BM-Store: A Transparent and High-performance Local Storage Architecture for Bare-metal Clouds Enabling Large-scale Deployment

Yiquan Chen[†*], Jiexiong Xu[†], Chengkun Wei[†], Yijing Wang[*], Xin Yuan[*]
Yangming Zhang[*], Xulin Yu[*], Yi Chen[†], Zeke Wang[†], Shuibing He[†], Wenzhi Chen[†]

[†]*Zhejiang University* [*]*Alibaba Group*

2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA) | 978-1-6654-7652-2/23/$31.00 ©2023 IEEE | DOI: 10.1109/HPCA56546.2023.10071029

***Abstract*—Bare-metal instances are crucial for high-value, mission-critical applications on the cloud. Tenants exclusively use these dedicated hardware resources. Local virtualized disks are essential for bare-metal instances to provide flexible and high-performance storage resources. Traditionally tenants can choose polling-based software virtualization techniques, but they consume too many valuable host CPU cores and suffer from performance degradation. Cloud vendors are hard to deploy existing hardware-assisted local storage solutions in bare-metal instances due to no access to the host OS to install customized drivers. Moreover, cloud vendors have difficulties managing and maintaining the local storage devices in bare-metal instances because hardware resources and host operating systems are completely utilized by tenants, then it will impact the availability of storage devices.**

This paper presents our design and experience with BM-Store, a novel high-performance hardware-assisted virtual local storage architecture for bare-metal clouds. BM-Store is transparent to the host that tenants are unaware of the underlying hardware architecture. Therefore, it can be deployed on a large scale in cloud vendors. BM-Store consists of two components: an FPGA-based BMS-Engine and an ARM-based BMS-Controller. The BMS-Engine accelerates the I/O path to enable high-performance virtual storage independent of disk devices without consuming any CPU resource on the host. The BMS-Controller is responsible for resource management and maintenance to achieve flexible and high available local storage. The results of the extensive experiments show that BM-Store can achieve near-native performance, which only introduces about 3 µs extra latency and average 4.0% throughput overhead to native disks. Compared to SPDK vhost, BM-Store achieves an average bandwidth improvement of 15.7% in microbenchmark and a maximum throughput enhancement of 13.4% in real-world applications.

I. Introduction

Bare-Metal (BM) clouds have become an essential Infrastructure-as-a-Service (IaaS) that leases dedicated physical servers (called bare-metal instances) rather than Virtual Machines (VM). Therefore, bare-metal instances are well suitable for resource-intensive applications such as big data applications, database servers, and media servers [26, 31]. Currently, bare-metal instances are widely available on public clouds (Amazon EC2 m5.metal instance [8], Alibaba Cloud ECS Bare Metal Instance Families [11], and Microsoft Azure Bare Metal Infrastructure [29]).

Bare-metal instances generally have two storage options: local storage [10] and remote storage [15, 20]. In remote storage, storage devices are located in the remote server, and tenants must access these storage resources through the network. In contrast, local storage means that storage devices are directly attached to the tenant's physical server and are accessed through the PCIe system bus. As a result, local storage provides high-throughput and low-latency I/O access for bare-metal instances. Cloud instances with local storage are widely employed by I/O intensive workloads [10] (e.g., online gaming [18], e-commerce [36], and live streaming [28]).

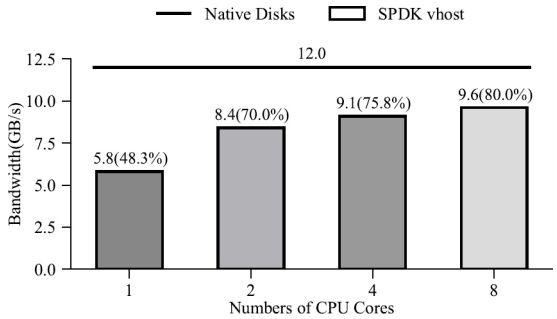


Fig. 1. Bandwidth of SPDK vhost binds with different numbers of CPUs on four SSDs. We ran fio with the test case of sequential read of 128k block size, queue depth of 256, 4 threads, using libaio as the I/O engine. Polling-based schemes consume too many valuable host CPU cores and suffer from performance degradation.

