IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received June 16, 2020, accepted July 2, 2020, date of publication July 7, 2020, date of current version July 20, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3007747

FENCE: Fast, ExteNsible, and ConsolidatEd Framework for Intelligent Big Data Processing

RAMNEEK [1], (Associate Member, IEEE), SEUNG-JUN CHA [2], SANGHEON PACK [1], (Senior Member, IEEE), SEUNG HYUB JEON [2], YEON JEONG JEONG [2], JIN MEE KIM [2], AND SUNGIN JUNG [2]

[1]School of Electrical Engineering, Korea University, Seoul 02841, South Korea
[2]Electronics and Telecommunications Research Institute (ETRI), Daejeon 34129, South Korea

Corresponding author: Sangheon Pack (shpack@korea.ac.kr)

This research was supported in part by National Research Foundation (NRF) Grant (No. 2020R1A2C3006786), and in part by Institute of Information and communications Technology Promotion (IITP) Grant (No. 2014-3-00035 and IITP-2019-2017-0-01633).

ABSTRACT The proliferation of smart devices and the advancement of data-intensive services has led to explosion of data, which uncovers massive opportunities as well as challenges related to real-time analysis of big data streams. The edge computing frameworks implemented over manycore systems can be considered as a promising solution to address these challenges. However, in spite of the availability of modern computing systems with a large number of processing cores and high memory capacity, the performance and scalability of manycore systems can be limited by the software and operating system (OS) level bottlenecks. In this work, we focus on these challenges, and discuss how accelerated communication, efficient caching, and high performance computation can be provisioned over manycore systems. The proposed Fast, ExteNsible, and ConsolidatEd (FENCE) framework leverages the availability of a large number of computing cores and overcomes the OS level bottlenecks to provide high performance and scalability for intelligent big data processing. We implemented a prototype of FENCE and the experiment results demonstrate that FENCE provides improved data reception throughput, read/write throughput, and application processing performance as compared to the baseline Linux system.

INDEX TERMS Manycore systems, edge computing, stream analytics, big data, IoT.

I. INTRODUCTION

Recently, the amount of data being generated by users, applications and devices is increasing at a staggering rate, and 90 percent of the data in the world was generated in the last few years [1]. The pace of data growth is further accelerated by penetration of internet of things (IoT) into various industrial sectors and mission critical applications, such as manufacturing, agriculture, transport, healthcare, and others. Such emergence of industrial IoT (IIoT) is driven by its ability of providing improved operational efficiency, as well as increased revenue by creating new business models, and making faster business decisions by means of machine-to-machine (M2M) communications with real-time big data analytics [2]. For instance, IIoT can be applied to manufacturing for automation and streamlining of system operations, for asset tracking and supply chain monitoring, for predictive analysis to reduce downtime of machinery, for improved safety and security by real-time monitoring and control, for improved energy efficiency, and for improved operational control and training using augmented reality (AR) and virtual reality (VR) [3].

The widespread adoption of such applications has led to a substantial increase in the number of M2M communication connections, resulting in an exponential increase in mobile data traffic. According to Cisco Visual Networking Index (VNI), although a number of M2M connections are expected to increase only 2.4 folds from 2017 to 2022, the mobile data traffic generated by these connections is expected to grow more than seven folds in the same period [4]. The main reason why the amount of data traffic is outgrowing the number of connections is the increased adoption of

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

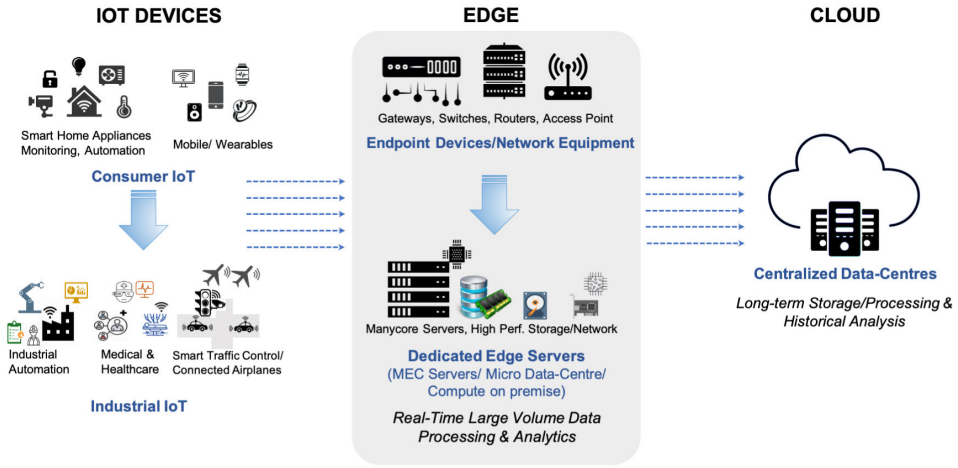


FIGURE 1. Evolution of edge computing and its role in satisfying stringent QoS requirements of advanced IoT applications.

data intensive applications requiring high bandwidth and low latency.

In order to derive useful knowledge from staggering volumes of streaming data, real-time analytics is essential. Big data analysis requires collection, storage, and processing of data in real-time [5]. The gathered data is loaded on to the storage platforms or processed directly using the knowledge discovery platforms that extract useful information from raw data. The analysis stage involves the processing of big data and may involve tagging, classification or categorization of data to convert unstructured data into structured data sets for further processing. The final stage involves application-specific processing and analysis of the classified or categorized data.

In the past, mobile applications and services have been upheld by centralized cloud resources for performing computationally intensive tasks. However, emerging IoT applications impose significant challenges for current computing architectures [6]. Such applications are resource-hungry and data-intensive and require real-time analysis, context and location awareness, increased security, and bandwidth efficiency. These requirements can partially be fulfilled by employing scalable network infrastructures including advanced network equipment, intelligent networking applications, and advanced radio access technologies. However, these applications involve unprecedented levels of data that needs to be collected and processed in real-time.

As IoT evolves from consumers to industrial sectors, the growing size and complexity of data make its real-time collection, processing, and analysis even more challenging. This has led to the evolution of edge computing to support the requirements of critical IIoT applications, as illustrated in Figure 1. Stringent requirements of such applications require the use of scalable manycore edge frameworks with high performance storage and network capabilities, as compared to lightweight IoT applications (generating small amount of data intermittently) requiring simplified edge functionality that can be implemented using endpoint devices or network equipment such as gateways or switches. Low latency requirements of critical applications can be satisfied by performing data analysis and intelligent application processing at the edge i.e., at multi-access edge computing (MEC) servers, micro-data centers or on-premise servers, while historical analysis can be performed at the core [7]. As a result, there is a need of powerful edge computing infrastructure to support the stringent requirements of critical applications.

Although edge computing offers groundbreaking solution for providing low latency by placing key processing tasks closer to the end user, its performance can be limited by underlying operating system (OS) and system level bottlenecks in the edge host. As the processor technology is advancing towards exa-scale computing era, the processors have evolved from multicore to manycore systems. Despite the fact that computing systems have become more and more powerful with the increase in computing processor cores, their performance and scalability are limited by underlying operating environments such as monolithic OSs. Such OSs have been developed and tuned only for a limited number of homogeneous processors with a coherent shared memory [8]. However, their design principles such as shared memory, locking, and caching mechanisms, cause performance bottlenecks that become difficult to overcome as the number of cores increase.

Most of the solutions to overcome these performance limitations are user-space approaches based on kernel bypassing, especially for the applications involving file system input/output (I/O) and network intensive workloads [9]. For instance, the universal kernel bypass suite by solarflare uses non-volatile memory express (NVMe)-over-TCP and allows the acceleration of network and storage server traffic. In addition, data plane development kit (DPDK) [10] and remote direct memory access (RDMA) are popular kernel bypass

solutions for accelerated network performance [11]. However, these user space solutions have numerous issues related to security, flexibility, robustness, inter-operability, and hardware vendor lock-in. Hence, improved OS techniques need to be investigated for high-performance and flexible edge computing platforms.

To enhance the edge caching capabilities, several machine learning (e.g., clustering, reinforcement learning, and similarity learning) based edge caching mechanisms have also been explored [12]. However, such schemes may not be suitable while caching unrelated information from heterogeneous domains.

Moreover, most of the commercial edge computing solutions are tailored for specific use-cases. Thus, they lack flexibility, and their performance can be disrupted by the influx of huge volume of time-sensitive data. As a remarkable open edge computing platform, mobile central office re-architected as a datacenter (m-CORD) leverages a number of open source projects such as docker, OpenStack, and open network operating system (ONOS) to provide built-in service capabilities on the commodity hardware that is managed and coordinated by XOS [13]. These platforms facilitate the provision of unified networking and cloud services by the service providers. However, to reap full benefits of these frameworks, there is a need to optimize underlying infrastructure for provisioning high performance, scalable, and flexible hardware and virtualization framework capable of adapting to evolving IIoT requirements.

In this paper, we introduce the FENCE framework which is (i) fast, as it provides high performance speedup and low latency on manycore systems; (ii) extensible, as it allows seamless inclusion of heterogeneous resources as required; and (iii) provides a consolidated framework for accelerated communications, efficient caching, and high performance computation. We present an illustrative use-case to demonstrate how edge computing requirements can be satisfied using those technologies. The presented evaluation results corroborate the enhanced performance and scalability achieved using the FENCE framework. The main contributions of this paper can be summarized as follows: 1) we analyzed the major performance and scalability bottlenecks in monolithic OS such as Linux; 2) we discussed several key enabling technologies that can be used to overcome the performance bottlenecks and provide optimized performance and scalability; 3) we introduced the FENCE framework for achieving high performance and scalability for intelligent big data processing over manycore systems; and 4) we also presented an evolving graph processing use-case to demonstrate the performance of FENCE.

