E3: Energy-Efficient Microservices on SmartNIC-Accelerated Servers

Ming Liu
University of Washington

Simon Peter
The University of Texas at Austin

Arvind Krishnamurthy
University of Washington

Phitchaya Mangpo Phothilimthana*
University of California, Berkeley

Abstract

We investigate the use of SmartNIC-accelerated servers to execute microservice-based applications in the data center. By offloading suitable microservices to the SmartNIC's low-power processor, we can improve server energy-efficiency without latency loss. However, as a heterogeneous computing substrate in the data path of the host, SmartNICs bring several challenges to a microservice platform: network traffic routing and load balancing, microservice placement on heterogeneous hardware, and contention on shared SmartNIC resources.

We present E3, a microservice execution platform for SmartNIC-accelerated servers. E3 follows the design philosophies of the Azure Service Fabric microservice platform and extends key system components to a SmartNIC to address the above-mentioned challenges. E3 employs three key techniques: ECMP-based load balancing via SmartNICs to the host, network topology-aware microservice placement, and a data-plane orchestrator that can detect SmartNIC overload. Our E3 prototype using Cavium LiquidIO SmartNICs shows that SmartNIC offload can improve cluster energy-efficiency up to 3× and cost efficiency up to 1.9× at up to 4% latency cost for common microservices, including real-time analytics, an IoT hub, and virtual network functions.

1 Introduction

Energy-efficiency has become a major factor in data center design [80]. U.S. data centers consume an estimated 70 billion kilowatt-hours of energy per year (about 2% of total U.S. energy consumption) and as much as 57% of this energy is used by servers [22, 74]. Improving server energy-efficiency is thus imperative [17]. A recent option is the integration of low-power processors in server network interface cards (NICs). Examples are the Netronome Agilio-CX [59], Mellanox BlueField [51], Broadcom Stingray [13], and Cavium LiquidIO [15], which rely on ARM/MIPS-based processors and on-board memory. These SmartNICs can process microsecond-scale client requests but consume much less energy than server CPUs. By sharing idle power and the chassis with host servers, SmartNICs also promise to be more energy and cost efficient than other heterogeneous or low-power clusters. However, SmartNICs are not powerful enough to run large, monolithic cloud applications, preventing their offload.

Today, cloud applications are increasingly built as microservices, prompting us to revisit SmartNIC offload in the cloud. A microservice-based workload comprises loosely coupled processes, whose interaction is described via a dataflow graph. Microservices often have a small enough memory footprint for SmartNIC offload and their programming model efficiently supports transparent execution on heterogeneous platforms. Microservices are deployed via a microservice platform [3–5, 40] on shared datacenter infrastructure. These platforms abstract and allocate physical datacenter computing nodes, provide a reliable and available execution environment, and interact with deployed microservices through a set of common runtime APIs. Large-scale web services already use microservices on hundreds of thousands of servers [40, 41].

In this paper, we investigate efficient microservice execution on SmartNIC-accelerated servers. Specifically, we are exploring how to integrate multiple SmartNICs per server into a microservice platform with the goal of achieving better energy efficiency at minimum latency cost. However, transparently integrating SmartNICs into microservice platforms is non-trivial. Unlike traditional heterogeneous clusters, SmartNICs are collocated with their host servers, raising a number of issues. First, SmartNICs and hosts share the same MAC address. We require an efficient mechanism to route and load-balance traffic to hosts and SmartNICs. Second, SmartNICs sit in the host's data path and microservices running on a SmartNIC can interfere with microservices on the host. Microservices need to be appropriately placed to balance network-to-compute bandwidth. Finally, microservices can contend on shared SmartNIC resources, causing overload. We need to efficiently detect and prevent such situations.

We present E3, a microservice execution platform for SmartNIC-accelerated servers that addresses these issues. E3

*The author is now at Google.

follows the design philosophies of the Azure Service Fabric microservice platform [40] and extends key system components to allow transparent offload of microservices to a SmartNIC. To balance network request traffic among SmartNICs and the host, E3 employs equal-cost multipath (ECMP) load balancing at the top-of-rack (ToR) switch and provides high-performance PCIe communication mechanisms between host and SmartNICs. To balance computation demands, we introduce *HCM*, a hierarchical, communication-aware microservice placement algorithm, combined with a data-plane orchestrator that can detect and eliminate SmartNIC overload via microservice migration. This allows E3 to optimize server energy efficiency with minimal impact on client request latency.

We make the following contributions:

- We show why SmartNICs can improve energy efficiency over other forms of heterogeneous computation and how they should be integrated with data center servers and microservice platforms to provide efficient and transparent microservice execution (§2).
- We present the design of E3 (§3), a microservice runtime on SmartNIC-accelerated server systems. We present its implementation within a cluster of Xeon-based servers with up to 4 Cavium LiquidIO-based SmartNICs per server (§4).
- We evaluate energy and cost-efficiency, as well as client-observed request latency and throughput for common microservices, such as a real-time analytics framework, an IoT hub, and various virtual network functions, across various homogeneous and heterogeneous cluster configurations (§5). Our results show that offload of microservices to multiple SmartNICs per server with E3 improves cluster energy-efficiency up to 3× and cost efficiency up to 1.9× at up to 4% client-observed latency cost versus all other cluster configurations.

2 Background

Microservices simplify distributed application development and are a good match for low-power SmartNIC offload. Together, they are a promising avenue for improving server energy efficiency. We discuss this rationale, quantify the potential benefits, and outline the challenges of microservice offload to SmartNICs in this section.

2.1 Microservices

Microservices have become a critical component of today's data center infrastructure with a considerable and diverse workload footprint. Microsoft reports running microservices 24/7 on over 160K machines across the globe, including Azure SQL DB, Skype, Cortana, and IoT suite [40]. Google reports that Google Search, Ads, Gmail, video processing, flight search, and more, are deployed as microservices [41]. These microservices include large and small data and code footprints, long and short running times, billed by run-time

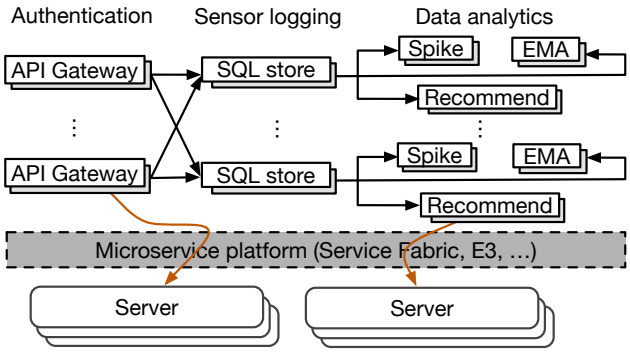


Figure 1: Thermostat analytics as DAG of microservices. The platform maps each DAG node to a physical computing node.

and by remote procedure call (RPC) [28]. What unifies these services is their software engineering philosophy.

Microservices use a modular design pattern, which simplifies distributed application design and deployment. Microservices are loosely-coupled, communicating through a set of common APIs, invoked via RPCs [86], and maintain state via *reliable collections* [40]. As a result, developers can take advantage of languages and libraries of their choice, while not having to worry about microservice placement, communication mechanisms, fault tolerance, or availability.

Microservices are also attractive to datacenter operators as they provide a way to improve server utilization. Microservices execute as light-weight processes that are easier to scale and migrate compared with a monolithic development approach. They can be activated upon incoming client requests, execute to request completion, and then swapped out.

A microservice platform, such as Azure Service Fabric [40], Amazon Lambda [3], Google Application Engine [4], or Nirmata [5], is a distributed system manager that enables isolated microservice execution on shared datacenter infrastructure. To do so, microservice platforms include the following components (cf. [40]): 1. *federation subsystem*, abstracting and grouping servers into a unified cluster that holds deployed applications; 2. *resource manager*, allocating computation resources to individual microservices based on their execution requirements; 3. *orchestrator*, dynamically scheduling and migrating microservices within the cluster based on node health information, microservice execution statistics, and service-level agreements (SLAs); 4. *transport subsystem*, providing (secure) point-to-point communication among various microservices; 5. *failover manager*, guaranteeing high availability/reliability through replication; 6. troubleshooting utilities, which assist developers with performance profiling/debugging and understanding microservice co-execution interference.

A microservice platform usually provides a number of programming models [10] that developers adhere to, like dataflow and actor-based. The models capture the execution requirements and describe the communication relationship among microservices. For example, the data-flow model (e.g. Amazon Datapipe [6], Google Cloudflow [29], Azure Data

Factory [55]) requires programmers to assemble microservices into a directed acyclic graph (DAG): nodes contain microservices that are interconnected via flow-controlled, lossless dataflow channels. These models bring attractive benefits for a heterogeneous platform since they explicitly express concurrency and communication, enabling the platform to transparently map it to the available hardware [68, 70]. Figure 1 shows an IoT thermostat analytics application [54] consisting of microservices arranged in 3 stages: 1. Thermostat sensor updates are authenticated by the API gateway; 2. Updates are logged into a SQL store sharded by a thermostat identifier; 3. SQL store updates trigger data analytic tasks (e.g, spike detection, moving average, and recommendation) based on thresholds. The dataflow programming model allows the SQL store sharding factor to be dynamically adjusted to scale the application with the number of thermostats reporting. Reliable collections ensure state consistency when re-sharding and the microservice platform automatically migrates and deploys DAG nodes to available hardware resources.

A microservice can be stateful or stateless. Stateless microservices have no persistent storage and only keep state within request context. They are easy to scale, migrate, and replicate, and they usually rely on other microservices for stateful tasks (e.g., a database engine). Stateful microservices use platform APIs to access durable state, allowing the platform full control over data placement. For example, Service Fabric provides reliable collections [40], a collection of data structures that automatically persist mutations. Durable storage is typically disaggregated for microservices and accessed over the network. The use of platform APIs to maintain state allows for fast service migration compared with traditional virtual machine migration [19], as the stateful working set is directly observed by the platform. All microservices in Figure 1 are stateful. We describe further microservices in §4.