Traditionally, cloud vendors directly provide physical storage devices to bare-metal tenants. However, existing local storage schemes for bare-metal clouds have the following limitations:

- **Lack of hardware-assisted virtualization capability.** Generally, bare-metal tenants would deploy containers or virtual machines on physical servers. Meanwhile, tenants require flexible and isolated local storage resources. Therefore, virtualization capability is essential. Traditionally, tenants can adopt advanced software polling-based solutions (e.g., SPDK vhost [42] and NVMe-MDev[32]), but they consume too many valuable host CPU cores and suffer from performance degradation. For example, we evaluated the SPDK vhost [42] with four intel P4510 SSDs. In Fig.1, SPDK vhost needs to bind at least eight CPU cores for four SSDs to get only 80% of native performance. Furthermore, cloud vendors may adopt hardware-assisted virtualization mechanisms (e.g., pass through [6] and SR-IOV [13]) in a traditional way to provide virtual local storage. However, these solutions lack the sharing capability and compatibility that are

978-1-6654-7652-2/23/$31.00 ©2023 IEEE

unsuitable for deployment on bare-metal instances.

- **Management and maintenance challenges.** Cloud vendors have challenges managing and maintaining local storage devices in existing bare-metal instances due to no access to the host OS. In the traditional virtual machine instances, cloud vendors usually run specific programs in the host OS for these tasks, including disk status monitoring, disk firmware upgrade, I/O monitoring, etc. Unfortunately, cloud vendors cannot take the previous approaches in bare-metal instances.

Many hardware-assisted solutions are proposed to provide virtual local storage, but these schemes cannot be deployed on bare-metal clouds. LeapIO [27] proposed a solution to offload the local storage stack to the ARM SoC. FVM [24, 25] proposed a hardware solution based on FPGA to provide virtualization capability to local storage. However, both FVM and LeapIO adopted the Peer-to-Peer (P2P) architecture, which inevitably requires the installation of customized drivers. Thus, it is difficult for cloud vendors to deploy these hardware solutions on bare-metal instances due to cloud vendors cannot access the host OS. On the other hand, no existing local storage solution addresses the management and maintenance challenges.

This paper proposes BM-Store, the first hardware-assisted local storage architecture designed for bare-metal clouds, achieving transparency, virtualization, high performance, manageability, and high availability. BM-Store consists of two core components: *BMS-Engine* and *BMS-Controller*. The key ideas of BM-Store are: 1) BM-Store adopts transparent architecture to achieve large-scale deployment in bare-metal instances. We discuss details in Section IV-A. Bare-metal tenants are unaware of BM-Store hardware. They can access virtual local storage resources using standard NVMe drivers without additional drivers; 2) BM-Store provides high-performance storage virtualization to bare-metal instances by offloading the virtualization layer to the BMS-Engine hardware. Specifically, we design a zero-copy mechanism for data transfer to enable back-end SSDs directly access host memory, eliminating redundant data copy; 3) BM-Store integrates an out-of-band management mechanism and maintenance functions into the BMS-Controller, which separates the control path and I/O path. Cloud vendors can manage and maintain the local storage through remote management commands, bypassing the host OS. We also design the hot-plug, hot-upgrade, and I/O monitor support to enhance the availability.

We implement BM-Store on a Xilinx® Zynq® UltraScale+™ MPSoC ZU19EG board [40] with Intel P4510 NVMe SSD [2] and deploy BM-Store in production data centers. Our experimental results show that BM-Store can achieve near-native performance and only introduces about 3 µs extra latency and only 4.0% average throughput degradation. BM-Store improves the average bandwidth by 15.7% than SPDK vhost. More importantly, BM-Store achieves a maximum throughput improvement of 13.4% in real-world applications compared to SPDK vhost. The main contributions of this paper are as follows:

- **Novel hardware-assisted local storage architecture for bare-metal clouds.** We propose BM-Store, a novel local storage architecture designed for bare-metal instances. BM-Store achieves transparency, virtualization, high performance, manageability, and high availability, which enables large-scale deployment.
- **Transparency and high compatibility.** BM-Store achieves transparency to enable large-scale deployment on bare-metal clouds regardless of versions of host operating systems or kernels. For high compatibility, BM-Store can support various NVMe devices independent of SSD manufacturers. BM-Store architecture can further support multiple types of devices, including SATA disks, which are vital in local storage.
- **High-performance virtual local storage to bare-metal instances.** BM-Store provides the virtualization capability to local storage in bare-metal instances. The back-end storage resources can be dynamically divided into multiple namespaces for the front-end virtual function. BM-Store adopts the hardware-accelerated I/O path and zero-copy mechanism to achieve extreme performance with minimal overhead. BM-Store also adopts the QoS mechanism to ensure the isolation of virtual local storage.
- **Manageability and high availability.** BM-Store is the first to implement an out-of-band management mechanism based on MCTP over PCIe, enabling local storage management and maintenance for bare-metal clouds. Furthermore, BM-Store offloads management and maintenance functions to the embedded ARM SoC, which enhances the availability of local storage services based on experience in production environments.

