, pages 53–64, New York, NY, USA, 2012. ACM.

[5] Thomas Ball, Ella Bounimova, Byron Cook, Vladimir Levin, Jakob Lichtenberg, Con McGarvey, Bohus Ondrusek, Sriram K. Rajamani, and Abdullah Ustuner. Thorough static analysis of device drivers. In *Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006*, EuroSys '06, pages 73–85, New York, NY, USA, 2006. ACM.

[6] Barbalace, Antonio and Iliopoulos, Anthony and Rauchfuss, Holm and Brasche, Goetz. It's Time to Think About an Operating System for Near Data Processing Architectures. In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*, HotOS '17, pages 56–61, New York, NY, USA, 2017. ACM.

[7] Luiz Andre Barroso, Jeffrey Dean, and Urs Hölzle. Web search for a planet: The google cluster architecture. *IEEE Micro*, 23:22–28, 2003.

[8] Adam Belay, George Prekas, Ana Klimovic, Samuel Grossman, Christos Kozyrakis, and Edouard Bugnion. IX: A Protected Dataplane Operating System for High Throughput and Low Latency. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 49–65, Broomfield, CO, October 2014. USENIX Association.

[9] Dean, Jeffrey and Barroso, Luiz André. The Tail at Scale. *Commun. ACM*, 56(2):74–80, February 2013.

[10] Dominik Brodowski, Nico Golde, Rafael J. Wysocki, Viresh Kumar. CPU frequency and voltage scaling code in the Linux(TM) kernel. https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt.

[11] Aleksandar Dragojević, Dushyanth Narayanan, Miguel Castro, and Orion Hodson. Farm: Fast remote memory. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 401–414, Seattle, WA, 2014. USENIX Association.

[12] Brad Fitzpatrick. Distributed Caching with Memcached. *Linux Journal*, 2004(124):5, August 2004.

[13] Vinod Ganapathy, Matthew J. Renzelmann, Arini Balakrishnan, Michael M. Swift, and Somesh Jha. The design and implementation of microdrivers. In *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XIII, pages 168–178, New York, NY, USA, 2008. ACM.

[14] Boncheol Gu, Andre S. Yoon, Duck-Ho Bae, Insoon Jo, Jinyoung Lee, Jonghyun Yoon, Jeong-Uk Kang, Moonsang Kwon, Chanho Yoon, Sangyeun Cho, Jaeheon Jeong, and Duckhyun Chang. Biscuit: A framework for near-data processing of big data workloads. In *Proceedings of the 43rd International Symposium on Computer Architecture*, ISCA '16, pages 153–165, Piscataway, NJ, USA, 2016. IEEE Press.

[15] Chang-Hong Hsu, Qingyuan Deng, Jason Mars, and Lingjia Tang. Smoothoperator: Reducing power fragmentation and improving power utilization in large-scale datacenters. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '18, pages 535–548, New York, NY, USA, 2018. ACM.

[16] Huang, Qi and Ang, Petchean and Knowles, Peter and Nykiel, Tomasz and Tverdokhlib, Iaroslav and Yajurvedi, Amit and Dapolito,IV, Paul and Yan, Xifan and Bykov, Maxim and Liang, Chuen and Talwar, Mohit and Mathur, Abhishek and Kulkarni, Sachin and Burke, Matthew and Lloyd, Wyatt. SVE: Distributed Video Processing at Facebook Scale. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, pages 87–103, New York, NY, USA, 2017. ACM.

[17] Intel. Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3B:System Programming Guide, Part 2. https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-3b-part pdf.

[18] Intel. Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3C:System Programming Guide, Part 3. https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-3c-part pdf.

[19] Intel. Linux ixgbe* Base Driver Overview and Installation. https://www.intel.com/content/www/us/en/support/articles/000005688/network-and-i-o/ethernet-products.html.

[20] Intel 82599 10 Gigabit Ethernet Controller: Datasheet. https://www.intel.com/content/www/us/en/embedded/products/networking/82599-10-gbe-controller-datasheet.html.

[21] Intel Corporation. Intel DPDK: Data Plane Development Kit. http://dpdk.org.

[22] EunYoung Jeong, Shinae Wood, Muhammad Jamshed, Haewon Jeong, Sunghwan Ihm, Dongsu Han, and KyoungSoo Park. mtcp: a highly scalable user-level TCP stack for multicore systems. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 489–502, Seattle, WA, April 2014. USENIX Association.