2.2 SmartNICs

SmartNICs have appeared on the market [15, 51, 59] and in the datacenter [25]. SmartNICs include computing units, memory, traffic managers, DMA engines, TX/RX ports, and several hardware accelerators for packet processing, such as cryptography and pattern matching engines. Unlike traditional accelerators, SmartNICs integrate the accelerator with the NIC. This allows them to process network requests in-line, at much lower latency than other types of accelerators.

Two kinds of SmartNIC exist: (1) *general-purpose*, which allows transparent microservice offload and is the architecture we consider. For example, Mellanox BlueField [51] has 16 ARMv8 A72 cores with 2×100GE ports and Cavium LiquidIO [15] has 12 cnMIPS cores with 2×10GE ports. These SmartNICs are able to run full operating systems, but also ship with lightweight runtime systems that can provide kernel-bypass access to the NIC's IO engines. (2) FPGA and ASIC based SmartNICs target highly specialized applications. Examples include match-and-action processing [25, 43] for network dataplanes, NPUs [26], and TPUs [39] for deep neural network inference acceleration. FPGAs and ASICs do not support transparent microservice offload. However, they can be combined with general-purpose SmartNICs.

A SmartNIC-accelerated server is a commodity server with one or more SmartNICs. Host and SmartNIC processors do not share thermal, memory, or cache coherence domains, and communicate via DMA engines over PCIe. This allows them to operate as independent, heterogeneous computers, while sharing a power domain and its idle power.

SmartNICs hold promise for improving server energy-efficiency when compared to other heterogeneous computing approaches. For example, racks populated with low-power servers [8] or a heterogeneous mix of servers, suffer from high idle energy draw, as each server requires energy to power its chassis, including fans and devices, and its own ToR switch port. System-on-chip designs with asymmetric performance, such as ARM's big.LITTLE [38] and DynamIQ [2] architectures, and AMD's heterogeneous system architecture (HSA) [7], which combines a GPU with a CPU on the same die, have scalability limits due to the shared thermal design point (TDP). These architectures presently scale to a maximum of 8 cores, making them more applicable to mobile than to server applications. GPGPUs and single instruction multiple threads (SIMT) architectures, such as Intel's Xeon Phi [36] and HP Moonshot [34], are optimized for computational throughput and the extra interconnect hop prevents these accelerators from running latency-sensitive microservices efficiently [57]. SmartNICs are not encumbered by these problems and can thus be used to balance the power draw of latency-sensitive services efficiently.

2.3 Benefits of SmartNIC Offload

We quantify the potential benefit of using SmartNICs for microservices on energy efficiency and request latency. To do so, we choose two identical commodity servers and equip one with a traditional 10GbE Intel X710 NIC and the other with a 10GbE Cavium LiquidIO SmartNIC. Then we evaluate 16 different microservices (detailed in §4) on these two servers with synthetic benchmarks of random 512B requests. We measure request throughput, wall power consumed at peak throughput (defined as the knee of the latency-throughput graph, where queueing delay is minimal) and when idle, as well as client-observed, average/tail request latency in a closed loop. We use host cores on the traditional server and SmartNIC cores on the SmartNIC server for microservice execution. We use as many identical microservice instances, CPUs, and client machines as necessary to attain peak throughput and put unused CPUs to their deepest sleep state. The SmartNIC does not support per-core low power states and always keeps all 12 cores active, diminishing SmartNIC energy efficiency results somewhat. The SmartNIC microservice runtime system uses a kernel-

Microservice	Host (Linux)						Host (DPDK)							SmartNIC					
	RPS	W	C	L	99%	RPJ	RPS	W	C	L	99%	RPJ	%	RPS	W	L	99%	RPJ	×
IPsec	821.3K	117.0	12	1.8	6.6	7.0K	911.9K	112.1	12	1.7	5.2	8.1K	15.9	1851.1K	23.4	0.2	0.8	79.0K	9.7
BM25	91.9K	116.4	12	40.3	205.8	0.8K	99.5K	110.0	12	30.7	155.6	0.9K	14.5	394.1K	19.2	4.1	12.4	20.6K	22.8
NIDS	1781.1K	111.0	12	0.06	0.2	16.1K	1841.1K	106.8	12	0.05	0.15	17.2K	7.4	1988.8K	23.4	0.03	0.1	84.8K	4.9
Recommend	3.6K	109.4	12	86.6	477.0	0.03K	4.1K	111.7	12	78.7	358.6	0.04K	11.6	12.8K	18.9	21.3	123.6	0.7K	18.4
NATv4	1889.6K	72.1	8	0.04	0.1	26.2K	1917.5K	52.1	4	0.04	0.1	36.8K	40.4	2053.1K	23.6	0.03	0.09	86.9K	2.4
Count	1960.8K	68.1	6	0.07	0.1	28.8K	1960.0K	48.6	4	0.03	0.1	40.3K	40.0	2016.8K	21.0	0.03	0.09	96.1K	2.4
EMA	1966.1K	72.7	8	0.04	0.2	27.0K	2009.2K	52.1	4	0.03	0.09	38.6K	42.8	2052.0K	22.0	0.03	0.08	93.5K	2.4
KVS	1946.2K	48.6	8	0.04	0.1	40.0K	2005.0K	33.6	2	0.04	0.1	59.6K	49.0	2033.4K	21.6	0.03	0.1	97.1K	1.6
Flow mon.	1944.1K	70.9	8	0.04	0.1	27.4K	2014.4K	49.8	4	0.03	0.09	40.4K	47.4	2032.6K	24.3	0.03	0.08	83.6K	2.1
DDoS	1989.5K	111.2	12	0.05	0.2	17.9K	1844.8K	105.7	12	0.05	0.2	17.4K	**-3.0**	1952.5K	24.3	0.03	0.1	80.4K	4.6
KNN	42.2K	118.3	12	53.7	163.4	0.4K	42.4K	110.4	12	45.8	161.3	0.4K	7.5	29.9K	20.0	20.6	80.3	1.5K	3.9
Spike	91.9K	112.5	12	29.3	94.5	0.8K	104.3K	112.3	12	25.7	83.0	0.9K	13.7	73.8K	23.5	9.0	50.3	3.1K	3.4
Bayes	12.1K	113.9	12	82.0	406.5	0.1K	13.7K	112.0	12	80.6	400.5	0.1K	14.8	1.6K	19.5	41.9	164.7	0.08K	**0.7**
API gw	1537.6K	108.5	12	0.9	3.2	14.2K	1584.3K	110.6	12	0.8	2.7	14.3K	1.1	124.5K	24.7	8.5	403.6	5.0K	**0.4**
Top ranker	711.9K	119.7	12	4.0	15.0	5.9K	771.9K	109.2	12	3.5	12.3	7.1K	18.9	14.8K	20.3	31.1	154.9	0.7K	**0.1**
SQL	463.3K	114.7	12	6.9	31.1	4.0K	528.0K	113.0	12	6.7	29.5	4.7K	15.7	39.5K	18.8	29.5	104.2	2.1K	**0.4**

Table 1: Microservice comparison among host (Linux and DPDK) and SmartNIC. RPS = Throughput (requests/s), W = Active power (W), C = Number of active cores, L = Average latency (ms), 99% = 99th percentile latency, RPJ = Energy efficiency (requests/Joule).

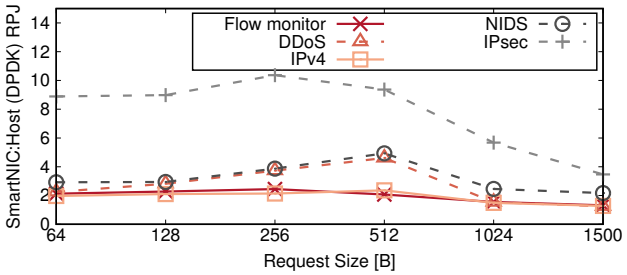


Figure 2: Request size impact on SmartNIC RPJ benefits.

bypass network stack (cf. §4). To break out kernel overheads from the host experiments, we run all microservices on the host in two configurations: 1. Linux kernel network stack; 2. kernel-bypass network stack [63], based on Intel's DPDK [1].

Table 1 presents measured peak request throughput, active power (wall power at peak throughput minus idle wall power), number of active cores, (tail-)latency, and energy efficiency, averaged over 3 runs. Active power allows a direct comparison of host to SmartNIC processor power draw. Energy efficiency equals throughput divided by active power.

Kernel overhead. We first analyze the overhead of in-kernel networking on the host (Linux versus DPDK). As expected, the kernel-bypass networking stack performs better than the in-kernel one. On average, it improves energy efficiency by 21% (% column in Table 1) and reduces tail latency by 16%. Energy efficiency improves because (1) DPDK achieves similar throughput with fewer cores; (2) at peak server CPU utilization, DPDK delivers higher throughput.

SmartNIC performance. SmartNIC execution improves the energy efficiency of 12 of the measured microservices by a geometric mean of 6.5× compared with host execution using kernel bypass (× column in Table 1). The SmartNIC consumes at most 24.7W active power to execute these microservices while the host processor consumes up to 113W. IPSec, BM25, Recommend, and NIDS particularly benefit from various SmartNIC hardware accelerators (crypto coprocessor, fetch-and-add atomic units, floating point engines, and pattern matching units). NATv4, Count, EMA, KVS, Flow monitor, and DDoS can take advantage of the computational bandwidth and fast memory interconnect of the SmartNIC. In these cases, the energy efficiency comes not just from the lower power consumed by the SmartNIC, but also from peak throughput improvements versus the host processor. KNN and Spike attain lower throughput on the SmartNIC. However, since the SmartNIC consumes less power, the overall energy efficiency is still better than the host. For all of these microservices, the SmartNIC also improves client-observed latency. This is due to the hardware accelerated packet buffers and the elimination of PCIe bus traversals. SmartNICs can reduce average and tail latency by a geometric mean of 45.3% and 45.4% versus host execution, respectively.