II. Background

A. Bare-metal Instances in Cloud Computing

The bare metal is an essential cloud instance that leases the physical machines to tenants rather than virtual machines. Virtual machine tenants can only access virtual resources defined by cloud vendors. In contrast, bare-metal tenants can access the hardware resources directly and install their own host operating systems and applications. The bare-metal instances are well suited for resource-intensive applications such as big data applications, database servers, and media servers. Generally, cloud vendors provide local storage and remote storage to bare-metal instances. Among them, local storage is the first choice for I/O intensive applications due to its high throughput and low latency.

In bare-metal instances, cloud vendors have no access to the host OS to install dedicated software. For data security, privacy, and compatibility concerns, tenants are usually unwilling to install customized drivers from cloud vendors. In such a case, cloud vendors can only provide physical hardware-level services independent of the host OS.

B. NVM Express

NVM Express (NVMe) [30] is a high-performance interface standard for accessing local non-volatile memory devices over

PCIe [5] bus. The NVMe protocol proposes a large number of deep and paired Submission Queues (SQ) and Completion Queues (CQ) for interacting between the NVMe driver and the NVMe controller, which fully exploits the potential of the non-volatile memory device. Due to their vastly superior performance in terms of both bandwidth and latency compared to SATA-SSDs and SAS-SSDs, NVMe SSDs are widely chosen in cloud data centers to cater to the ever-increasing demands of I/O intensive applications.

C. Storage Virtualization

1) Software-based Virtualization: The existing *software-based* storage virtualization enables storage device sharing and presents virtual devices among multiple VMs. *Full virtualization* is a software-based virtualization mechanism that adopts the trap-and-emulate approach to provide virtual devices to VMs without modifying the guest OSes. Compared to full virtualization, *paravirtualization* creates efficient virtual device interfaces between guest OSes and hypervisors.

To virtualize emerging high-performance storage devices, some *polling-based approaches* are proposed to dedicate multiple CPU cores for device emulation and completion polling [32, 42]. These approaches achieve better performance than previous methods by eliminating VM exits and minimizing CPU context switches. However, the polling-based methods consume a large number of computing resources to execute their polling-based device emulation [24, 27].

2) Hardware-assisted Virtualization: To reduce the performance overhead and host CPU inefficiency of the software-based virtualization mechanism, direct pass-through [6] is proposed to enable VMs to access storage devices directly without software intervention, achieving near-native performance. Direct Pass-through allows direct assignment of PCIe storage devices through IOMMU, with its DMA and interrupt remapping mechanisms. However, it loses device sharing capability among multiple VMs. The physical device must be exclusively assigned to a single VM.

To address such shortcomings, PCIe SR-IOV [13] is proposed to share a physical device among many VMs at the hardware level. An SR-IOV capable device presents multiple physical functions (PFs) and virtual functions (VFs) at the PCIe level. SR-IOV can enable resource isolation to serve multiple VMs that each PF/VF has its own PCIe configuration registers. SR-IOV capable devices enable hardware virtualization through their internal bridge module and do not rely on any host software.

III. Motivation

We reviewed state-of-the-art solutions in virtual local storage and I/O accelerators, as shown in Table I. However, the existing local storage solutions have several limitations in bare-metal instances in terms of performance, host efficiency, compatibility, transparency, performance, deployability, and manageability. These factors motivate us to propose BM-Store.

TABLE I
Features of existing local storage techniques

	Mdev [32]	SPDK vhost [42]	SR-IOV [13]	LeapIO [27]	FVM [24]	***BM-Store***
Host efficiency			✓	✓	✓	✓
Compatibility	✓	✓		✓	✓	✓
Transparency			✓			✓
Performance	✓	✓	✓		✓	✓
Deployability	✓	✓	✓			✓
Mangeability						✓

A. Host CPU inefficiency for software storage virtualization

Existing advanced polling-based software solutions [32, 42] rely on dedicated host CPU cores to emulate virtual NVMe devices. However, they suffer from high CPU consumption and performance degradation in practice, as shown in Fig. 1. As storage virtualization I/O tasks are inherently I/O- and control-bound, they cannot make use of the full power of super-scalar, out-of-order general-purpose CPU architecture.