The rest of paper is organized as follows: Section II presents the motivation of FENCE; Section III describes the details of FENCE framework and its key enabling technologies; Section IV demonstrates the performance of FENCE in terms of data reception, storage, and application processing performance, followed by Section V that concludes the paper.

II. BACKGROUND AND MOTIVATION

The edge computing technology is critical to enable advanced applications such as streaming data analytics, active monitoring and control, location tracking, immersive applications, and real-time content delivery. Such applications involve a huge volume of data and have strict Quality of Service (QoS) requirements. For instance, streaming data analytics requires massive amount of data to be collected and processed in real-time to extract meaningful insights. Moreover, since data may be generated by different types of sensors at different times, it is important to pre-process and arrange the data in temporal order [7]. Implementing such analytics at the edge allows to achieve a higher level of correctness and timeliness as compared to traditional business analytics.

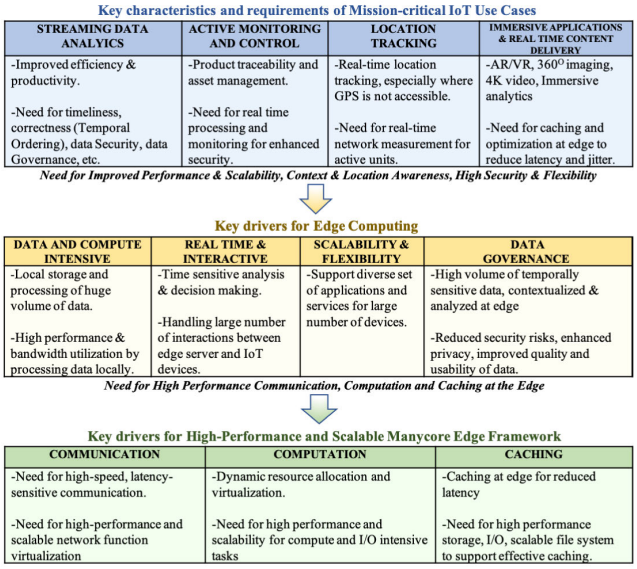


FIGURE 2. Key characteristics and requirements of critical IoT use-cases driving the need for optimized manycore edge framework.

While big data analytics in IIoT environments can help to extract meaningful insights and business values, challenges related to heterogeneity, huge volume, and real-time velocity of data need to be addressed. Such requirements may exceed the computing power of existing systems and the conventional analytic models are no more suitable. As a result, a lot of research has been focusing on efficient analytics models such as data stream mining and concentric computing models [14]. Moreover, it is essential to filter and pre-process the data to expedite the analysis process. As a result, several libraries, tools, platforms, and advanced methodologies such as self-adaptive pre-processing [15] have been proposed for prepossessing the data, including data cleaning, normalization, transformation, missing value imputation, noise identification, and so on [16]. Although data prepossessing is a powerful tool to treat the complex data and speed-up the analysis process, it may consume a large amount of processing resources at the edge.

Hence, there is a need for powerful and scalable edge infrastructure to support advanced applications. Figure 2

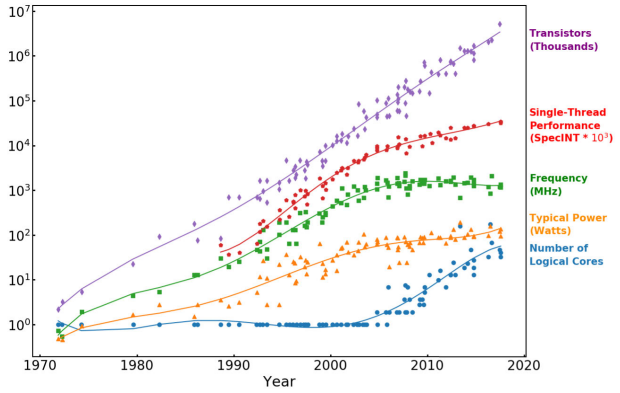


FIGURE 3. Microprocessor trend data for last five decades [19].

summarizes the key IoT use-cases, and highlights the key drivers for optimized manycore edge framework for providing high-performance and scalable compute, communication and caching at the edge. In addition to IIoT, internet of vehicles (IoV) is another emerging paradigm where massive data generated by autonomous vehicles, IoT devices and the environment can be harnessed to provide intelligent IoV services [17], [18]. Hence, to support time-sensitive content dissemination, and provide intelligent services, high performance and scalable edge framework is required for performing data collection, aggregation and processing locally.

Although edge computing offers groundbreaking solution for providing low latency by placing key processing tasks closer to the end user, its performance can be limited by underlying OS and system level bottlenecks in the edge host.

Looking at the evolution of processors over the last three decades, it can be observed that the number of transistors continued to increase until 2008, resulting in higher operating speeds and single-thread performance, as shown by the trend-lines in Figure 3 [19]. However, since 2008, although the transistor density is still increasing, it is becoming difficult to increase the operating speeds, indicating that single-thread performance is stagnant. The main reason for stalled single processor performance is the inability to increase the clock speed further due to high power consumption and power dissipation. On the other hand, the number of cores has been increasing significantly since 2008. Thus, from recent technology advancement trends in processors technology, it can be seen that there has been a continuous increase in the number of cores.

However, the performance of existing software architectures and monolithic OSs does not scale well with the increase in number of cores. The monolithic OSs were designed as feature-rich OS for processors with small number of cores. However, their design principles such as shared memory, locking, and caching mechanisms cause performance bottlenecks that become difficult to overcome as the number of cores increase. In the case of single-core processors, the gap between the on-chip computing power and the memory access speed could be addressed using sophisticated caching architectures or pre-fetch engines. However, the emergence of manycore chips adds another level of complexity and this issue cannot be addressed by hardware optimizations alone. Hence, there is a need to focus on software and OS level bottlenecks to address the issues related to concurrent access to scarce shared resources (memory bandwidth, network resources, on-chip memory, etc.) by large number of cores, without a major impact on the overall performance.

First of all, the *computation* performance of manycore systems can be affected by the scheduling, synchronization, and locking mechanisms in the OS kernel, as a large number of heterogeneous applications with varying requirements access the shared resources in parallel [8]. The traditional scheduling mechanisms do not perform well for parallel applications on manycore systems due to spin-lock contention and a large number of context switches. This problem is aggravated further when limited memory bandwidth is shared amongst large number of CPU cores for compute-intensive applications due to the increased cost of context switch. Hence, there is a need for lightweight task scheduling, scalable synchronization and enhanced locking mechanisms to unlock the potential of powerful manycore systems.

Early Linux was developed for single processor systems. Later, kernel 2.0 included support for symmetric multiprocessing (SMP), but the performance was not good because of the big kernel lock that serialized the access across the system. The real power of SMP was exploited with kernel version 2.6.39, that included fine-grained locking mechanism, resulting in performance improvement. In the later kernel versions, although big kernel locks have been improved, the scalabilty problem recurred while testing in the manycore environment.

To analyze the Linux kernel performance for manycore systems, we experimented with AIM7 benchmark tool for ext4 file system with hard disk drive (HDD), solid state drive (SSD), and virtual memory file system (tmpfs). The shared workload comprising of even distribution between compute and I/O operations was used for the AIM7 benchmark to analyze how the system scales under ever-increasing load. AIM7 is a multi-user benchmark suite capable of testing the performance of different aspects of operating system functionalities such as disk-file operations, process creation, user virtual memory operations, pipe I/O, and compute-bound arithmetic loops [20]. The experiments conducted with AIM7 for Linux kernel version 4.1 (ticket spinlock) are illustrated in Figure 4. Experimental results show that there are serious performance issues in the block I/O environment including the disk, and scalability up to 60 cores was shown in the environment where the block I/O was removed (i.e., tmpfs environment). Performance degradation was observed, with a significant performance drop from 60 cores on-wards. Thus, the problems of the locking mechanism itself, the problem of big kernel lock, and the architecture of traditional operating systems designed with the concept of sharing and fairness of resources impose obstacles to Linux scalability in manycore

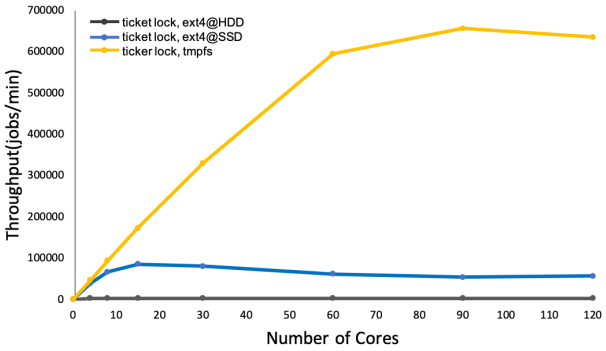


FIGURE 4. Performance results with AIM7 Benchmark for multi-user workload for ext4 filesystem with hard disk drive (HDD), solid state drive (SSD),and virtual memory filesystem (tmpfs) on manycore system.