The host outperforms the SmartNIC for Top ranker, Bayes classifier, SQL, and API gateway by a geometric mean of 4.1× in energy efficiency, 41.2% and 30.0% in average and tail latency reduction. These microservices are branch-heavy with large working sets that are not handled well by the simpler cache hierarchy of the SmartNIC. Moreover, the API gateway uses double floating point numbers for the rate limiter implementation, which the SmartNIC emulates in software.

Request size impact. SmartNIC performance depends also on request size. To demonstrate this, we vary the request size of our synthetic workload and evaluate SmartNIC energy efficiency benefits of 5 microservices versus host execution. Figure 2 shows that with small (≤128B) requests, SmartNIC benefit of IPSec, NIDS, and DDoS is smaller. Small requests are more computation intensive and we are limited by the SmartNIC's wimpy cores. SmartNIC offload hits a sweet-spot at 256–512B request size, where the benefit almost doubles. Here, network and compute bandwidth utilization are balanced for the SmartNIC. At larger request sizes, we are network bandwidth limited, allowing us to put host CPUs to sleep and SmartNIC benefits again diminish. This can be seen in particular for IPsec, which outperforms on the SmartNIC due

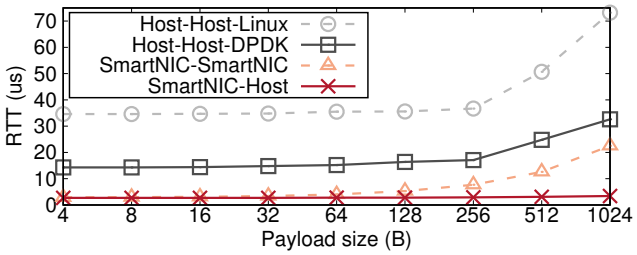


Figure 3: Average RTT (3 runs) of different communication mechanisms in a SmartNIC-accelerated server.

to hardware cryptography acceleration, but still diminishes with larger request sizes. We conclude that request size has a major impact on the benefit of SmartNIC offload. Measuring it is necessary to make good offload choices.

We conclude that SmartNIC offload can provide large energy efficiency and latency benefits for many microservices. However, it is not a panacea. Computation and memory intensive microservices are more suitable to run on the host processor. We need an efficient method to define and monitor critical SmartNIC offload criteria for microservices.

2.4 Challenges of SmartNIC Offload

While there are quantifiable benefits, offloading microservices to SmartNICs brings a number of additional challenges:

- SmartNICs share the same Ethernet MAC address with the host server. Layer 2 switching is not enough to route traffic between SmartNICs and host servers. We require a different switching scheme that can balance traffic and provide fault tolerance when a server equips multiple SmartNICs.
- Microservice platforms assume uniform communication performance among all computing nodes. However, Figure 3 shows that SmartNIC-Host (via PCIe) and SmartNIC-SmartNIC (via ToR switch) communication round-trip-time (RTT) is up to 83.3% and 86.2% lower than host-host (via ToR switch) kernel-bypass communication. We have to consider this topology effect to achieve good performance.
- Microservices share SmartNIC resources and contend with SmartNIC firmware for cache and memory bandwidth. This can create a head-of-line blocking problem for network packet exchange with both SmartNIC and host. Prolonged head-of-line blocking can result in denial of service to unrelated microservices and is more severe than transient sources of interference, such as network congestion. We need to sufficiently isolate SmartNIC-offloaded microservices from firmware to guarantee quality of service.

3 E3 Microservice Platform

We present the E3 microservice platform for SmartNIC-accelerated servers. Our goal is to maximize microservice energy efficiency at scale. Energy efficiency is the ratio of

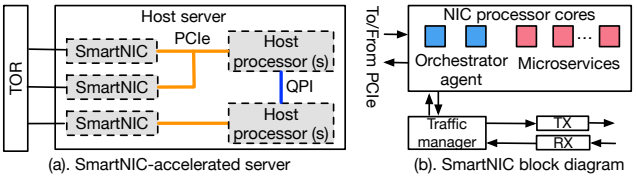


Figure 4: Hardware and software architecture of E3.

microservice throughput and cluster power draw. Power draw is determined by our choice of SmartNIC-acceleration, while E3 focuses on maximizing microservice throughput on this heterogeneous architecture. We describe how we support microservice offload to a SmartNIC and address the request routing, microservice placement, and scheduling challenges.

E3 overview. E3 is a distributed microservice execution platform. We follow the design philosophies of Azure Service Fabric [40] but add energy efficiency as a design requirement. Figure 4 shows the hardware and software architecture of E3. E3 runs in a typical datacenter, where servers are grouped into racks, with a ToR switch per rack. Each server is equipped with one or more SmartNICs, and each SmartNIC is connected to the ToR. This creates a new topology where host processors are reachable via any of the SmartNICs (Figure 4-a). SmartNICs within the same server also have multiple communication options—via the ToR or PCIe (§3.1).

Programming model. E3 uses a dataflow programming model. Programmers assemble microservices into a DAG of microservice nodes interconnected via channels in the direction of RPC flow (cf. Figure 1). A channel provides lossless data communication between nodes. A DAG in E3 describes all RPC and execution paths of a single microservice application, but multiple DAGs may coexist and execute concurrently. E3 is responsible for mapping DAGs to computational nodes.

Software stack. E3 employs a central, replicated cluster resource controller [40] and a microservice runtime on each host and SmartNIC. The resource controller includes four components: (1) traffic control, responsible for routing and load balancing requests between different microservices; (2) control-plane manager, placing microservice instances on cluster nodes; (3) data-plane orchestrator, dynamically migrates microservices across cluster nodes; (4) failover/replication manager, providing failover and node membership management using consistent hashing [75]. The microservice runtime includes an execution engine, an orchestrator agent, and a communication subsystem, described next.

Execution engine. E3 executes each microservice as a multi-threaded process, either on the SmartNIC or on the host. The host runs Linux. The SmartNIC runs a lightweight firmware. Microservices interact only via microservice APIs,

allowing E3 to abstract from the OS. SmartNIC and host support hardware virtual memory for microservice confinement. E3 is work-conserving and runs requests to completion. It leverages a round-robin policy for steering incoming requests to cores, context switching cores if needed.

Orchestrator agent. Each node runs an orchestrator agent to periodically monitor and report runtime execution characteristics to the resource controller. The information is used by (1) the failover manager to determine cluster health and (2) the data-plane orchestrator to monitor the execution performance of each microservice and make migration decisions. On the host, the agent runs as a separate process. On the SmartNIC, the agent runs on dedicated cores (blue in Figure 4-b) and a traffic manager hardware block exchanges packets between the NIC MAC ports and the agent. For each packet, the agent determines the destination (network, host, or SmartNIC core).

3.1 Communication Subsystem

E3 leverages various communication mechanisms, depending on where communicating microservices are located.

Remote communication. When communicating among host cores across different servers, E3 uses the Linux network stack. SmartNIC remote communication uses a user-level network stack [63].

Local SmartNIC-host communication. SmartNIC and host cores on the same server communicate via PCIe. Prior work has extensively explored communication channels via PCIe [47,48,60], and we adopt their design. High-throughput messaging for PCIe interconnects requires leveraging multiple DMA engines in parallel. E3 takes advantage of the eight DMA engines on the LiquidIO, which can concurrently issue scatter/gather requests.

Local SmartNIC-SmartNIC communication. SmartNICs in the same host can use three methods for communication. 1. Using the host to relay requests, involving two data transfers over PCIe and pointer manipulation on the host, increasing latency. 2. PCIe peer-to-peer [23], which is supported on most SmartNICs [15,51,59]. However, the bandwidth of peer-to-peer PCIe communication is capped in a NUMA system when the communication passes between sockets [57]. 3. ToR switch. We take the third approach and our experiments show that this approach incurs lower latency and achieves higher bandwidth than the first two.

3.2 Addressing and Routing

Since SmartNICs and their host servers share Ethernet MAC addresses, we have to use an addressing/routing scheme to distinguish between these entities and load balance across them. For illustration, assume we have a server with two SmartNICs; each NIC has one MAC port. If remote microservices communicate with this server, there will be two possible paths and each might be congested.

We use equal-cost multi-path (ECMP) [84] routing on the ToR switch to route and balance load among these ports. We assign each SmartNIC and the host its own IP. We then configure the ToR switch to route to SmartNICs directly via the attached ToR switch port and an ECMP route to the host IP via any of the ports. The E3 communication subsystem on each SmartNIC differentiates by destination IP address whether an incoming packet is for the SmartNIC or the host. On the host, we take advantage of NIC teaming [56] (also know as port trunking) to bond all related SmartNIC ports into a single logical interface, and then apply the dynamic link aggregation policy (supporting IEEE 802.3ad protocol). ECMP automatically balances connections to the host over all available ports. If a link or SmartNIC fails, ECMP will automatically rebalance new connections via the remaining links, improving host availability.

3.3 Control-plane Manager

The control-plane manager is responsible for energy-efficient microservice placement. This is a computing intensive operation due to the large search space with myriad constraints. Hence, it is done on the control plane. Service Fabric uses simulated annealing, a well-known approximate algorithm, to solve microservice placement. It considers three types of constraints: (1) currently available resources of each computing node (memory, disk, CPU, network bandwidth); (2) computing node runtime statistics (aggregate outstanding microservice requests); (3) individual microservice execution behavior (average request size, request execution time and frequency, diurnal variation, etc.). Service Fabric ignores network topology and favors spreading load over multiple nodes.