In fact, the limited host CPU cores could be used to provide more computing resources. Then bare-metal tenants will significantly benefit from the extra host CPU cores. This motivates us to offload the local storage virtualization functions to the BMS-Engine to enable host CPU efficiency.

B. Disadvantages of the existing hardware-assisted design

Performance degradation. Hardware-assisted LeapIO [27] offloads the entire storage stack to the ARM SoC with the support of both local and remote storage. However, it suffers from severe performance degradation that it only achieves 68% [24] throughput of the single native disk due to the limited computing capabilities of ARM CPU. Hence, BM-Store offloads the I/O path to the FPGA for high performance.
Lack of compatibility. Existing hardware-assisted solution SR-IOV [13] can solve CPU inefficiency problems. However, only a few SR-IOV capable NVMe SSDs can be chosen since SR-IOV is not yet a standard for NVMe. Thus cloud vendors cannot exploit various local storage devices [4, 30] independent of manufacturers. Hence, SR-IOV capable devices lack compatibility with existing storage devices. At the same time, SR-IOV capable devices also have management and maintenance obstacles, which are discussed in Section III-C.
Low deployability. FVM [24] and LeapIO [27] adopts P2P architecture. They need modified host/guest OS, QEMU, and inevitable customized drivers for initialization and configuration, which are not transparent to hosts. Tenants are usually unwilling to install vendor-specific drivers that would cause concerns about data security, privacy, and compatibility. Therefore, FVM etc. can hardly be deployed on bare-metal instances due to cloud vendors cannot access the host OS.

To make BM-Store transparent and deployable on bare-metal instances, we choose *direct-attached architecture* instead of the PCIe peer-to-peer-based solution.

C. Management and availability Challenges

Existing solutions do not solve manageability and the availability challenges of local storage services in production.

Cloud vendors traditionally run management and configuration programs (kernel modules or user-level applications) in host operating systems. However, it is tough to deploy these schemes on bare-metal instances.

The availability of local storage services is one of the priorities for cloud vendors, which ensures the infrastructure can continue to function even if a component fails. For example, firmware upgrades and faulty device replacements are common for cloud vendors to maintain physical devices but usually have to disrupt local storage services. Cloud vendors have to minimize the impact on tenants, such as not interrupting tenant I/O. In bare-metal instances, cloud vendors cannot adopt previous approaches.

Thus, these challenges motivate BM-Store to implement the out-of-band management mechanism based on MCTP over PCIe with enhanced availability (hot-plug, hot-upgrade, and I/O monitor) for local storage in bare-metal instances.

Summary. Cloud vendors need a host-efficient, compatible, transparent, high-performance, and manageable local storage solution. Nevertheless, we did not find a feasible local storage solution for bare-metal instances that meets the cloud vendors' requirements. BM-Store addresses all the above challenges that can be large-scaled deployed in bare-metal clouds.

IV. BM-Store Design and Implementation

To deploy virtual local storage to bare-metal instances, cloud vendors have strong demands for transparent, high-performance, manageable, and high-availability local storage solution, which enables deployment on a large scale in cloud computing centers. To this end, we set the following design goals to resolve local storage challenges in bare-metal clouds.

Host-efficient: The local storage scheme in bare-metal clouds should be host-efficient which frees up the valuable host CPU resources to provide more computing power to bare-metal tenants.

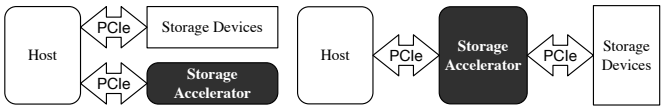
Transparent and high compatibility: It should be transparent to the host OS, enabling large-scale deployment on bare-metal clouds. It also should be compatible to support existing various types of storage devices to provide local disks (e.g., NVMe SSDs and SATA HDDs).

Virtualization and high performance: More importantly, it should provide the local storage with high-performance virtualization capability for bare-metal instances.