[23] Jouppi, Norman P. and Young, Cliff and Patil, Nishant and Patterson, David and Agrawal, Gaurav and Bajwa, Raminder and Bates, Sarah and Bhatia, Suresh and Boden, Nan and Borchers, Al and Boyle, Rick and Cantin, Pierre-luc and Chao, Clifford and Clark, Chris and Coriell, Jeremy and Daley, Mike and Dau, Matt and Dean, Jeffrey and Gelb, Ben and Ghaemmaghami, Tara Vazir and Gottipati, Rajendra and Gulland, William and Hagmann, Robert and Ho, C. Richard and Hogberg, Doug and Hu, John and Hundt, Robert and Hurt, Dan and Ibarz, Julian and Jaffey, Aaron and Jaworski, Alek and Kaplan, Alexander and Khaitan, Harshit and Killebrew, Daniel and Koch, Andy and Kumar, Naveen and Lacy, Steve and Laudon, James and Law, James and Le, Diemthu and Leary, Chris and Liu, Zhuyuan and Lucke,

Kyle and Lundin, Alan and MacKean, Gordon and Maggiore, Adriana and Mahony, Maire and Miller, Kieran and Nagarajan, Rahul and Narayanaswami, Ravi and Ni, Ray and Nix, Kathy and Norrie, Thomas and Omernick, Mark and Penukonda, Narayana and Phelps, Andy and Ross, Jonathan and Ross, Matt and Salek, Amir and Samadiani, Emad and Severn, Chris and Sizikov, Gregory and Snelham, Matthew and Souter, Jed and Steinberg, Dan and Swing, Andy and Tan, Mercedes and Thorson, Gregory and Tian, Bo and Toma, Horia and Tuttle, Erick and Vasudevan, Vijay and Walter, Richard and Wang, Walter and Wilcox, Eric and Yoon, Doe Hyun. In-Datacenter Performance Analysis of a Tensor Processing Unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ISCA '17, pages 1–12, New York, NY, USA, 2017. ACM.

[24] Asim Kadav and Michael M. Swift. Understanding modern device drivers. In *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVII, pages 87–98, New York, NY, USA, 2012. ACM.

[25] Kanev, Svilen and Darago, Juan Pablo and Hazelwood, Kim and Ranganathan, Parthasarathy and Moseley, Tipp and Wei, Gu-Yeon and Brooks, David. Profiling a Warehouse-scale Computer. In *Proceedings of the 42Nd Annual International Symposium on Computer Architecture*, ISCA '15, pages 158–169, New York, NY, USA, 2015. ACM.

[26] Antoine Kaufmann, SImon Peter, Naveen Kr. Sharma, Thomas Anderson, and Arvind Krishnamurthy. High performance packet processing with flexnic. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '16, page 67–81, New York, NY, USA, 2016. Association for Computing Machinery.

[27] Andrew Krioukov, Prashanth Mohan, Sara Alspaugh, Laura Keys, David Culler, and Randy Katz. Napsac: Design and implementation of a power-proportional web cluster. *SIGCOMM Comput. Commun. Rev.*, 41(1):102–108, January 2011.

[28] Joshua LeVasseur, Volkmar Uhlig, Jan Stoess, and Stefan Götz. Unmodified device driver reuse and improved system dependability via virtual machines. In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, OSDI'04, pages 2–2, Berkeley, CA, USA, 2004. USENIX Association.

[29] Hyeontaek Lim, Dongsu Han, David G. Andersen, and Michael Kaminsky. MICA: A holistic approach to fast in-memory key-value storage. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 429–444, Seattle, WA, April 2014. USENIX Association.

[30] David Lo, Liqun Cheng, Rama Govindaraju, Luiz André Barroso, and Christos Kozyrakis. Towards energy proportionality for large-scale latency-critical workloads. *SIGARCH Comput. Archit. News*, 42(3):301–312, June 2014.

[31] Anil Madhavapeddy, Richard Mortier, Charalampos Rotsos, David Scott, Balraj Singh, Thomas Gazagnaire, Steven Smith, Steven Hand, and Jon Crowcroft. Unikernels: Library operating systems for the cloud. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '13, pages 461–472, New York, NY, USA, 2013. ACM.

[32] Mark Silberstein. Accelerators in data centers: the systems perspective. https://www.sigarch.org/accelerators-in-data-centers-the-systems-perspective/

[33] David Meisner, Christopher M. Sadler, Luiz André Barroso, Wolf-Dietrich Weber, and Thomas F. Wenisch. Power management of online data-intensive services. *SIGARCH Comput. Archit. News*, 39(3):319–330, June 2011.

[34] Mellanox Innova SmartNIC. http://www.businesswire.com/news/home/20160615005424/en/.