E3 extends this algorithm to support bump-in-the-wire SmartNICs, considering network topology. We categorize *computing nodes* (host or SmartNIC processors) into different levels of communication distance and perform a search from the closest to the furthest. We present the HCM algorithm (Algorithm 1). HCM takes as input the microservice DAG G and source nodes V_{src}, as well as the cluster topology T, including runtime statistics for each computing node (as collected). HCM performs a breadth-first traversal of G to map microservices to cluster computing nodes (*MS_DAG_TRAVERSE*).

If not already deployed (*get_deployed_node*), HCM (via *MS_DAG_TRAVERSE*) assigns a microservice V to a computing node N via the *find_first_fit* function (lines 9-11) and deploys it via *set_deployed_node*. *find_first_fit* is a greedy algorithm that returns the first computing node that satisfies the microservice constraints (via its resource and runtime statistics) without considering communication cost. If no such node is found, it returns a node closest to the constraints. Next,

Algorithm 1 HCM microservice placement algorithm

```
1:  G : microservice DAG graph
2:  V_src : source microservice node(s) of the DAG
3:  T : server cluster topology graph
4:  procedure MS_DAG_TRAVERSE(G, V_src, T)
5:      Q.enqueue(V_src)                          ▷ Let Q be a queue
6:      while Q is not empty do
7:          V ← Q.dequeue()
8:          N ← get_deployed_node(V)
9:          if N is NULL then
10:             N ← find_first_fit(V, T)
11:             set_deployed_node(V, N)
12:         for W in all direct descendants of V in G do
13:             N_W ← MS_PLACE(W, N, T)
14:             set_deployed_node(W, N_W)
15:             Q.enqueue(W)
16:
17: V : microservice to place
18: N : computational node of V's ancestor
19: T : server cluster topology graph
20: procedure MS_PLACE(V, N, T)
21:     Topo ← get_hierarchical_topo(N, T)
22:     for L in all Topo.Levels do
23:         N ← find_best_fit(V, Topo.node_list(L))
24:         if N is not NULL then
25:             return N
26:     return find_first_fit(V, T)               ▷ Ignore topology
```

for the descendant microservices of a node V (lines 12-15), HCM assigns them to computing nodes based on their communication distance to V (*MS_PLACE*). To do so, HCM first computes the *hierarchical topology representation* of computing node N via *get_hierarchical_topo*. Each level in the hierarchical topology includes computing nodes that require a similar communication mechanism, starting with the closest. For example, in a single rack there are four levels in this order: 1. The same computing node as V; 2. An adjacent computing node on the same server; 3. A SmartNIC computing node on an adjacent server; 4. A host computing node on an adjacent server. If there are multiple nodes in the same level, HCM uses *find_best_fit* to find the best fit, according to resource constraints. If no node in the hierarchical topology fits the constraints, we fall back to *find_first_fit*.

3.4 Data-plane Orchestrator

The data-plane orchestrator is responsible for detecting load changes and migrating microservices in response to these changes among computational nodes at run-time. To do so, we piggypack several measurements onto the periodic node health reports made by orchestrator agents to the resource controller: This approach is lightweight and integrates well with runtime execution. We believe that our proposed methods can also be used in other microservice schedulers [33, 62, 66].

In this section, we introduce the additional techniques implemented in our data-plane orchestrator to mitigate issues of SmartNIC overload caused by compute-intensive microservices. These can interfere with the SmartNIC's traffic manager, starving the host of network packets. They can also simply execute too slowly on the SmartNIC to be able to catch up with the incoming request rate.

Host starvation. This issue is caused by head-of-line blocking of network traffic due to microservice interference with firmware on SmartNIC memory/cache. It is typically caused by a single compute-intensive microservice overloading the SmartNIC. To alleviate this problem, we monitor the incoming/outgoing network throughput and packet queue depth at the traffic manager. If network bandwidth is under-utilized, but there is a standing queue at the traffic manager, the SmartNIC is overloaded, and we need to migrate microservices.

Microservice overload. This issue is caused by microservices in aggregate requiring more computational bandwidth than the SmartNIC can offer, typically because too many microservices are placed on the same SmartNIC. To detect this problem, we periodically monitor the execution time of each microservice and compare to its exponential moving average. When the difference is negative and larger than 20%, we assume a microservice overload and trigger microservice migration. The threshold was determined empirically.

Microservice migration. For either issue, the orchestrator will migrate the microservice with the highest CPU utilization to the host. To do so, it uses a cold migration approach, similar to other microservice platforms. Specifically, when the orchestrator makes a migration decision, it will first push the microservice binary to the new destination, and then notify the runtime of the old node to (1) remove the microservice instance from the execution engine; (2) clean up and free any local resources; (3) migrate the working state, as represented by reliable collections [40], to the destination. After the orchestrator receives a confirmation from the original node, it will update connections and restart the microservice execution on the new node.

3.5 Failover/Replication Manager

Since SmartNICs share the same power supply as their host server, our failover manager treats all SmartNICs and the host to be in the same fault domain [40], avoiding replica placement within the same. Replication for fault tolerance is typically done across different racks of the same datacenter or across datacenters, and there is no impact from placing SmartNICs in the same failure domain as hosts.

Microservice	S	Description
IPsec		Authenticates (SHA-1) & encrypts (AES-CBC-128) NATv4 [42]
BM25		Search engine ranking function [85], e.g., ElasticSearch
NATv4		IPv4 network address translation using DIR-24-8-BASIC [32]
NIDS		Network intrusion detection w/ aho-corasick parallel match [81]
Count	✓	Item frequency counting based on a bitmap [42]
EMA	✓	Exponential moving average (EMA) for data streams [83]
KVS	✓	Hashtable-based in-memory key-value store [24]
Flow mon.	✓	Flow monitoring system using count-min sketch [42]
DDoS	✓	Entropy-based DDoS detection [58]
Recommend	✓	Recommendation system using collaborative filtering [82]
KNN		Classifier using the K-nearest neighbours algorithm [87]
Spike	✓	Spike detector from a data stream using Z-score [72]
Bayes		Naive Bayes classifier based on *maximum a posteriori* [49]
API gw	✓	API rate limiter and authentication gateway [9]
Top Ranker	✓	Top-K ranker using quicksort [78]
SQL	✓	In-memory SQL database [52]

Table 2: 16 microservices implemented on E3. S = Stateful.

Application	Description	N	Microservices
NFV-FIN	Flow monitoring [42, 64]	72	Flow mon., IPsec, NIDS
NFV-DIN	Intrusion detection [64, 88]	60	DDoS, NATv4, NIDS
NFV-IFID	IPsec gateway [42, 88]	84	NATv4, Flow mon., IPsec, DDoS
RTA-PTC	Twitter analytics [78]	60	Count, Top Ranker, KNN
RTA-SF	Spam filter [35]	96	Spike, Count, KVS, Bayes
RTA-SHM	Server health mon. [37]	84	Count, EMA, SQL, BM25
IOT-DH	IoT data hub [77]	108	API gw, Count, KNN, KVS, SQL
IOT-TS	Thermostat [54]	108	API,EMA,Spike,Recommend,SQL

Table 3: 8 microservice applications. N = # of DAG nodes.

4 Implementation

Host software stack. The E3 resource controller and host runtime are implemented in 1,287 and 3,617 lines of C (LOC), respectively, on Ubuntu 16.04. Communication among co-located microservices uses per-core, multi-producer, single-consumer FIFO queues in shared memory. Our prototype uses UDP for all network communication.

SmartNIC runtime. The E3 SmartNIC runtime is built in 3,885 LOC on top of the Cavium CDK [16], with a user-level network stack. Each microservice runs on a set of non-preemptive hardware threads. Our implementation takes advantage of a number of hardware accelerator libraries. We use (1) a hardware managed memory manager to store the state of each microsevice, (2) the hardware traffic controller for Ethernet MAC packet management, and (3) atomic fetch-and-add units to gather performance statistics. We use page protection of the cnMIPS architecture to confine microservices.

Microservices. We implemented 16 popular microservices on E3, as shown in Table 2, in an aggregate 6,966 LOC. Six of the services are stateless or use read-only state that is modified only via the cluster control plane. The remaining services are stateful and use reliable collections to maintain their state. When running on the SmartNIC, IPsec and API gateway can use the crypto coprocessor (105 LOC), while Recommend and NIDS can take advantage of the deterministic finite automata unit (65 LOC). For Count, EMA, KVS, and Flow monitor, our compiler automatically uses the dedicated atomic fetch-and-add units on the SmartNIC. When performing single-precision floating-point computations (EMA, KNN, Spike, Bayes), our compiler generates FPU code on the SmartNIC. Double-precision floating-point calculations (API gateway) are software emulated. E3 reliable collections currently only support hashtables and arrays, preventing us from migrating the SQL engine. We thus constrain the control-plane manager to pin SQL instances to host processors.

System/Cluster	Cost [$]	BC	WC	Mem	Idle	Peak	Bw
Beefy	4,500	12	0	64	83	201	20
Wimpy	2,209	0	32	2	79	95	20
Type1-SmartNIC	4,650	12	12	68	98	222	20
Type2-SmartNIC	6,750	16	48	144	145	252	40
SuperBeefy	12,550	24	0	192	77	256	80
4×Beefy	18,000	48	0	256	332	804	80
4×Wimpy	8,836	0	128	8	316	380	80
2×B.+2×W.	13,018	24	64	132	324	592	80
2×Type2-SmartNIC	13,500	32	96	288	290	504	80
1×SuperBeefy	12,550	24	0	192	77	256	80

Table 4: Evaluated systems and clusters. BC = Beefy cores, WC = Wimpy cores, Mem = Memory (GB), Idle and Peak power (W), Bw = Network bandwidth (Gb/s).