Manageability and high availability: It should enable cloud vendors to manage local storage even if they cannot access the operating system in bare-metal instances. Meanwhile, it should enhance local storage service availability.

A. Design Choices for transparency

Architectural transparency is crucial for deploying hardware virtual local storage solutions on bare-metal clouds. Fig. 2 shows two design options for hardware local storage architecture. In P2P architecture (Fig.2(a)), the storage accelerator and the storage devices are both connected to the host through the PCIe bus. In such a case, the accelerator communicates with the storage devices through the PCIe P2P mechanism, which requires customized drivers in the host OS. Many previous hardware solutions [24, 27] adopts the P2P-based architecture but can hardly be deployed on bare-metal clouds.



(a) P2P Architecture (b) Direct-attached Architecture

Fig. 2. Architecture design options of hardware local storage architecture: (a) Peer-to-peer-based architecture (b) Device direct-attached architecture.

In this work, BM-Store adopts the architecture that *direct-attached architecture* as shown in Fig.2(b). To achieve transparency, BM-Store adopts a standard SR-IOV layer in the BMS-Engine and an out-of-band management mechanism based on MCTP over PCIe in the BMS-Controller.

Standard SR-IOV layer. BM-Store exposes standard NVMe interfaces to the host operating system through an SR-IOV layer in the BMS-Engine. Therefore, applications or VMs on the bare-metal instance can access storage resources by the standard NVMe driver.

MCTP out-of-band management. In-band management [9] is the common way to manage and maintain the device. The host needs to be involved in the device management and configuration process, requiring drivers or software to be installed. Therefore, the in-band management cannot handle the *bare-metal instances* because tenants own the physical resources and host OS entirely. In BM-Store, we designed an out-of-band mechanism and implemented MCTP over PCIe on the ARM co-processor, which bypasses the host OS. Our out-of-band management scheme provides fast and efficient management and maintenance of the virtual local storage.

To this end, our BM-Store adopts **a standard SR-IOV layer** and **out-of-band management** to make it transparent to the host. This feature enables BM-Store deployment on bare-metal instances without modifying the host OS. Transparency also facilitates large-scale deployments, disregarding the various versions of the host operating systems and kernels in cloud computing centers.

B. Overall Architecture

BM-Store decouples the front-end (storage interface) and back-end (storage devices) of the local storage. Fig. 3 presents the BM-Store architecture and its components. BM-Store is composed of BMS-Engine and BMS-Controller. BMS-Engine consists of six modules: 1) standard SR-IOV layer to provide standard NVMe interfaces to tenants; 2) target controller to forward I/O and admin commands; 3) I/O mapping module to map address; 4) QoS module to guarantee isolation and fairness; 5) DMA routing module to enable zero-copy; 6) host adaptor to interact with back-end disks. BMS-Controller is responsible for management and maintenance tasks, such as configuration, I/O monitor, hot-plug, and hot-upgrade.

For high performance, we offload the entire I/O path and virtualization functions to the FPGA-based BMS-Engine.

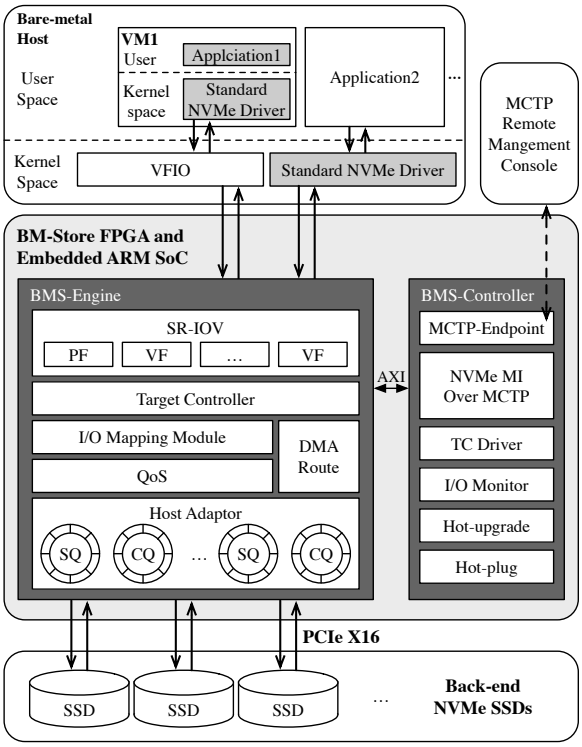
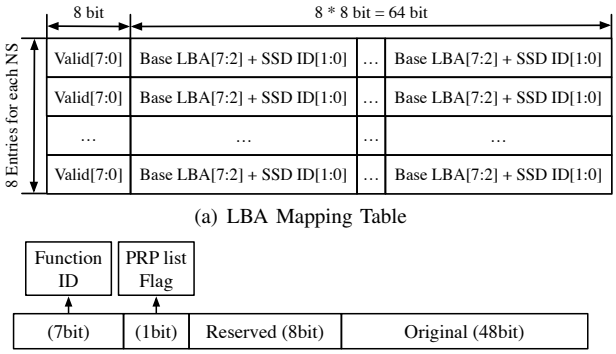