[35] Aleksey Pesterev, Jacob Strauss, Nickolai Zeldovich, and Robert T. Morris. Improving network connection locality on multicore systems. In *Proceedings of the 7th ACM European Conference on Computer Systems*, EuroSys '12, page 337–350, New York, NY, USA, 2012. Association for Computing Machinery.

[36] Simon Peter, Jialin Li, Irene Zhang, Dan R. K. Ports, Doug Woos, Arvind Krishnamurthy, Thomas Anderson, and Timothy Roscoe. Arrakis: The Operating System is the Control Plane. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 1–16. USENIX Association, October 2014.

[37] George Prekas, Mia Primorac, Adam Belay, Christos Kozyrakis, and Edouard Bugnion. Energy proportionality and workload consolidation for latency-critical applications. In *Proceedings of the Sixth ACM Symposium on Cloud Computing*, SoCC '15, page 342–355, New York, NY, USA, 2015. Association for Computing Machinery.

[38] Andrew Putnam, Adrian M. Caulfield, Eric S. Chung, Derek Chiou, Kypros Constantinides, John Demme,

Hadi Esmaeilzadeh, Jeremy Fowers, Gopi Prashanth Gopal, Jan Gray, Michael Haselman, Scott Hauck, Stephen Heil, Amir Hormati, Joo-Young Kim, Sitaram Lanka, James Larus, Eric Peterson, Simon Pope, Aaron Smith, Jason Thong, Phillip Yi Xiao, and Doug Burger. A reconfigurable fabric for accelerating large-scale datacenter services. In *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, ISCA '14, pages 13–24, Piscataway, NJ, USA, 2014. IEEE Press.

[39] R. Balasubramonian and J. Chang and T. Manning and J. H. Moreno and R. Murphy and R. Nair and S. Swanson. Near-Data Processing: Insights from a MICRO-46 Workshop. *IEEE Micro*, 34(4):36–42, July 2014.

[40] Rajesh Nishtala and Hans Fugal and Steven Grimm and Marc Kwiatkowski and Herman Lee and Harry C. Li and Ryan McElroy and Mike Paleczny and Daniel Peek and Paul Saab and David Stafford and Tony Tung and Venkateshwaran Venkataramani. Scaling Memcache at Facebook. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 385–398, Lombard, IL, 2013. USENIX.

[41] Ali Raza, Parul Sohal, James Cadden, Jonathan Appavoo, Ulrich Drepper, Richard Jones, Orran Krieger, Renato Mancuso, and Larry Woodman. Unikernels: The next stage of linux's dominance. In *Proceedings of the Workshop on Hot Topics in Operating Systems*, HotOS '19, page 7–13, New York, NY, USA, 2019. Association for Computing Machinery.

[42] Matthew J. Renzelmann and Michael M. Swift. Decaf: Moving device drivers to a modern language. In *Proceedings of the 2009 Conference on USENIX Annual Technical Conference*, USENIX'09, pages 14–14, Berkeley, CA, USA, 2009. USENIX Association.

[43] Leonid Ryzhyk, Peter Chubb, Ihor Kuz, and Gernot Heiser. Dingo: Taming device drivers. In *Proceedings of the 4th ACM European Conference on Computer Systems*, EuroSys '09, pages 275–288, New York, NY, USA, 2009. ACM.

[44] Leonid Ryzhyk, Adam Walker, John Keys, Alexander Legg, Arun Raghunath, Michael Stumm, and Mona Vij. User-guided device driver synthesis. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, OSDI'14, pages 661–676, Berkeley, CA, USA, 2014. USENIX Association.

[45] Leonid Ryzhyk, Yanjin Zhu, and Gernot Heiser. The case for active device drivers. In *Proceedings of the First ACM Asia-pacific Workshop on Workshop on Systems*, APSys '10, pages 25–30, New York, NY, USA, 2010. ACM.

[46] Dan Schatzberg, James Cadden, Han Dong, Orran Krieger, and Jonathan Appavoo. Ebbrt: A framework for building per-application library operating systems. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 671–688, GA, 2016. USENIX Association.

[47] Adrian Schüpbach, Andrew Baumann, Timothy Roscoe, and Simon Peter. A declarative language approach to device configuration. In *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVI, pages 119–132, New York, NY, USA, 2011. ACM.

[48] Qiang Wu, Qingyuan Deng, Lakshmi Ganesh, Chang-Hong Hsu, Yun Jin, Sanjeev Kumar, Bin Li, Justin Meza, and Yee Jiun Song. Dynamo: Facebook's data center-wide power management system. In *Proceedings of the 43rd International Symposium on Computer Architecture*, ISCA '16, pages 469–480, Piscataway, NJ, USA, 2016. IEEE Press.