Applications. Based on these microservices, we develop eight applications across three application domains: (1) Distributed real-time analytics (RTA), such as Apache Storm [78], implemented as a dataflow processing graph of workers that pass data tuples in real time to trigger computations; (2) Network function (NF) virtualization (NFV) [61], which is used to build cloud-scale network middleboxes, software switches, and enterprise IT networks, by chaining NFs; (3) An IoT hub (IOT) [53], which gathers sensor data from edge devices and generates events for further processing (e.g., spike detection, classifier) [**?**, 77]. To maximize throughput, applications may shard and replicate microservices, resulting in a DAG node count larger than the involved microservice types. Table 3 presents the microservice types involved in each application, the deployed DAG node count, and references the workloads used for evaluation. The workloads are trace-based and synthetic benchmarks, validated against realistic scenarios. The average and maximum node fanouts among our applications are 6 and 12, respectively. Figure1 shows IOT-TS as an example. IOT-TS is sharded into 6×API, 12×SQL, 12×EMA, 12×Spike, and 12×recommend and each microservice has one backup replica.

5 Evaluation

Our evaluation aims to answer the following questions:

1. What is the energy efficiency benefit of microservice SmartNIC-offload? Is it proportional to client load? What is the latency cost? (§5.1)
2. Does E3 overcome the challenges of SmartNIC-offload? (§5.2, §5.3, §5.4)
3. Do SmartNIC-accelerated servers provide better total cost of ownership than other cluster architectures? (§5.5)
4. How does E3 perform at scale? (§5.6)

Experimental setup. Our experiments run on a set of clusters (Table 4 presents the server and cluster configurations), attached to an Arista DCS-7050S ToR switch. Beefy is a Supermicro 1U server, with a 12-core E5-2680 v3 processor at 2.5GHz, and a dual-port 10Gbps Intel X710 NIC. Wimpy is ThunderX-like, with a CN6880 processor (32 cnMIPS64 cores running at 1.2GHz), and a dual-port 10Gbps NIC. SuperBeefy is a Supermicro 2U machine, with a 24-core Xeon Platinum 8160 CPU at 2.1GHz, and a dual-port 40Gbps Intel XL710 NIC. Our SmartNIC is the Cavium LiquidIOII [15], with one OCTEON processor with 12 cnMIPS64 cores at 1.2GHz, 4GB memory, and two 10Gbps ports. Based on this, we build two SmartNIC servers: Type1 is Beefy, but swaps the X710 10Gbps NIC with the Cavium LiquidIOII; Type2 is a 2U server with two 8-core Intel E5-2620 processors at 2.1GHz, 128GB memory, and 4 SmartNICs. All servers have a Seagate HDD. We build the clusters such that each has the same amount of aggregate network bandwidth. This allows us to compare energy efficiency based on the compute bandwidth of the clusters, without varying network bandwidth. We also exclude the switch from our cost and energy evaluations, as each cluster uses an identical number of switch ports.

We measure server power consumption using the servers' IPMI data center management interface (DCMI), cross-checked by a Watts Up wall power meter. Throughput and average/tail latency across 3 runs are measured from clients (Beefy machines), of which we provide as many as necessary. We enable hyper-threading and use the `Intel_pstate` governor for power management. All benchmarks in this section report energy efficiency as throughput over server/cluster **wall power** (not just active power).

5.1 Benefit and Cost of SmartNIC-Offload

Peak utilization. We evaluate the latency and energy efficiency of using SmartNICs for microservice applications, compared to homogeneous clusters. We compare 3×Beefy to 3×Type1-SmartNIC, to ensure that microservices also communicate remotely. We focus first on peak utilization, which is desirable for energy efficiency, as it amortizes idle power draw. To do so, we deploy as many instances of each application and apply as much client load as necessary to maximize request throughput without overloading the cluster, as determined by the knee of the latency-throughput curve.

Figure 5 shows that Type1-SmartNIC achieves an average 2.5×, 1.3×, and 1.3× better energy efficiency across the NFV, RTA, and IOT application classes, respectively. This goes along with 43.3%, 92.3%, and 80.4% average latency savings and 35.5%, 90.4%, 88.6% 99th percentile latency savings, respectively. NFV-FIN gains the most—3× better energy efficiency—because E3 is able to run all microservices on the SmartNICs. RTA-PTC benefits the least—12% energy efficiency improvement at 4% average and tail latency cost—as E3 only places the Count microservice on the SmartNIC and migrates the rest to the host.

Power proportionality. This experiment evaluates the power proportionality of E3 (energy efficiency at lower than peak utilization). Using 3×Type1-SmartNIC, we choose an application from each class (NFV-FIN, RTA-SHM, and IOT-TS) and vary the offered request load between idle and peak via a client side request limiter. Figure 8 shows that RTA-SHM and IOT-TS are power proportional. NFV-FIN is not power proportional but also draws negligible power. NFV-FIN runs all microservices on the SmartNICs, which have low active power, but the cnMIPS architecture has no per-core sleep states.

We conclude that applications can benefit from E3's microservice offload to SmartNICs, in particular at peak cluster utilization. Peak cluster utilization is desirable for energy efficiency and microservices make it more common due to light-weight migration. However, transient periods of low load can occur and E3 draws power proportional to request load. We can apply insights from Prekas, et al. [65] to reduce polling overheads and improve power proportionality further.

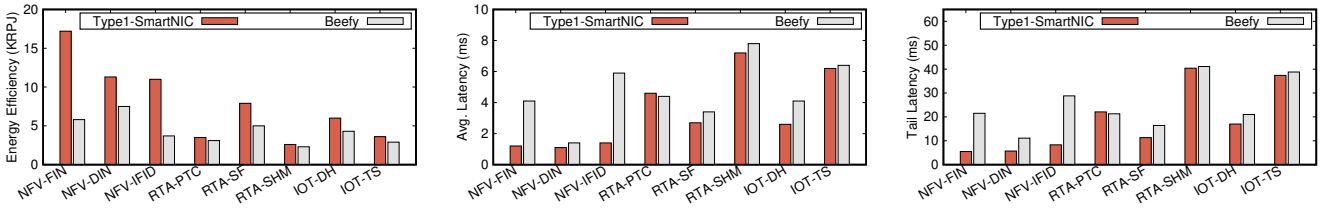
5.2 Avoiding Host Starvation

We show that E3's data-plane orchestrator prevents host starvation by identifying head-of-line blocking of network traffic. To do so, we use 3×Type1-SmartNIC and place as many microservices on the SmartNIC as fit in memory. E3 identifies the microservices that cause interference (Top Ranker in RTA-PTC, Spike in RTA-SF, API gateway in IOT-DH and IOT-TS) and migrates them to the host. As shown in Figure 7, our approach achieves up to 29× better energy efficiency and up to 89% latency reduction across RTA-PTC, RTS-SF, IOT-DH, and IOT-TS. For the other applications, our traffic engine has little effect because the initial microservice assignment already put the memory intensive microservices on the host.

5.3 Sharing SmartNIC and Host Bandwidth

This experiment evaluates the benefits of sharing SmartNIC network bandwidth with the host. We compare two Type2-SmartNIC configurations: 1. Sharing aggregate network bandwidth among host and SmartNICs, using ECMP to balance host traffic over SmartNIC ports; 2. Replacing one SmartNIC with an Intel X710 NIC used exclusively to route traffic to the host. To emphasize the load balancing benefits, we always place the client-facing microservices on the host server. Note that SmartNIC-offloaded microservices still exchange network traffic (when communicating remotely or among SmartNICs) and interfere with host traffic.

Figure 9 shows that load balancing improves application throughput up to 2.9× and cluster energy efficiency up to 2.7× (NFV-FIN). Available host network bandwidth when sharing SmartNICs can be up to 4× that of the dedicated



(a) Energy-efficiency **(b) Average latency** **(c) 99th-percentile latency**

Figure 5: Energy-efficiency, average/tail latency comparison between Type1-SmartNIC and Beefy at peak utilization.

Cluster	NFV-FIN	RTA-SHM	IOT-TS
4×Beefy	5.1	1.9	2.7
4×Wimpy	29.9	0.4	0.1
2×B.+2×W.	8.2	1.4	1.9
2×Type2-SmartNIC	29.0	4.5	6.1
1×SuperBeefy	8.8	2.9	5.0

Table 5: Energy efficiency across five clusters (KRPJ).

NIC, which balances better with the host compute bandwidth. With a dedicated NIC, host processors can starve for network bandwidth. IOT-TS is compute-bound and thus benefits the least from sharing. In terms of latency, all cases behave the same since the request execution flows are the same.

5.4 Communication-aware Placement

To show the effectiveness of communication-aware microservice placement, we evaluate HCM on E3 without data-plane orchestrator. In this case, all microservices are stationary after placement. We avoid host starvation and microservice overload by constraining problematic microservices to the host.

Using 3×Type1-SmartNIC and all placement constraints of Service Fabric [40] (described in §3.3), we compare HCM with both simulated annealing and an integer linear program (ILP). HCM places the highest importance on minimizing microservice communication latency. Simulated annealing and ILP use a cost function with the highest weight on minimizing co-execution interference. Hence, HCM tries to co-schedule communicating microservices on proximate resources, while the others will spread them out. ILP attempts to find the best configuration, while simulated annealing approximates. Figure 6 shows that compared to simulated annealing and ILP, HCM improves energy efficiency by up to 35.2% and 22.0%, and reduces latency by up to 24.0% and 18.6%, respectively. HCM's short communication latency benefits outweigh interference from co-execution.