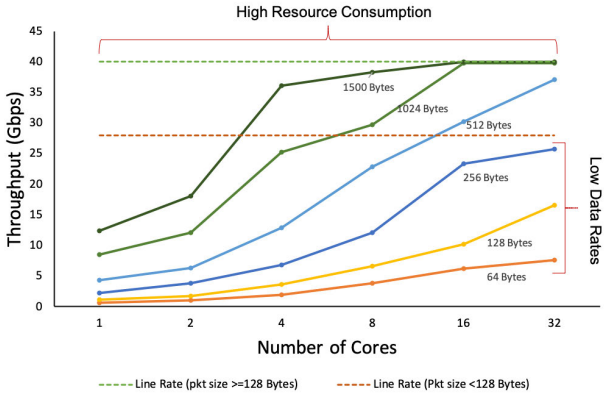


FIGURE 5. Variation of packet processing throughput with different numbers of cores and packet sizes.

systems. Hence, there is a need for enhanced locking mechanisms as well as scalable file systems to enhance the *caching* performance.

Although the processing power of end-systems has increased considerably and network speed has scaled up to high data rates (e.g., 40G/100G), the network performance is limited by the network stacks in monolithic OS kernels such as Linux. High performance and scalable *communication* is critical for supporting ultra-low latency requirements of edge applications, as well as for satisfying the QoS requirements for implementing network function virtualization (NFV) at the edge. NFV allows implementation of network functions in software, thereby providing higher flexibility and scalability for communication between the virtualized applications, and with the external integrated networks [21]. However, it is difficult to match the performance of hardware-based network functions and middleboxes.

Communications performance (i.e., throughput) for 40G network interface on a manycore system running Linux kernel 4.1 is illustrated in Figure 5. For small packet sizes, the achieved rate is much lower than the actual line rate. Meanwhile, for large packet sizes, a large number of cores are required to achieve full line rate performance. Moreover, the performance doesn't scale beyond 32 cores due to overheads associated with the kernel network stack. Several core mechanisms in the kernel such as scheduling, synchronization, interrupt handling, memory management, etc. cause serious performance overheads as the packets travel through the kernel network stack.

Hence, for manycore systems to be ubiquitously used, there is a need for enhanced system software, software models, programming models and tools to harness the full capability of these cores [22]. Current state of art solutions such as kernel bypass mechanisms and new OS designs are not feasible for flexible application design and processing and large scale virtualization, as they do not leverage the features of existing OS kernels that have already verified their robustness and stability in the production environment. It is therefore important to address the OS level bottlenecks in existing OSs so that the flexibility and features offered by software stacks in monolithic kernels can be used without performance and scalability issues.

III. FENCE FRAMEWORK

In this section, we first describe the framework of FENCE and then explain its key enabling technologies.

A. OVERALL ARCHITECTURE

The main architectural components of edge computing include edge host plane, and the management plane comprising of host level management and system level management [23]. The FENCE framework and its role as a high performance and scalable mobile edge host in ETSI MEC architecture, as a reference architecture [24], is illustrated in Figure 6. In the MEC architecture, the mobile edge host level management handles the management of the mobile edge specific functionality of a particular mobile edge host and the applications running on it. Meanwhile, the mobile edge system level management comprises of the MEC orchestrator, operation support system (OSS), and the user application life-cycle management proxy.

FENCE represents the edge host entity, and its main role is to provide a high performance and scalable platform and virtualization infrastructure for providing compute, communication and storage resources for a diverse set of applications. However, note that its role is not restricted as MEC edge host in particular and can also be deployed in micro data centers, compute on-premise or enterprise edge, depending on the deployment scenario.

As shown in Figure 6, the FENCE framework comprises of FENCE edge host and FENCE edge platform. The FENCE edge platform includes the required functionality for running edge applications and enabling them to consume edge services. Also, the FENCE edge host consists of 1) manycore infrastructure layer, 2) high performance and scalable OS and virtualization layer, and 3) application and service layer. The infrastructure layer comprises of a large number of cores, connected to each other, and to fast storage resources (SSD, NVMe, Non-Volatile Dual In-line Memory Module

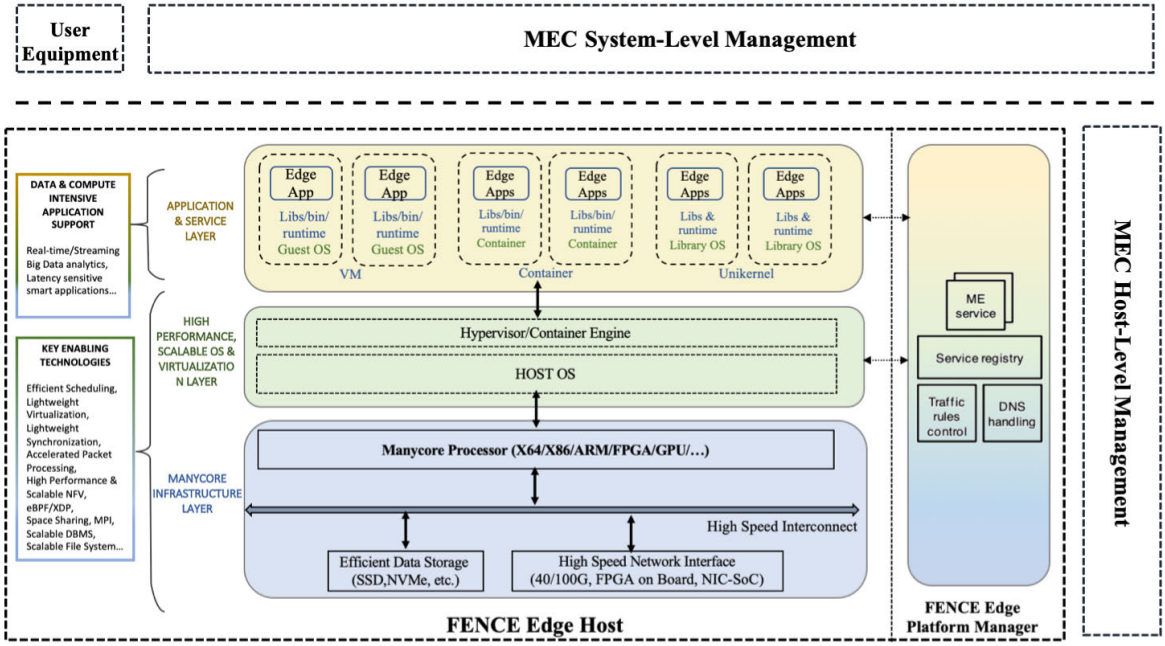


FIGURE 6. FENCE framework architecture.

(NVDIMM)) and high speed network interfaces (40G/100G) using high-speed interconnect. The types of cores i.e. x64, x86, ARM, or special processor cores such as FPGA or GPU can be selected based on the implementation scenario. To further enhance the performance, heterogeneous cores can be used together, such that a large number of small cores can be used for parallel processing and faster cores for executing the sequential tasks efficiently [25]. The availability of different types of cores allows resource allocation based on the specific application requirements, thereby optimizing the performance and scalability. Moreover, there is a possibility of seamless inclusion of heterogeneous resources, making the FENCE framework more flexible and extensible.

The OS and virtualization layer leverages the underlying manycore infrastructure and further enhances performance speedup achieved. To overcome the bottlenecks in monolithic kernels as discussed in the previous section, and achieve optimal performance and scalability, we propose the use of various state-of-art OS technologies and kernel enhancements for manycore systems. As a result, FENCE is capable of providing (i) high performance *computation* using lightweight task scheduling (FLsched), enhanced locking and synchronization mechanisms (ShflLock, MV-RLU) and lightweight virtualization; (ii) advanced *communications* (e.g., accelerated packet processing) using packet acceleration and optimized kernel packet processing (mKPAC); (iii) efficient data *storage* using scalable file systems (hybridF2FS, pNOVA). Each of these enabling technologies and enhancements supported by FENCE will be discussed in the subsequent subsection.

Lastly, the application and service layer supports different IoT applications and various services such as location service, machine learning engines, and network management service, etc., required to host edge applications effectively.

B. KEY ENABLING TECHNOLOGIES

In this section, we will highlight the key enabling and optimization technologies for providing high performance and scalability over manycore systems. These built-in enhancements can help to improve the overall system behavior during run-time and meet the requirements for massive parallelism, fast caching, computation, and high speed data transfer.

1) LIGHTWEIGHT TASK SCHEDULING

The dynamic scheduling of tasks on manycore processors can be challenging and may incur additional overheads. The existing scheduling mechanisms used in the monolithic OS, such as completely fair scheduler (CFS), first in first out (FIFO), round robin (RR) in Linux, do not perform well for parallel applications on manycore systems due to spin-lock contentions and a large number of context switches, as summarized in Table 1 [26]. Such schedulers were designed for a small number of cores such as quad core processors, however, commonly used servers in existing data centers comprise of up-to 32 cores. The performance and scalability of different applications can be drastically affected by small sequential components in the system. According to Amdahl's law, a significant decrease in performance (max speed) from 50 to 33 percent can occur even with a small increase of about 1 to 2 percent in the sequential component in the whole system [25]. The existing schedulers in Linux kernel are unable to handle a high degree of parallelism because their design comprises of various locking

TABLE 1. Locks involved in various scheduling mechanisms [26].

Lock Types	CORE	CFS	FIFO/RR
raw_spin_lock	16	1	12
raw_spin_lock_irq	4	2	-
raw_spin_lock_irqsave	9	3	2
rcu_read_lock	14	5	1
spin_lock	-	-	-
spin_lock_irq	3	-	-
spin_lock_irqsave	9	-	-
read_lock	3	-	-
read_lock_irq	-	-	-
read_lock_irqsave	1	-	-
mutex_lock	6	-	-
TOTAL	**65**	**11**	**15**

primitives such as mutex, read-write semaphores, spinlocks, etc. The performance degradation becomes worse for communication intensive applications requiring frequent scheduler intervention. Another issue is the increased cost of context switches, which exaggerates when limited memory bandwidth is shared amongst large number of CPU cores, especially for compute-intensive applications.