5.5 Energy Efficiency = Cost Efficiency

While SmartNICs benefit energy efficiency and thus potentially bring cost savings, can they compete with other forms of heterogeneous clusters, especially when factoring in the capital expense to acquire the hardware? In this experiment, we evaluate the cost efficiency, in terms of request throughput over total cost and time of ownership, of using SmartNICs for microservices, compared with four other clusters (see Table 4). Assuming that clusters are usually at peak utilization, we use the cost efficiency metric $\frac{Throughput \times T}{CAPEX + (Power \times T \times Electricity)}$, where $Throughput$ is the measured average throughput at peak utilization for each application, as executed by E3 on each cluster, T is elapsed time, $CAPEX$ is the capital expense to purchase the cluster including all hardware components ($), $Power$ is the elapsed peak power draw of the cluster (Watts), and $Electricity$ is the price of electricity ($/Watts). The cluster cost and power data is shown in Table 4 and we use the average U.S. electricity price [31] of $0.0733/kWh. Figure 10 reports results for three applications of very different points in the workload space, extrapolated over time of ownership by our cost efficiency metric.

We make three observations. First, in the long term (>1 year of ownership), cost efficiency is increasingly dominated by energy efficiency. This highlights the importance of energy efficiency for data center design, where servers are typically replaced after several years to balance CAPEX [12]. Second, when all microservices are able to run on a low power platform (NFV-FIN), both 4×Wimpy and 2×Type2-SmartNIC clusters are the most cost efficient. After 5 years, 4×Wimpy is 14.1% more cost efficient than 2×Type2-SmartNIC because of the lower power draw. Third, when a microservice application contains both compute and IO-intensive microservices (RTA-SHM, IOT-TS), the 2×Type2-SmartNIC cluster is up to 1.9× more cost efficient after 5 years of ownership than the next best cluster configuration (4×Beefy in both cases).

Table 5 presents the measured energy-efficiency, which shows cost efficiency in the limit (over very long time of ownership). We can see that 4×Wimpy is only 3% more energy efficient (but has lower CAPEX) than 2×Type2-SmartNIC for NFV-FIN. 2×Type2-SmartNIC is on average 2.37× more energy-efficient (but has higher CAPEX) than 1×SuperBeefy, which is the second-best cluster in terms of energy-efficiency.

5.6 Performance at Scale

We evaluate and discuss the scalability of E3 along three axes: 1. Mechanism performance scalability; 2. Tail-latency; 3. Energy-efficiency.

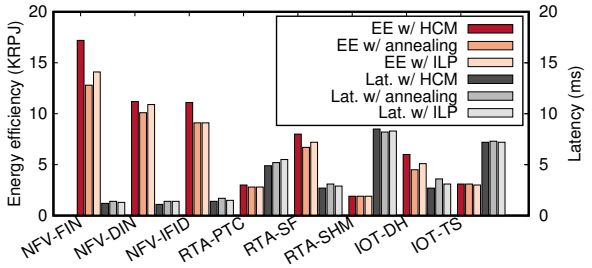


Figure 6: Communication-aware microservice placement.

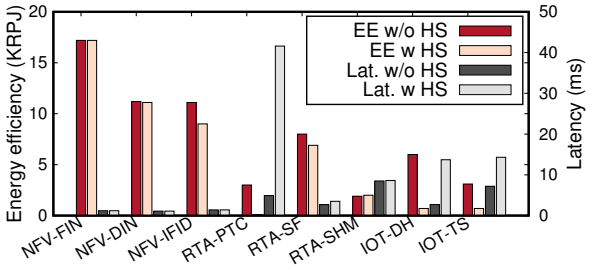


Figure 7: Avoiding host starvation (HS).

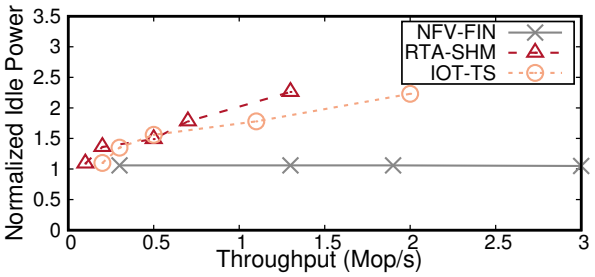


Figure 8: Power draw of 3 applications normalized to idle power of 3×Type1-SmartNIC, varying request load.

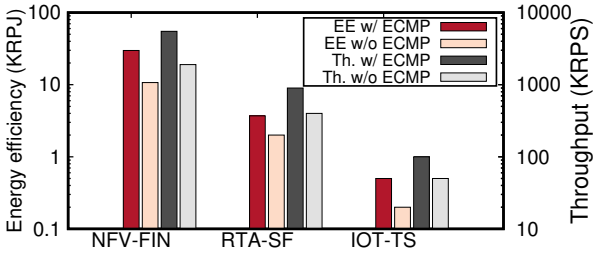
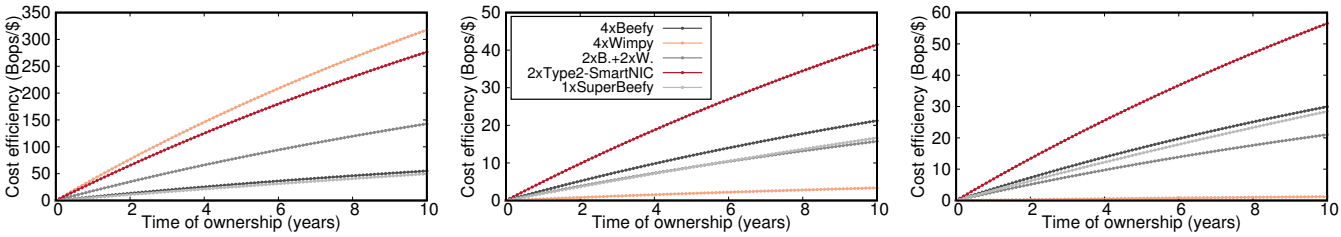


Figure 9: ECMP-based SmartNIC sharing (log y scale).



(a) NFV-FIN **(b) RTA-SHM** **(c) IOT-TS**

Figure 10: Cost efficiency of 3 applications across the cluster configurations from Table 4.

Servers →	100	200	400	600	800	1,000
HCM	4.85	8.31	19.83	34.32	74.39	263.46
Annealing	3.15	4.73	7.43	15.64	23.50	61.42
ILP	7.64	19.43	84.83	361.85	$\gg 1s$	$\gg 1s$

Table 6: Per-microservice deployment time (ms) scalability.

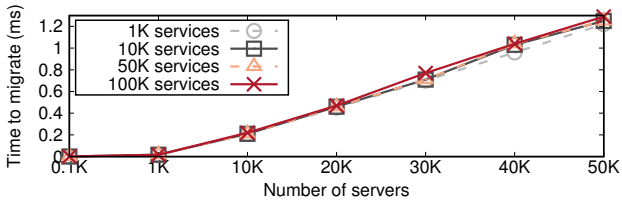


Figure 11: Orchestrator migration decision time scalability.

Mechanism scalability. At scale, pressure on the control-plane manager and data-plane orchestrator increases. We evaluate the performance scalability of both mechanisms with an increasing number of Type2-SmartNIC servers in a simulated FatTree [30] topology with 40 servers per rack. To avoid host starvation and microservice overload, E3's data-plane orchestrator receives one heartbeat message (16B) every 50ms from each SmartNIC that reports the queue length of the traffic manager and the SmartNIC's microservice execution times. The orchestrator parses the heartbeat message and makes a migration decision (§3.4). Figure 11 shows that the time taken to transmit the message and make a decision with a large number of servers stays well below the time taken to migrate the service (on the order of 10s-100s of ms) and is negligibly impacted by the number of deployed microservices. This is because the heartbeat message contributes only 1Kbps of traffic, even with 50K servers.

E3 uses HCM in the control-plane manager. We compare it to simulated annealing and ILP, deploying 10K microservices on an increasing number of servers. Table 6 shows that while HCM does not scale as well as simulated annealing, it can deploy new microservices in a reasonable time span (<<1s) at scale. ILP fails to deliver acceptable performance.

Tail latencies. At scale, tail latencies dominate [20]. While SmartNICs can introduce high tail latency for some microservices (§2.3), E3 places these microservices on the host to ensure that application-level tail-latency inflation is minimized

(§5.1). The tail-latency impact of SmartNIC offload is reduced at scale, as baseline communication latency increases with increasing inter-rack distance.

Energy-efficiency and power budgets. E3's energy efficiency benefits are constant regardless of deployment size and power budgets. At scale, there is additional energy cost for core and spine switches, but these are negligible compared to racks (ToRs and servers). Within a rack, ToR switch energy consumption stays constant, as all compared systems use the same number of switch ports. Our results show that E3 achieves up to 1.9x more throughput under the same power budget. Conversely, operators can save 48% of power, offering the same bandwidth.

6 Related Work

Architecture studies for microservices. Prior work has explored the architectural implications of running microservices [21, 27, 79]. It shows that wimpy servers hold potential for microservices under low loads, as they have less cache utilization and network virtualization overhead compared with traditional monolithic cloud workloads. FAWN [8] explored using only low-power processor architecture for data-intensive workloads as an energy and cost-effective alternative to server multiprocessors. However, FAWN assumed that I/O speeds are much lower than CPU speeds and so CPUs would be left idle for data-intensive applications. With the advent of fast network technologies, server CPUs are still required. Motivated by these studies, we focus on using SmartNIC-accelerated servers for energy efficiency.