Hence, to overcome these bottlenecks and enhance the scalability, we advocate the use of light weight scheduling mechanisms that are optimized for manycore systems with highly parallel workloads. For instance, [26] presents a lightweight scheduling approach (FLsched) characterized by a lock-less design, less context switches and more effective scheduling decisions by minimizing the scheduling updates. In the case of manycore systems, making fast scheduling decisions is more critical than making fine-grained decisions as the availability of a large number of cores can be leveraged to handle different tasks.

2) LIGHTWEIGHT VIRTUALIZATION

Since edge computing provides a virtualized environment for hosting IoT application and edge services, it is essential to extract maximum performance and parallelism while minimizing the impact of virtualization, especially for compute intensive or latency sensitive applications. Different virtualization technologies such as virtual machines (VM), containers, and unikernels can be used to allocate virtualized resources depending on the application requirements [27].

An overview of these virtualization technologies is shown in Figure 7. VMs share the underlying hardware resources through virtualization, but run their own copies of the guest OS. VMs suffer from poor scalability as the hypervisor multiplexes the hardware resources for assigning virtual CPUs to physical CPUs, which can cause the preemption of the guest OS while it is still executing its critical section. In this case, multi-level scheduling can be used to address the inconsistency between the hypervisor and the guest OS [28]. Hence, higher throughput and scalability can be achieved in virtualized environments by preventing preemption when guest OS is running critical tasks.

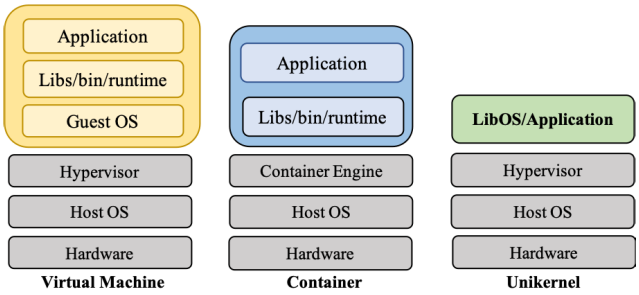


FIGURE 7. Overview of virtualization techniques including VMs, containers and unikernels.

Meanwhile, containers provide a lightweight virtual environment for applications by encapsulating only the essential dependencies and applications over a shared OS. This also results in low management overhead as compared to VMs. The shared kernel features also allow a higher density of virtual instances on given hardware due to the reduced image volume.

Unikernel is a specialized ultra-lightweight virtualization technique that allows the creation of a single address space machine image using library OS. It allows the creation of OS with the minimal set of OS constructs or libraries, as required by the applications, thus having a very small footprint [29]. Therefore, unikernel is ultra-lightweight and provides higher security and deployment efficiency as compared to VMs and containers.

Virtualization adds another level of complexity in manycore systems, as the introduction of virtual CPUs further complicates the scalability of spinlocks for synchronization. Moreover, manycore systems deployed as edge hosts must take several key aspects of edge computing such as diverse application requirements, lightweight performance, software portability, security and need for high scalability into consideration. Therefore, we consider unikernel as a key candidate technology for lightweight virtualization. Unikernels comprise of a minimal set of constructs required to run the application, and hence they boot and run faster, involve less context switching between kernel and user space, and provide enhanced security through minimized attack surface [30]. Hence, they can potentially extract maximum performance and parallelism while minimizing the impact of virtualization.

3) SCALABLE LOCKING

Increased adoption of manycore machines has uncovered scalability issues in monolithic operating systems such as Linux. Locks are critical to the design of parallelized application and concurrent programming in multicore system software; however, locking algorithms are a primary and probably most severe cause of performance and scalability bottlenecks. Lock designs have evolved over time to solve these issues, however, their design is primarily influenced by hardware evolution and hence may be optimized only for a particular platform or scenario. For instance, MCS

lock, a cache-aware implementation of read/write spinlocks was proposed to alleviate the cache-line congestion occurring when a large number of threads attempt to acquire the lock in parallel [31]. On the other hand, Cohort locks were designed to overcome locking issues in non-uniform memory access (NUMA) systems. However, there is a trade-off, as the optimized spinlocks suffer from scalability issues in NUMA systems, while NUMA-aware locking mechanisms suffer from sub-optimal single-thread performance. Hence, there is a need to consider other factors such as memory footprint, core over-subscription, thread count, etc., that may impact the scalability of different locks and their adoption in different scenarios.

In this case, we advocate the use of scalable locking mechanism such as ShflLocks, that consider all these factors without slowing down the critical path, to achieve high throughput and scalability [32]. The key idea is to shuffle the queue of threads waiting to acquire the lock based on some pre-established policy. Moreover, it allows the implementation of a diverse set of policies depending on hardware and application requirements, resulting in high performance and scalability for a wide array of use-cases.

4) HIGH PERFORMANCE AND SCALABLE SYNCHRONIZATION

In a manycore system, multiple threads frequently access shared resources simultaneously, and as the number of threads running concurrently increases, the performance of the system depends heavily on the synchronization mechanisms. In most software designs, such as operating systems, database systems, network stacks, and storage systems, synchronization mechanisms are essential building blocks and have a significant impact on performance and scalability. Recently, the number of processor cores is increasing and hardware parallelism is rapidly developing, but the performance scalability of software does not match the hardware advancement. Various algorithms have been proposed for synchronization; however no algorithm provides performance scalability in proportion to the hardware performance of modern manycore machines. Hence, to reduce contention, enhanced synchronization methods such as read-log-update (RLU) and multi-version RLU (MV-RLU) must be employed.

Hence it is essential to choose the mechanism that provides high performance and scalability for a large number of cores, without compromising the flexibility of use. One such mechanism is the read-log update (RLU) [33], which is an extension of the well-known read-copy-update (RCU) mechanism. RLU allows an unsynchronized sequence of read operations to execute in parallel with updates, thereby enhancing the scalability. Although it offers enhanced performance for read-mostly workloads, its performance significantly drops in workloads dominated by write operations, attributed to the fact that RLU supports only dual version concurrency. To overcome these limitations, MV-RLU was proposed to provide efficient multi-versioning [34].

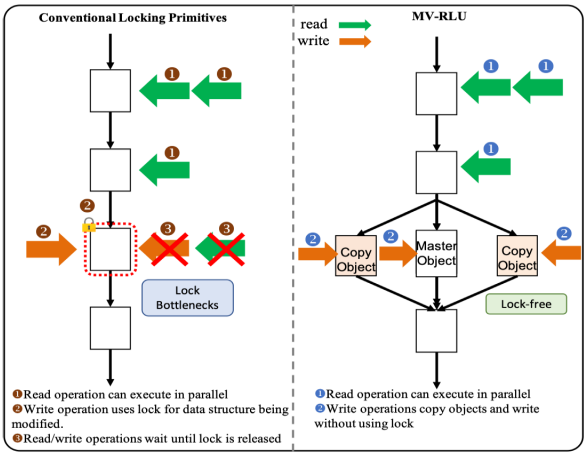


FIGURE 8. Illustration of conventional locking primitives vs. MV-RLU for large number of high-Speed read/writes for in-memory databases.

Multi-version logging is a mechanism that allows multiple threads to access shared resources simultaneously. It is a structure that connects one node or object by version chain and multi-versions and gives each version a timestamp so that each thread can access the appropriate version, as shown in Figure 8. This mechanism is supported by traditional database systems, but the biggest problem is the cost of version chain traverse and performance degradation caused by the recycling of log buffers. To solve this problem of traditional multi-version, log buffer recycling is performed in parallel for each thread, autonomous garbage collection is used to minimize version chain traverse cost, and timestamp publishing cost is minimized. Such mechanisms help to considerably improve the performance and scalability for database management systems (e.g. Kyoto Cabinet, DBX-1000) used for enhanced storage and caching support on manycore systems. Moreover, since OS network stack, database systems, and storage systems rely heavily on synchronization mechanisms, better performance and scalability are observed by employing MV-RLU over FENCE.

5) PACKET ACCELERATION AND IMPROVED KERNEL PACKET PROCESSING

Communications performance in a virtualized edge platform is critical to meet the QoS demands of different applications. As the advanced networks are now capable of supporting high speeds, the performance bottleneck is shifting from the bandwidth of the transmission media to the capability of end-hosts to process the incoming and outgoing data.

Although the processing power of end-systems has increased considerably and network speed has scaled up to high data rates (e.g., 40G/100G), the communications performance is limited by the network stacks in monolithic OS kernels such as Linux. The performance can further degrade when underlying network resources are shared between different virtualized applications through network function virtualization (NFV) [21]. Hence, the packet

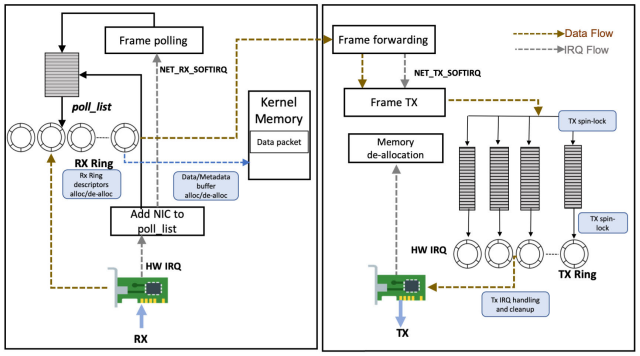


FIGURE 9. Packet Reception (RX) and Transmission (TX) in Linux kernel network stack and associated kernel-space overheads.

processing performance in monolithic kernels fail to match the performance of hardware-based network functions and middleboxes.

In order to optimize the packet processing performance for high data rate flows several acceleration techniques have been exploited [35]. Hardware-based acceleration frameworks such as ClickNP and PacketShader and NFV platforms such as NetVM and ClickOS have been used to accelerate the network processing. Moreover, network offloading techniques including kernel-bypass mechanisms (e.g., DPDK and Netmap) and user-space TCP stacks (e.g., mTCP, accelerated network stack (ANS)) provide high packet processing performance [36]. However, such acceleration mechanisms have a number of drawbacks such as excessive memory usage, low security, scalability issues, and may require development of new systems or re-implementation of software functions from scratch in user space, instead of relying on well-implemented and robust kernel stacks. Moreover, since kernel bypass approaches provide drivers for user space applications to directly access the hardware, they are exposed to security vulnerabilities, especially when applications from different service providers and third-party application providers share the same virtualized infrastructure.

Therefore, we emphasize on leveraging the robustness of implementation of existing feature-rich kernels and use their built-in features such as netfilter, iptables, ipsec, cgroups, etc., for flexible composition and deployment of network functions. High performance and scalability are achieved over FENCE by using mKPAC for addressing the bottlenecks in the transmit (TX) and receive (RX) paths of Linux kernel network stack. Main overheads are associated with RX/TX data and metadata buffer allocation and freeing, TX/RX ring descriptor management, TX spinlocks, and TX interrupt handling and cleanup, as summarized in Figure 9 [37]. In addition, optimizations such as the use of huge pages to reduce the frequency of page table lookups, CPU pinning to improve cache locality, NUMA-aware resource allocation etc. can be exploited to leverage heterogeneous manycore systems for performance improvement. To further optimize the throughput and latency, offloading the network processing to smart NICs (system-on-chip (SoC), NICs with on-board FPGAs) or packet processing based on extended Berkeley packet filer (eBPF) [38], and traffic classifier can be considered [39].

6) SCALABLE FILE SYSTEMS

Fast and scalable storage at the edge is one of the key enablers for satisfying the latency requirement of I/O intensive use-cases, that require a huge amount of data to be stored and accessed, at least temporarily, for efficient analysis and decision making. Providing storage capabilities at the edge helps to process the data locally, avoiding the transmission cost, delay and security risks associated with sending data to the cloud.

As application level parallelism is increasing in response to the advancement of the hardware, high performance database engines are being designed to leverage the computational parallelism and handle the queries concurrently. The parallel write performance of storage devices significantly affect the performance of I/O intensive workloads as compared to parallel reads. Hence, choosing the right memory and storage solutions at the edge is critical for mission-critical use-cases, and flash-memory based fast storage mediums like SSDs can be the promising solutions. To further enhance the scalability for I/O intensive workloads, NVMe has been used for accessing SSDs connected through high speed interconnect such as PCI Express bus. In addition, evolving edge computing has triggered the use of persistent memory modules such as NVDIMM. By connecting non-volatile NAND flash memory based storage media via dual in-line memory module (DIMM) slots, low access latency is achieved as NVDIMM devices are directly exposed to the memory bus without the intervention of I/O controllers [40].

In spite of the availability of fast storage and related specifications for parallelism, the traditional file systems in monolithic OS disrupt the application scalability, attributed mainly to indexing structure, consistency mechanisms, and coarse-grained locking mechanisms used by them [41]. As a result, new file systems such as flash-friendly file system (F2FS) and non-volatile memory accelerated file system (NOVA) were developed for improving the performance of fast storage devices. However, the scalability issues for I/O intensive applications still remained unsolved.

Hence, there is a need to address the scalability issues to exploit the full potential of high speed storage on manycore systems. For instance, the F2FS shows performance degradation on parallel write operations due to synchronization overheads, high processing overhead when multiple threads call Fsync in parallel, and overheads related to periodic checkpointing mechanism to recover from system crashes. To mitigate these overheads, FENCE makes use of scalable file systems such as hybridF2FS [42]. HybridF2FS mechanism proposes (i) the use of fine-grained range locks, allowing parallel write to mutually exclusive file ranges, and (ii) use of extended NVM storage space for storing file and file system metadata at high speed during checkpointing and Fsync operations [43].

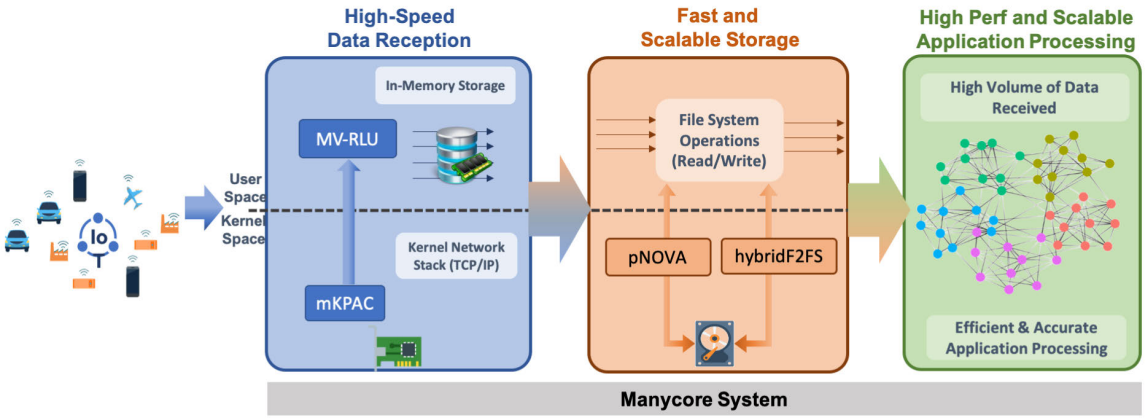


FIGURE 10. Overview of FENCE prototype for evolving graph processing flow.

Similarly, NOVA is an advanced persistent NVM file system that ensures data and metadata consistency by logging on per-file basis. Moreover, for persistent memory such as NVDIMM, NOVA outperforms legacy file systems [44]. Although such NVM-based file systems provide better performance (higher throughput and lower read/write latency) as compared to the block device based file systems, they also exhibit scalability issues when multiple threads perform I/O operations simultaneously. The scalability issue in NOVA is attributed to the coarse-grained locking mechanism per file and invalidation of cache-lines of waiting threads when a reader lock is acquired. To overcome these issues, a variant of NOVA called pNOVA has been proposed [45]. It implements an interval tree based fine-grained range locking mechanism and alleviates the cache-line invalidation by using range locking variable per-file. Hence, by employing hybridF2FS and pNOVA in FENCE, high read/write throughput and high scalability is achieved for a large number of cores, as these file systems are optimized for parallel reads and writes.

IV. PERFORMANCE EVALUATION

To illustrate the benefits of the proposed framework, we implemented a prototype of FENCE on manycore system and used it for performing massive stream processing for evolving graph processing application. Graphs are the key building blocks used to address various problems including machine learning, data mining, scientific computing, world wide web, and social networks. However, an unprecedented increase in the size of data sets poses fundamental challenges to current graph processing engines [46]. To overcome this issue, most of the research is focused on distributed graph processing engines, which involves high cost (large number of servers and interconnects) and complexity (load imbalance, distributed locking, etc.). Hence, FENCE framework can potentially be used to solve the capacity and performance issues.

Figure 10 shows the overview of the processing flow for graph engine, commonly used for analyzing streaming big data in real-time. A large amount of stream data is continuously generated and delivered to the server. In the data reception phase, the data is received and stored in the temporary storage such as in-memory database. In-memory storage is faster than disk-optimized databases as accessing data in memory eliminates seek time when querying the data, which provides faster and more predictable performance than disk. The storage step processes the temporarily stored data into meaningful graph data, and stores it in the database for further use. These databases are then regularly accessed and used for graph processing to make decisions.

To achieve scalable performance required for receiving, storing, and processing the data for evolving graph processing, we applied some of enabling techniques and optimizations such as mKPAC, MV-RLU, hybridF2FS, and pNOVA, (discussed in Section III-B), as illustrated in Figure 10. 56-core Intel Xeon processor with 512 MB memory, running Linux kernel version 4.1, 40G Intel XL710 network interface card (NIC), and Samsung 860 pro 512GB SSD were used for the prototype implementation. We also tested the performance with Intel Optane Data Center Persistent Memory Module (DCPMM) persistent memory (Apachepass) 128GB*12, DDR4 PC4 ECC Register 32GB*12 to demonstrate the benefit of FENCE for advanced storage media.