Heterogeneous scheduling. A set of schedulers for performance-asymmetric architectures have been proposed. For example, Kumar et al. [44, 45] use instructions per cycle to determine relative speedup of each thread on different types of cores. HASS [73] introduces the architectural signature concept as a scheduling indicator, which contains information about memory-boundedness, available instruction-level parallelism, etc. CAMP [71] combines both efficiency and thread-level parallelism specialization and proposes a light-weight technique to discover which threads could use fast cores more efficiently. PTask [69] provides a data-flow programming model for programmers to manage computation for GPUs. It enables sharing GPUs among multiple processes, parallelizes multiple tasks, and eliminates unnecessary data movements. These approaches target long-running computations, mostly on cache coherent architectures, rather than microsecond-scale, request-based workloads over compute nodes that do not share memory and are hence not applicable.

Microservice scheduling. Wisp [76] enforces end-to-end performance objectives by globally adapting rate limiters and request schedulers based on operator policies under varying system conditions. This work is not concerned with SmartNIC heterogeneity. UNO [46] is an NFV framework that can systematically place NFs across SmartNIC and host with a resource-aware algorithm on the control plane. E3 is a microservice platform and thus goes several steps further: (1) E3 uses a data-plane orchestrator to detect node load and migrates microservices if necessary; (2) HCM considers communication distance during the placement. With the advent of SmartNICs and programmable switches, researchers have identified the potential performance benefits of applying request processing across the communication path [14]. E3 is such a system designed for the programmable cloud and explores the energy efficiency benefits of running microservices.

Power proportionality. Power proportional systems can vary energy use with the presented workload [50]. For example, Prekas, et al. propose an energy-proportional system management policy for memcached [65]. While E3 can provide energy proportionality, we are primarily interested in energy-efficiency. Geoffrey et al. [18] propose a heterogeneous power-proportional system. By carefully selecting component ensembles, it can provide an energy-efficient solution for a particular task. However, due to the high cost of ensemble transitions, we believe that this architecture is not fit for high bandwidth I/O systems. Rivoire, et al. propose a more balanced system design (for example, a low-power, mobile processor with numerous laptop disks connected via PCIe) and show that it achieves better energy efficiency for sorting large data volumes [67]. Pelican [11] presents a software storage stack on under-provisioned hardware targeted at cold storage workloads. Our proposal could be viewed as a balanced-energy approach for low-latency query-intensive server applications, rather than cold, throughput intensive ones.

7 Conclusion

We present E3, a microservice execution platform on SmartNIC-accelerated servers. E3 extends key system components (programming model, execution engine, communication subsystem, scheduling) of the Azure Service Fabric microservice platform to a SmartNIC. E3 demonstrates that SmartNIC offload can improve cluster energy-efficiency up to 3× and cost efficiency up to 1.9× at up to 4% latency cost for common microservices.

Acknowledgments

This work is supported in part by NSF grant 1751231, 1616774, 1714508, and the Texas Systems Research Consortium. We would like to thank the anonymous reviewers and our shepherd, Ada Gavrilovska, for their comments and feedback.

References

[1] DPDK. https://www.dpdk.org/.

[2] DynamIQ. https://developer.arm.com/technologies/dynamiq.

[3] Amazon Lambda Serverless Computing Platform. https://aws.amazon.com/lambda/, 2018.

[4] Google App Engine. https://cloud.google.com/appengine/, 2018.

[5] Nirmata Platform. https://www.nirmata.com/, 2018.

[6] Amazon. Amazon Data Pipeline. https://aws.amazon.com/datapipeline/, 2018.

[7] AMD. AMD HSA. https://www.amd.com/en-us/innovations/software-technologies/hsa, 2018.

[8] David G. Andersen, Jason Franklin, Michael Kaminsky, Amar Phanishayee, Lawrence Tan, and Vijay Vasudevan. FAWN: A Fast Array of Wimpy Nodes. In *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*, 2009.

[9] Microservice Architecture. Api gateway. http://microservices.io/patterns/apigateway.html, 2018.

[10] Microsoft Azure. Programming Model in the Azure Service Fabric. https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-choose-framework, 2018.

[11] Shobana Balakrishnan, Richard Black, Austin Donnelly, Paul England, Adam Glass, Dave Harper, Sergey Legtchenko, Aaron Ogus, Eric Peterson, and Antony Rowstron. Pelican: A Building Block for Exascale Cold Data Storage. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, 2014.

[12] Luiz André Barroso, Urs Hölzle, and Parthasarathy Ranganathan. The Datacenter as a Computer: Designing Warehouse-Scale Machines. *Synthesis Lectures on Computer Architecture*, 2018.

[13] Broadcom. Broadcom Stingray SmartNICs. https://www.broadcom.com/products/ethernet-connectivity/smartnic/ps225, 2018.

[14] Adrian Caulfield, Paolo Costa, and Monia Ghobadi. Beyond SmartNICs: Towards a fully programmable cloud. In *IEEE International Conference on High Performance Switching and Routing, ser. HPSR*, volume 18, 2018.

[15] Cavium. Cavium LiquidIO SmartNICs. https://cavium.com/pdfFiles/LiquidIO_II_CN78XX_Product_Brief-Rev1.pdf, 2018.

[16] Cavium. Cavium OCTEON Development Kits. https://cavium.com/octeon-software-develop-kit.html, 2018.

[17] Luis Ceze, Mark D Hill, and Thomas F Wenisch. Arch2030: A vision of computer architecture research over the next 15 years. *arXiv preprint arXiv:1612.03182*, 2016.

[18] Geoffrey Challen and Mark Hempstead. The Case for Power-agile Computing. In *Proceedings of the 13th USENIX Conference on Hot Topics in Operating Systems*, 2011.

[19] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation-Volume 2*, 2005.

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

[21] Christina Delimitrou. The Hardware-Software Implications of Microservices and How Big Data Can Help. 2018.

[22] A. Shehabi et al. United States Data Center Energy Usage Report. Technical report, Lawrence Berkeley National Laboratory, 2016.

[23] Dolphin Express. Remote Peer to Peer made easy. https://www.dolphinics.com/download/WHITEPAPERS/Dolphin_Express_IX_Peer_to_Peer_whitepaper.pdf, 2018.

[24] Bin Fan, David G. Andersen, and Michael Kaminsky. MemC3: Compact and Concurrent MemCache with Dumber Caching and Smarter Hashing. In *10th USENIX Symposium on Networked Systems Design and Implementation*, 2013.

[25] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, Harish Kumar Chandrappa, Somesh Chaturmohta, Matt Humphrey, Jack Lavier, Norman Lam, Fengfen Liu, Kalin Ovtcharov, Jitu Padhye, Gautham Popuri, Shachar Raindel, Tejas Sapre, Mark Shaw, Gabriel Silva, Madhan Sivakumar, Nisheeth Srivastava, Anshuman Verma, Qasim Zuhair, Deepak Bansal, Doug Burger, Kushagra Vaid, David A. Maltz, and Albert Greenberg.

Azure Accelerated Networking: SmartNICs in the Public Cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation*, 2018.

[26] J. Fowers, K. Ovtcharov, M. Papamichael, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, L. Adams, M. Ghandi, S. Heil, P. Patel, A. Sapek, G. Weisz, L. Woods, S. Lanka, S. K. Reinhardt, A. M. Caulfield, E. S. Chung, and D. Burger. A Configurable Cloud-Scale DNN Processor for Real-Time AI. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture*, 2018.

[27] Y. Gan and C. Delimitrou. The Architectural Implications of Cloud Microservices. *IEEE Computer Architecture Letters*, 17(2):155–158, 2018.

[28] Yu Gan, Yanqi Zhang, Dailun Cheng, Ankitha Shetty, Priyal Rathi, Nayan Katarki, Ariana Bruno, Justin Hu, Brian Ritchken, Brendon Jackson, Kelvin Hu, Meghna Pancholi, Yuan He, Brett Clancy, Chris Colen, Fukang Wen, Catherine Leung, Siyuan Wang, Leon Zaruvinsky, Mateo Espinosa, Rick Lin, Zhongling Liu, Jake Padilla, and Christina Delimitrou. An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud & Edge Systems. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019.

[29] Google. Google Cloud Dataflow. https://cloud.google.com/dataflow/, 2018.

[30] Albert Greenberg, James R Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A Maltz, Parveen Patel, and Sudipta Sengupta. VL2: a scalable and flexible data center network. In *ACM SIGCOMM computer communication review*, volume 54, pages 95–104, 2009.

[31] Site Selection Group. Power in the Data Center and its Cost Across the U.S. https://info.siteselectiongroup.com/blog/power-in-the-data-center-and-its-costs-across-the-united-states, 2017.

[32] Pankaj Gupta, Steven Lin, and Nick McKeown. Routing lookups in hardware at memory access speeds. In *INFOCOM'98. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, 1998.

[33] Md E. Haque, Yuxiong He, Sameh Elnikety, Thu D. Nguyen, Ricardo Bianchini, and Kathryn S. McKinley. Exploiting Heterogeneity for Tail Latency and Energy Efficiency. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, 2017.

[34] HP. HPE ProLiant m800 Server Cartridge. https://support.hpe.com/hpsc/doc/public/display?docId=emr_na-c04500667&sp4ts.oid=6532018, 2018.

[35] Thomas Hunter II. *Advanced Microservices: A Hands-on Approach to Microservice Infrastructure and Tooling*. 2017.

[36] Intel. Intel Xeon Phi Coprocessor 7120A. https://ark.intel.com/products/80555/Intel-Xeon-Phi-Coprocessor-7120A-16GB-1238-GHz-61-core, 2018.

[37] Rajkumar Jalan, Swaminathan Sankar, and Gurudeep Kamat. Distributed system to determine a server's health, 2018. US Patent 9,906,422.

[38] Brian Jeff. Ten Things to Know About big. LITTLE. *ARM Holdings*, 2013.

[39] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, 2017.