A. DATA RECEPTION PERFORMANCE

High-speed incoming data received via high speed network interface (40G Intel XL710), is quickly received using the kernel network stack and passed on to the system for further processing. To overcome the kernel space overheads associated with the RX processing path of the network stack, mKPAC kernel-level optimizations were applied to Linux kernel running on the manycore server. As a result, optimized RX performance, i.e., faster data reception with lower data loss is achieved. The performance enhancement achieved using mKPAC on FENCE, as compared to the baseline Linux kernel is illustrated in Figure 11(a). With 64 Bytes packet

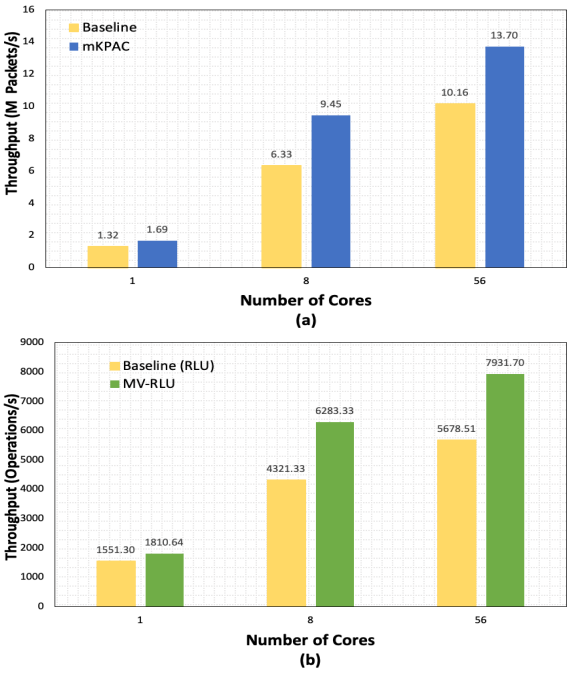


FIGURE 11. Data reception performance: (a) RX throughout and (b) memory read/write throughput (read:write ratio of 1:1).

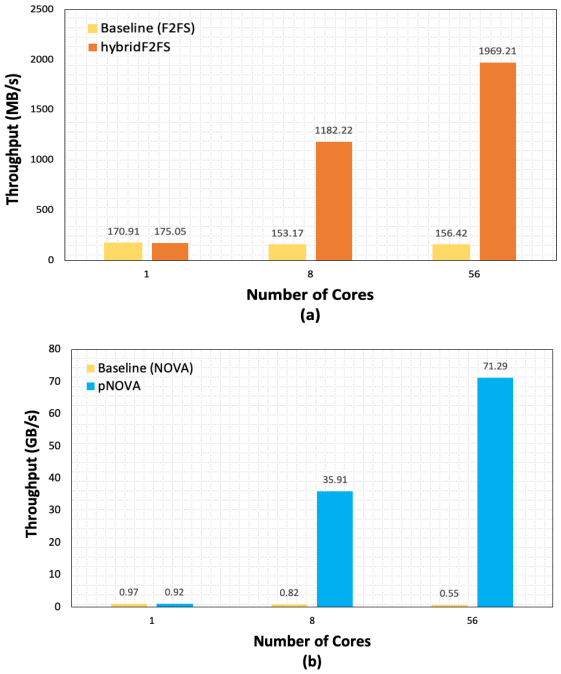


FIGURE 12. Storage performance (read/write throughput) for: (a) SSD and (b) NVDIMM.

size, up to 40 percent acceleration in RX rate, in packets per second (PPS), as compared to baseline Linux kernel, was achieved using FENCE.

The received data is then temporarily stored in in-memory database before it is read into the databases for further storage and processing. The read operations for storing the graph data to the disk are performed in parallel with the large write operations to the memory. In order to solve the bottleneck caused by concurrent read and write operations, we adapted MV-RLU into in-memory database. To address these issues, MV-RLU employs a lock free structure through multi-versioning, which leads to improved performance. Specifically, as illustrated in Figure 11(b), MV-RLU improves the throughput (in operations per second) up to 45 percent for a large number of cores, as compared to RLU in the baseline system.

To summarize, high-speed data ingestion can be achieved by using mKPAC for enhanced kernel packet processing and MV-RLU for optimized read/write performance for in-memory database.

B. STORAGE PERFORMANCE

High-speed storage involves continuously reading the data stored in the temporary storage and converting it into meaningful graph data, and storing the converted graph data in the database. The read/write performance and manycore scalability can be degraded when multiple threads attempt to read/write to a shared file concurrently, as discussed in Section III-B. In this case, high performance variant of F2FS file systems i.e., hybridF2FS can be applied for FENCE to overcome these bottlenecks and support high speed and scalable storage for SSD (Samsung 860 pro 512GB SSD). The read/write throughputs for hybridF2FS in FENCE and F2FS in the baseline system are illustrated in Figure 12(a). It can be seen that HybridF2FS shows around 6x improvement in performance as compared to F2FS as the number of cores increases. To mitigate the overheads that arise when multiple threads attempt to write the meaningful graph data to the database, hybridF2FS uses fine-grained range locks and extends the NVM storage space for storing file and file system metadata at high speed.

Moreover, as NVDIMM is a promising storage technology for future use in edge computing, as discussed in Section III-B, we tested its performance to demonstrate the benefit of FENCE for advanced storage media (Intel Optane). In this case, pNOVA, a high performance variant of NOVA, was used. pNOVA implements an interval tree based fine-grained range locking mechanism, to improve the read/write performance when a large number of threads try to read/write in parallel. From Figure 12(b), it can be clearly seen that pNOVA significantly outperforms NOVA over Linux, and the performance scales well with the increase of the number of cores. As a result, fast and scalable storage performance was achieved for advanced storage media, while storing the relevant graph data for further processing.

C. APPLICATION PROCESSING PERFORMANCE

Different state of art technologies applied together help to optimized different subsystems of Linux, resulting in better overall system performance and scalability for data reception, storage, and processing. Hence, the full potential of

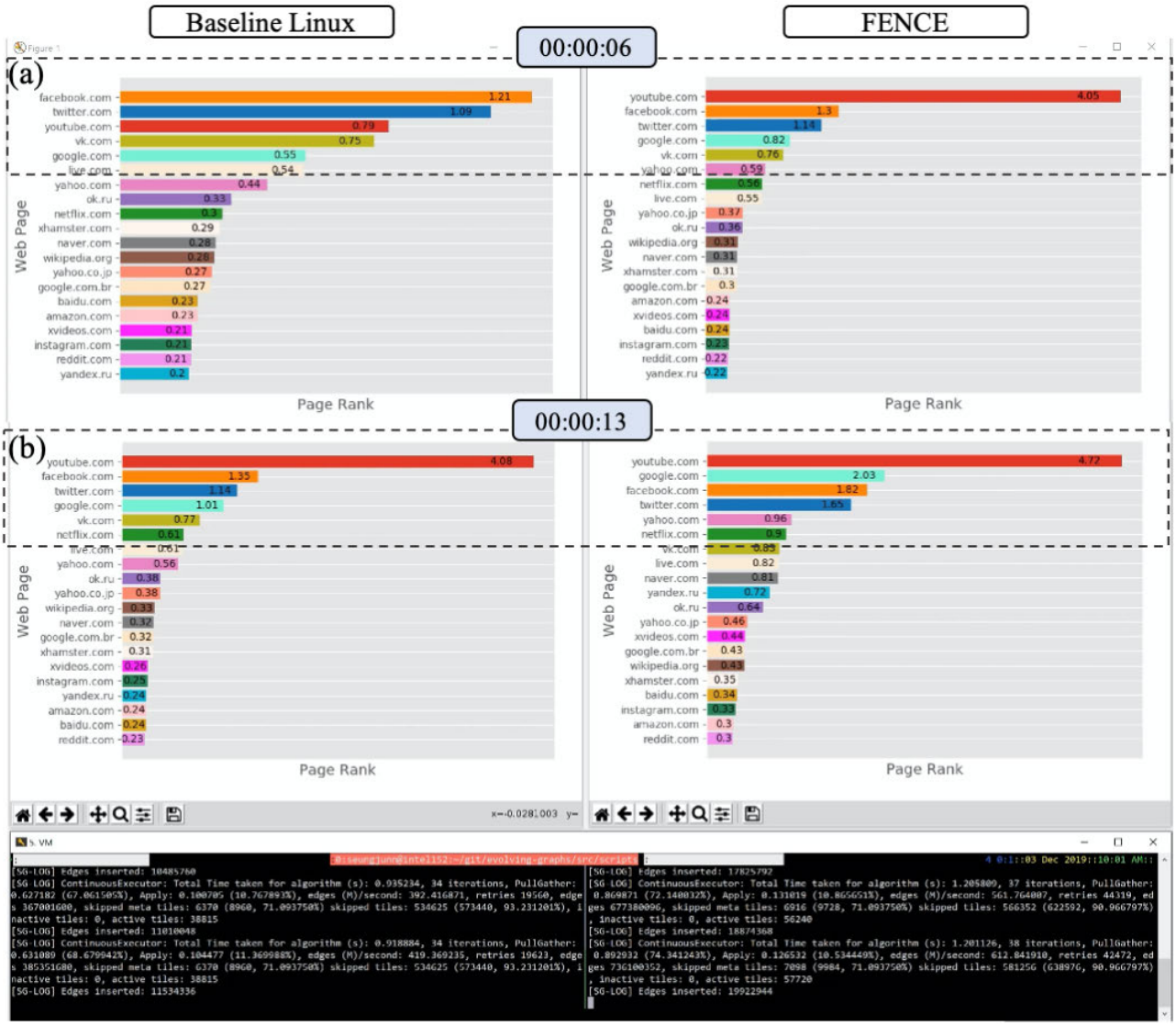


FIGURE 13. Graph processing execution results for PageRank analysis at: (a) $t = 6s$ and (b) $t = 13s$.

underlying manycore systems can be exploited, resulting in enhanced application performance and scalability. The proposed optimizations result in a large volume of data being available for efficient and accurate application processing. To show the application processing performance, we implemented an evolving graph processing engine [46], capable of processing a very large number of edges on a single machine.

Efficient graph processing is the key requirement for many applications, including graph analytic platforms. Graph analytics is an extremely flexible and powerful tool, that can consume both structured and unstructured data and works particularly well in cases with complex relationships among data points. Hence, it is particularly suitable for discovering unknown patterns and relationships between data in IIoT scenarios generating complex and dynamic data. For instance, its application to smart manufacturing, which is characterized by heterogeneous data with temporal and spatial redundancy, can help in enhancing supply-chain efficiency, improving factory operations and product quality, reducing machine downtime, creating new business values and enhancing customer experience.

In the current prototype, we tested the performance of graph processing for the PageRank algorithm. PageRank is one of the well-known link analysis algorithms and it assigns a numerical weight to each graph set, with the purpose of measuring its relative importance for big data application using an evolving graph engine. We demonstrate the superiority of FENCE in handling large volume and velocity of data by running PageRank on the Twitter graph data [47]. This enables the emulation of graph processing based on tweets generated and delivered in real-time. Note that a lot of YouTube related keywords appeared in tweets in real-time.

The results for the execution of the PageRank algorithm are illustrated in Figure 13. At time $t = 6s$, the differences in ranking by FENCE and baseline Linux are shown in Figure 13(a). It can be observed that the rank of YouTube is

the highest in the case of FENCE. On the other hand, the rank of Facebook is the highest on the system running baseline Linux. Eventually, at time $t = 13s$, Figure 13(b) shows that the YouTube rank is updated to the highest on the Linux, similar to the value observed in FENCE. This demonstrates that baseline Linux takes a significantly longer time to reflect real-time data. The latency in processing the incoming data makes it difficult to obtain accurate results and update them in real-time. Meanwhile, in the case of FENCE, a lot of stream data was received quickly through OS and software level optimizations, resulting in immediate processing of real-time data and accurate decision making.

V. CONCLUSION

The massive amount of data being generated in the IoT era needs to be harnessed and processed effectively to provide intelligent data-driven solutions. This requires high performance edge computing platforms to augment the intelligent networking effectively. This paper introduced a fast, extensible, and consolidated edge (FENCE) framework for intelligent big data processing. We analyzed the key drivers for edge computing in industrial IoT domains, and discussed how the FENCE framework can overcome OS and software level bottlenecks in manycore systems. We discussed various enabling technologies and system level optimization techniques for ensuring high performance and scalability in FENCE. The presented evaluation results corroborate the enhanced performance and scalability achieved using the FENCE framework. Given the high performance and scalability provided by FENCE, several additional challenges (e.g., distributed analytics model, data sharing across multiple domains, and security/privacy) need to be addressed for the ideal deployment of edge computing solutions in a distributed environment, which will be considered in the future work.

REFERENCES

[1] B. Marr. (2019). *How Much Data Do We Create Every Day? The Mind-Blowing Stats Everyone Should Read*. [Online]. Available: https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read/

[2] D. Mourtzis, E. Vlachou, and N. Milas, "Industrial big data as a result of IoT adoption in manufacturing," *Procedia CIRP*, vol. 55, pp. 290–295, Jan. 2016.

[3] B. Chen, J. Wan, A. Celesti, D. Li, H. Abbas, and Q. Zhang, "Edge computing in IoT-based manufacturing," *IEEE Commun. Mag.*, vol. 56, no. 9, pp. 103–109, Sep. 2018.

[4] Cisco. (2019). *Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2018–2022 Whitepaper*. [Online]. Available: http://www.cisco.com

[5] M. H. Rehman, I. Yaqoob, K. Salah, I. Muhammad, P. Jayaraman, and C. Perera, "The role of big data analytics in industrial Internet of Things," *Future Gener. Comput. Syst.*, vol. 99, pp. 247–259, Oct. 2019.

[6] H. Chang, A. Hari, S. Mukherjee, and T. V. Lakshman, "Bringing the cloud to the edge," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Apr. 2014, pp. 346–351.

[7] M. Mohammadi, A. Al-Fuqaha, S. Sorour, and M. Guizani, "Deep learning for IoT big data and streaming analytics: A survey," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 4, pp. 2923–2960, Jun. 2018.

[8] S. Boyd-Wickizer, A. T. Clements, Y. Mao, A. Pesterev, M. F. Kaashoek, R. Morris, and N. Zeldovich, "An analysis of Linux scalability to many cores," in *Proc. USENIX Conf. Operating Syst. Design Implement. (OSDI)*, Oct. 2010, pp. 1–16.

[9] I. Zhang, J. Liu, A. Austin, M. L. Roberts, and A. Badam, "I'm not dead yet!: The role of the operating system in a kernel-bypass era," in *Proc. Workshop Hot Topics Operating Syst.*, May 2019, pp. 73–80.

[10] (2019). *Data Plane Development Kit*. [Online]. Available: http://www.dpdk.org/

[11] R. Mittal, A. Shpiner, A. Panda, E. Zahavi, A. Krishnamurthy, S. Ratnasamy, and S. Shenker, "Revisiting network support for RDMA," in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2018, pp. 313–326.

[12] Z. Chang, L. Lei, Z. Zhou, S. Mao, and T. Ristaniemi, "Learn to cache: Machine learning for network edge caching in the big data era," *IEEE Wireless Commun.*, vol. 25, no. 3, pp. 28–35, Jun. 2018.

[13] (2019). *M-CORD Open Source Reference Solution for 5G Mobile Wireless Net—Works*. [Online]. Available: https://www.opennetworking.org/m-cord/

[14] M. H. U. Rehman, E. Ahmed, I. Yaqoob, I. A. T. Hashem, M. Imran, and S. Ahmad, "Big data analytics in industrial IoT using a concentric computing model," *IEEE Commun. Mag.*, vol. 56, no. 2, pp. 37–43, Feb. 2018.

[15] K. Lan, S. Fong, W. Song, A. Vasilakos, and R. Millham, "Self-adaptive pre-processing methodology for big data stream mining in Internet of Things environmental sensor monitoring," *Symmetry*, vol. 9, no. 10, p. 244, Oct. 2017.

[16] S. García, S. Ramírez-Gallego, J. Luengo, J. M. Benítez, and F. Herrera, "Big data preprocessing: Methods and prospects," *Big Data Anal.*, vol. 1, no. 1, pp. 1–22, Nov. 2016.

[17] J. Zhang and K. B. Letaief, "Mobile edge intelligence and computing for the Internet of vehicles," *Proc. IEEE*, vol. 108, no. 2, pp. 246–261, Feb. 2020.

[18] Z. Zhou, C. Gao, C. Xu, Y. Zhang, S. Mumtaz, and J. Rodriguez, "Social big-data-based content dissemination in Internet of vehicles," *IEEE Trans. Ind. Informat.*, vol. 14, no. 2, pp. 768–777, Feb. 2018.

[19] K. Rupp. (2020). *42 Years of Microprocessor Trend Data*. [Online]. Available: https://www.karlrupp.net/2018/02/42-years-of-microprocessor- trend-data/

[20] SourceForge. (2020). *AIM Benchmarks*. [Online]. Available: https://sourceforge.net/projects/aimbench/

[21] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Commun. Mag.*, vol. 53, no. 2, pp. 90–97, Feb. 2015.

[22] Ramneek, S.-J. Cha, S. H. Jeon, Y. J. Jeong, J. M. Kim, S. Jung, and S. Pack, "Boosting edge computing performance through heterogeneous manycore systems," in *Proc. Int. Conf. Inf. Commun. Technol. Converg. (ICTC)*, Oct. 2018, pp. 922–924.

[23] Y. Ai, M. Peng, and K. Zhang, "Edge computing technologies for Internet of Things: A primer," *Digit. Commun. Netw.*, vol. 4, no. 2, pp. 77–86, Apr. 2018.

[24] ETSI. (2019). *Multi-access Edge Computing (MEC); Framework and Reference Architecture*. [Online]. Available: https://www.etsi.org/

[25] M. D. Hill and M. R. Marty, "Amdahl's law in the multicore era," *Computer*, vol. 41, no. 7, pp. 33–38, Jul. 2008.

[26] H. Jo, W. Kang, C. Min, and T. Kim, "FLsched: A lockless and lightweight approach to OS scheduler for Xeon Phi," in *Proc. 8th Asia–Pacific Workshop Syst.*, Sep. 2017, pp. 1–8.

[27] R. Morabito, V. Cozzolino, A. Y. Ding, N. Beijar, and J. Ott, "Consolidate IoT edge computing with lightweight virtualization," *IEEE Netw.*, vol. 32, no. 1, pp. 102–111, Jan. 2018.

[28] S. Kashyap, C. Min, and T. Kim, "Scaling guest OS critical sections with ECS," in *Proc. Usenix Annu. Tech. Conf. (ATC)*, Jul. 2018, pp. 159–171.

[29] M. Plauth, L. Feinbube, and A. Polze, "A performance survey of lightweight virtualization techniques," in *Proc. Eur. Conf. Service-Oriented Cloud Comput. (ESOCC)*, Sep. 2017, pp. 34–48.

[30] A. Madhavapeddy, R. Mortier, C. Rotsos, D. Scott, B. Singh, T. Gazagnaire, S. Smith, S. Hand, and J. Crowcroft, "Unikernels: Library operating systems for the cloud," in *Proc. ACM Conf. Architectural Support Program. Lang. Operating Syst. (ASPLOS)*, Mar. 2013, pp. 461–472.

[31] J. M. Mellor-Crummey and M. L. Scott, "Algorithms for scalable synchronization on shared-memory multiprocessors," *ACM Trans. Comput. Syst. (TOCS)*, vol. 9, no. 1, pp. 21–65, Feb. 1991.