[40] Gopal Kakivaya, Lu Xun, Richard Hasha, Shegufta Bakht Ahsan, Todd Pfleiger, Rishi Sinha, Anurag Gupta, Mihail Tarta, Mark Fussell, Vipul Modi, et al. Service fabric: a distributed platform for building microservices in the cloud. In *Proceedings of the Thirteenth EuroSys Conference*, 2018.

[41] Svilen Kanev, Juan Pablo Darago, Kim Hazelwood, Parthasarathy Ranganathan, Tipp Moseley, Gu-Yeon Wei, and David Brooks. Profiling a Warehouse-scale Computer. In *Proceedings of the 42Nd Annual International Symposium on Computer Architecture*, 2015.

[42] Georgios P. Katsikas, Tom Barbette, Dejan Kostić, Rebecca Steinert, and Gerald Q. Maguire Jr. Metron: NFV Service Chains at the True Speed of the Underlying Hardware. In *15th USENIX Symposium on Networked Systems Design and Implementation*, 2018.

[43] 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*, 2016.

[44] Rakesh Kumar, Keith I Farkas, Norman P Jouppi, Parthasarathy Ranganathan, and Dean M Tullsen. Single-ISA heterogeneous multi-core architectures: The

potential for processor power reduction. In *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on*, 2003.

[45] Rakesh Kumar, Dean M Tullsen, Parthasarathy Ranganathan, Norman P Jouppi, and Keith I Farkas. Single-ISA heterogeneous multi-core architectures for multithreaded workload performance. In *Computer Architecture, 2004. Proceedings. 31st Annual International Symposium on*, 2004.

[46] Yanfang Le, Hyunseok Chang, Sarit Mukherjee, Limin Wang, Aditya Akella, Michael M Swift, and TV Lakshman. UNO: uniflying host and smart NIC offload for flexible packet processing. In *Proceedings of the 2017 Symposium on Cloud Computing*, 2017.

[47] Bojie Li, Kun Tan, Layong Larry Luo, Yanqing Peng, Renqian Luo, Ningyi Xu, Yongqiang Xiong, Peng Cheng, and Enhong Chen. Clicknp: Highly flexible and high performance network processing with reconfigurable hardware. In *Proceedings of the 2016 ACM SIGCOMM Conference*, 2016.

[48] Felix Xiaozhu Lin and Xu Liu. Memif: Towards Programming Heterogeneous Memory Asynchronously. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, 2016.

[49] Jianxiao Liu, Zonglin Tian, Panbiao Liu, Jiawei Jiang, and Zhao Li. An approach of semantic web service classification based on Naive Bayes. In *Services Computing, 2016 IEEE International Conference on*, 2016.

[50] David Lo, Liqun Cheng, Rama Govindaraju, Luiz André Barroso, and Christos Kozyrakis. Towards energy proportionality for large-scale latency-critical workloads. In *Computer Architecture, 2014 ACM/IEEE 41st International Symposium on*, 2014.

[51] Mellanox. Mellanox BlueField Platforms. http://www.mellanox.com/related-docs/npu-multicore-processors/PB_BlueField_Ref_Platform.pdf, 2018.

[52] MemSQL. In-memory SQL database. https://www.memsql.com, 2018.

[53] Microsoft. Azure IoT hub. https://azure.microsoft.com/en-us/services/iot-hub/, 2018.

[54] Microsoft. Honeywell Case Study. https://blogs.msdn.microsoft.com/azureservicefabric/2018/03/20/service-fabric-customer-profile-honeywell/, 2018.

[55] Microsoft. Microsoft Data Factory. https://azure.microsoft.com/en-us/services/data-factory/, 2018.

[56] Microsoft. NIC Teaming. https://docs.microsoft.com/en-us/windows-server/networking/technologies/nic-teaming/nic-teaming, 2018.

[57] Changwoo Min, Woonhak Kang, Mohan Kumar, Sanidhya Kashyap, Steffen Maass, Heeseung Jo, and Taesoo Kim. Solros: A Data-centric Operating System Architecture for Heterogeneous Computing. In *Proceedings of the Thirteenth EuroSys Conference*, 2018.

[58] Jelena Mirkovic and Peter Reiher. A taxonomy of DDoS attack and DDoS defense mechanisms. *ACM SIGCOMM Computer Communication Review*, 34(2):39–53, 2004.

[59] Netronome. Netronome Agilio SmartNIC. https://www.netronome.com/products/agilio-cx/, 2018.

[60] Rolf Neugebauer, Gianni Antichi, José Fernando Zazo, Yury Audzevich, Sergio López-Buedo, and Andrew W. Moore. Understanding pcie performance for end host networking. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, 2018.

[61] Shoumik Palkar, Chang Lan, Sangjin Han, Keon Jang, Aurojit Panda, Sylvia Ratnasamy, Luigi Rizzo, and Scott Shenker. E2: A Framework for NFV Applications. In *Proceedings of the 25th Symposium on Operating Systems Principles*, 2015.

[62] Jun Woo Park, Alexey Tumanov, Angela Jiang, Michael A. Kozuch, and Gregory R. Ganger. 3Sigma: Distribution-based Cluster Scheduling for Runtime Uncertainty. In *Proceedings of the Thirteenth EuroSys Conference*, 2018.

[63] 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 *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, 2014.

[64] Rishabh Poddar, Chang Lan, Raluca Ada Popa, and Sylvia Ratnasamy. SafeBricks: Shielding Network Functions in the Cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation*, 2018.

[65] 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*, 2015.

[66] Hang Qu, Omid Mashayekhi, Chinmayee Shah, and Philip Levis. Decoupling the Control Plane from Program Control Flow for Flexibility and Performance in Cloud Computing. In *Proceedings of the Thirteenth EuroSys Conference*, 2018.

[67] Suzanne Rivoire, Mehul A. Shah, Parthasarathy Ranganathan, and Christos Kozyrakis. JouleSort: A Balanced Energy-efficiency Benchmark. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, 2007.

[68] Christopher J Rossbach, Jon Currey, Mark Silberstein, Baishakhi Ray, and Emmett Witchel. PTask: operating system abstractions to manage GPUs as compute devices. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, 2011.

[69] Christopher J. Rossbach, Jon Currey, Mark Silberstein, Baishakhi Ray, and Emmett Witchel. PTask: Operating System Abstractions to Manage GPUs As Compute Devices. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, 2011.

[70] Christopher J Rossbach, Yuan Yu, Jon Currey, Jean-Philippe Martin, and Dennis Fetterly. Dandelion: a compiler and runtime for heterogeneous systems. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, 2013.

[71] Juan Carlos Saez, Manuel Prieto, Alexandra Fedorova, and Sergey Blagodurov. A comprehensive scheduler for asymmetric multicore systems. In *Proceedings of the 5th European conference on Computer systems*, 2010.

[72] Felix Scholkmann, Jens Boss, and Martin Wolf. An efficient algorithm for automatic peak detection in noisy periodic and quasi-periodic signals. *Algorithms*, 5(4):588–603, 2012.

[73] Daniel Shelepov, Juan Carlos Saez Alcaide, Stacey Jeffery, Alexandra Fedorova, Nestor Perez, Zhi Feng Huang, Sergey Blagodurov, and Viren Kumar. HASS: a scheduler for heterogeneous multicore systems. *ACM SIGOPS Operating Systems Review*, 43(2):66–75, 2009.

[74] SPEC. Trends in Server Efficiency and Power Usage in Data Centers. https://www.spec.org/events/beijing2016/slides/015-Trends_in_Server_Efficiency_and_Power_Usage_in_Data_Centers%20-%20Sanjay%20Sharma.pdf, 2016.

[75] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, 2001.

[76] Lalith Suresh, Peter Bodik, Ishai Menache, Marco Canini, and Florin Ciucu. Distributed resource management across process boundaries. In *Proceedings of the 2017 Symposium on Cloud Computing*, 2017.

[77] Mesh Systems. Mesh Systems. http://www.mesh-systems.com, 2018.

[78] Ankit Toshniwal, Siddarth Taneja, Amit Shukla, Karthik Ramasamy, Jignesh M Patel, Sanjeev Kulkarni, Jason Jackson, Krishna Gade, Maosong Fu, Jake Donham, et al. Storm@ twitter. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, 2014.

[79] Takanori Ueda, Takuya Nakaike, and Moriyoshi Ohara. Workload characterization for microservices. In *2016 IEEE international symposium on workload characterization (IISWC)*, 2016.

[80] Leendert van Doorn. Microsoft's datacenters. In *Proceedings of the 1st Workshop on Hot Topics in Data Centers*, 2016.

[81] Wikipedia. Aho-Corasick Algorithm. https://en.wikipedia.org/wiki/Aho%E2%80%93Corasick_algorithm, 2018.

[82] Wikipedia. Collaborative filtering. https://en.wikipedia.org/wiki/Collaborative_filtering, 2018.

[83] Wikipedia. Ema. https://en.wikipedia.org/wiki/Moving_average, 2018.

[84] Wikipedia. Equal-cost multi-path routing. https://en.wikipedia.org/wiki/Equal-cost_multi-path_routing, 2018.

[85] Wikipedia. Okapi BM25. https://en.wikipedia.org/wiki/Okapi_BM25, 2018.

[86] Wikipedia. Representational State Transfer Architecture. https://en.wikipedia.org/wiki/Representational_state_transfer, 2018.

[87] Wikipedia. k-nearest neighbors algorithm. https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm, 2019.

[88] Kai Zhang, Bingsheng He, Jiayu Hu, Zeke Wang, Bei Hua, Jiayi Meng, and Lishan Yang. G-NET: Effective GPU Sharing in NFV Systems. In *15th USENIX Symposium on Networked Systems Design and Implementation*, 2018.