[32] S. Kashyap, I. Calciu, X. Cheng, C. Min, and T. Kim, "Scalable and practical locking with shuffling," in *Proc. 27th ACM Symp. Operating Syst. Princ.*, Oct. 2019, pp. 586–599.

[33] A. Matveev, N. Shavit, P. Felber, and P. Marlier, "Read-log-update: A lightweight synchronization mechanism for concurrent programming," in *Proc. 25th Symp. Operating Syst. Princ. - SOSP*, 2015, pp. 168–183.
[34] J. Kim, A. Mathew, S. Kashyap, M. K. Ramanathan, and C. Min, "MV-RLU: Scaling read-log-update with multi-versioning," in *Proc. 24th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, Apr. 2019, pp. 779–792.
[35] S. Gallenmuller, P. Emmerich, F. Wohlfart, D. Raumer, and G. Carle, "Comparison of frameworks for high-performance packet IO," in *Proc. ACM/IEEE Symp. Archit. Netw. Commun. Syst. (ANCS)*, May 2015, pp. 29–38.
[36] E. Y. Jeong, S. Woo, M. Jamshed, H. Jeong, S. Ihm, D. Han, and K. Park, "mTCP: A highly scalable user-level TCP stack for Multicore systems," in *Proc. USENIX Conf. Netw. Syst. Design Implement. (NSDI)*, Apr. 2014, pp. 489–502.
[37] Ramneek, M. Kumar, T. Kim, and S. Jung, "MKPAC: Kernel packet processing for manycore systems," in *Proc. 19th Int. Middleware Conf. Middleware*, Dec. 2018, pp. 15–16.
[38] S. Baidya, Y. Chen, and M. Levorato, "EBPF-based content and computation-aware communication for real-time edge computing," in *Proc. IEEE INFOCOM Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Apr. 2018, pp. 865–870.
[39] T. Høiland-Jørgensen, J. D. Brouer, D. Borkmann, J. Fastabend, T. Herbert, D. Ahern, and D. Miller, "The eXpress data path: Fast programmable packet processing in the operating system kernel," in *Proc. 14th Int. Conf. Emerg. Netw. Exp. Technol.*, Dec. 2018, pp. 54–66.
[40] R. Chen, Z. Shao, and T. Li, "Bridging the I/O performance gap for big data workloads: A new NVDIMM-based approach," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, Oct. 2016, pp. 1–12.
[41] C. Min, S. Kashyap, S. Maass, and T. Kim, "Understanding manycore scalability of file systems," in *Proc. USENIX Annu. Tech. Conf. (ATC)*, Jun. 2016, pp. 71–85.
[42] (2019). *oslab-swrc/hybridF2FS: A Varient of F2FS using Hybrid Logging with NVM and SSD*. [Online]. Available: https://github.com/oslab-swrc/hybridF2FS
[43] C.-G. Lee, H. Byun, S. Noh, H. Kang, and Y. Kim, "Write optimization of log-structured flash file system for parallel I/O on manycore servers," in *Proc. 12th ACM Int. Conf. Syst. Storage*, May 2019, pp. 21–32.
[44] J. Izraelevitz, J. Yang, L. Zhang, J. Kim, X. Liu, A. Memaripour, Y. J. Soh, Z. Wang, Y. Xu, S. R. Dulloor, J. Zhao, and S. Swanson, "Basic performance measurements of the intel optane DC persistent memory module," 2019, *arXiv:1903.05714*. [Online]. Available: http://arxiv.org/abs/1903.05714
[45] J.-H. Kim, J. Kim, H. Kang, C.-G. Lee, S. Park, and Y. Kim, "PNOVA: Optimizing shared file I/O operations of NVM file system on manycore servers," in *Proc. 10th ACM SIGOPS Asia–Pacific Workshop Syst. APSys*, 2019, pp. 1–7.
[46] S. Maass, C. Min, S. Kashyap, W. Kang, M. Kumar, and T. Kim, "Mosaic: Processing a trillion-edge graph on a single machine," in *Proc. 12th Eur. Conf. Comput. Syst.*, Apr. 2017, pp. 527–543.
[47] H. Kwak, C. Lee, H. Park, and S. Moon, "What is Twitter, a social network or a news media?" in *Proc. 19th Int. Conf. World Wide Web WWW*, 2010, pp. 591–600.

RAMNEEK (Associate Member, IEEE) received the B.Tech. degree in computer science and engineering from Guru Nanak Dev University, Punjab, India, in 2010, the M.Tech. degree in IT from the International Institute of Information Technology (IIIT) Bangalore, India, in 2013, and the Ph.D. degree in grid and supercomputing from the Korea Institute of Science and Technology Information (KISTI) Campus, Korea University of Science and Technology, Daejeon, South Korea, in August 2017. She was a Postdoctoral Researcher with the Cloud Computing Core SW Research Section, Electronics and Telecommunications Research Institute (ETRI), Daejeon, from December 2017 to December 2019. She is currently a Research Professor with the Mobile Network and Communications Laboratory, Korea University. Her research interests include networking and communication, the Internet of Things, block chain technology, network systems, and operating systems.

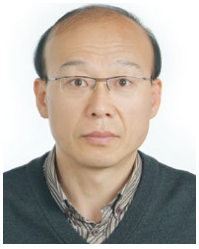
SEUNG-JUN CHA received the B.E., M.E., and Ph.D. degrees in computer engineering from Chungnam National University, Daejeon, South Korea, in 2006, 2008, and 2013, respectively. Since 2013, he has been with the Electronics and Telecommunications Research Institute (ETRI), where he is currently a Senior Researcher with the Cloud Computing SW Research Section. He has researched about database, XML, semantic web services, and middleware for service. His current research interests include operating system principles, including multi-kernel, microkernel, light weight kernel, and unikernel for the manycore systems.

SANGHEON PACK (Senior Member, IEEE) received the B.S. and Ph.D. degrees in computer engineering from Seoul National University, Seoul, South Korea, in 2000 and 2005, respectively. From 2005 to 2006, he was a Postdoctoral Fellow with the Broadband Communications Research Group, University of Waterloo, Waterloo, ON, Canada. In 2007, he joined Korea University, Seoul, as a Faculty Member, where he is currently a Professor with the School of Electrical Engineering. His research interests include future internet, SDN/ICN/DTN, mobility management, mobile cloud networking, multimedia networking, and vehicular networks. He was a recipient of the IEEE/IEIE Joint Award for IT Young Engineers Award, in 2017, the KIISE Young Information Scientist Award, in 2017, the Korea University Techno Complex (KUTC) Crimson Professor, in 2015, the KICS Haedong Young Scholar Award, in 2013, and the IEEE ComSoc APB Outstanding Young Researcher Award, in 2009. He served as a TPC Vice-Chair for information systems, the IEEE WCNC 2020, a track Chair for the IEEE CCNC 2019, a TPC Chair for EAI Qshine 2016, a Publication Co-Chair for the IEEE INFOCOM 2014, ACM MobiHoc 2015, a Co-Chair for the IEEE VTC 2010-Fall Transportation Track, a Co-Chair for the IEEE WCSP 2013 Wireless Networking Symposium, a TPC Vice-Chair for ICOIN 2013, and a Publicity Co-Chair for the IEEE SECON 2012. He serves as an Editor for the IEEE Internet of Things (IoT) Journal, *Journal of Communications Networks (JCN)*, and *IET Communications*. He is a Guest Editor of the IEEE Transactions on Emerging Topics in Computing (TETC).

SEUNG HYUB JEON received the M.S. degree from Korea University. He is currently a Senior Researcher with the Cloud Computing SW Research Section, Electronics and Telecommunications Research Institute (ETRI), Daejeon, South Korea. He focuses on the operating system scalability in manycore systems. His research interests include system software for parallel computing, virtualization, and multitier memory systems.

YEON JEONG JEONG received the B.S. and M.S. degrees in computer science from Pusan National University, Pusan, South Korea, in 1994 and 1996, respectively, and the Ph.D. degree in computer science from Chungnam National University, Daejeon, South Korea, in 2005. He is currently a Principal Researcher with the Cloud Computing SW Research Section, Electronics and Telecommunications Research Institute (ETRI), Daejeon. His research interests include unikernel, light weight kernel, and multikernel for manycore systems.

JIN MEE KIM received the B.E. degree in computer science from Pusan National University, in 1988, and the M.E. degree in computer science from Chungnam National University, in 1999. Since 1988, she has been with the Electronics and Telecommunications Research Institute (ETRI), where she is currently a Principal Researcher with the Basic Technology Research Center for Next-Generation OS. Her research interests include high-performance computing and operating system principles for the manycore systems.



SUNGIN JUNG received the B.E. and M.E. degrees in computer engineering from Pusan National University, in 1987 and 1989, respectively, and the Ph.D. degree in computer engineering from Chungnam National University, Daejeon, South Korea, in 2006. He was a UNIX kernel developer during the 1990s and has 29 years of experience in the operating systems area. He had participated in developing UNIX kernel for SMP, ccNUMA, and MPP systems in cooperation with Novell and SCO. In 2000, he began the Linux Kernel Project for Carrier Grade Linux (CGL) and Data Center Linux (DCL) work groups of Linux Foundation. He is currently a Principal Researcher with the Basic Technology Research Center for Next-Generation OS at Electronics and Telecommunications Research Institute (ETRI). Since 2014, he has been the project manager of the manycore OS research. He serves OSS activities, such as OSS policy and international events. His research interests include operating system kernel, cloud computing, HPC, and OSS.