Fig. 3. BM-Store Architecture Overview



(a) LBA Mapping Table

(b) Global PRP Entry

Fig. 4. The Address Format of : (a) LBA Mapping Table (b) Global PRP Entry

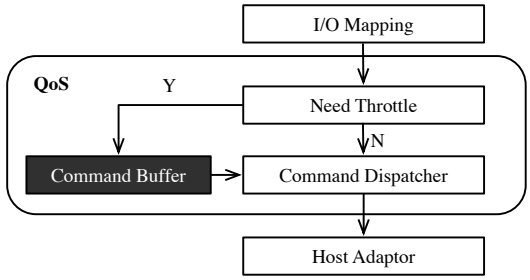


Fig. 5. Quality of Service Procedure

BMS-Engine accelerates LBA address mapping to fully exploit opportunities of parallelization.

We deploy QoS in the BMS-Engine for performance isolation. BMS-Engine also adopts the DMA request routing mechanism to enable zero-copy, eliminating redundant data copies. In this way, bare-metal tenants can fast and directly access the virtual storage resources through NVMe interfaces with near-native performance.

To enable local storage management and maintenance in bare-metal instances, we design an out-of-band management mechanism with enhanced availability on the ARM-based BMS-Controller.

BM-Store is the first work that pays attention to the local storage management in bare-metal instances. We also propose a feasible way to enhance local storage service availability in the production environment. Overall, We propose a hardware virtual local storage solution for bare-metal instances that can be easily deployed on a large scale.

C. Virtualization and Zero-copy I/O Path

BM-Store provides virtualization capability to local storage in bare-metal clouds. BM-Store adopts the FPGA-accelerated virtualization functions for high-performance. We design the *LBA mapping* and *QoS* mechanism on the BMS-Engine to virtualize back-end storage resources. Furthermore, we propose a *DMA requests routing* mechanism to enable direct data transfer between SSDs and host memory.

LBA address mapping. We set up an LBA Mapping Table (MT) in the BMS-Engine to map the host LBA to back-end physical LBA. BMS-Engine supports multi-namespace and resource isolation. We assign eight mapping table entries to each namespace (NS). As shown in Fig. 4(a), each *mapping entry* of MT contains two components: a base LBA (6-bits) and an SSD ID (2-bits). Our MT is a two-dimensional array contains eight rows, and each row contains eight *mapping entrys* and one *validation entry*. The *validation entry* is an 8-bits vector, and each bit $(1/0)$ indicates whether the corresponding *mapping entry* in the same row is valid or invalid. In practice, we divide the back-end SSDs storage space into a chunk of 64 GB. Formally, the address mapping between the Host LBA (HL) and Physical LBA (PL) is defined as:

$$i = (HL \ / \ CS) \ / \ EN \quad (1)$$

$$j = (HL \ / \ CS) \bmod EN \quad (2)$$

$$SSD\_ID = MT_{ij}\ [1:0] \quad (3)$$

$$PL = MT_{ij}\ [7:2] \ * \ CS + HL \bmod CS \quad (4)$$

BMS-Engine can quickly obtain the Mapping Table Entry (MTE) through the Host LBA (HL), Chunk Size (CS), and Entry Number (EN) per row. Then it can get the SSD ID and Physical LBA (PL) of the back-end storage devices from the Mapping Table Entry. Lastly, the BMS-Engine will update the LBA field in the NVMe command and then forward the command to the SQ in the Host Adaptor.

Quality of Service The Quality of Service (QoS) is essential in the cloud for isolation and fairness to prevent performance interference and enable performance isolation. The mechanism of QoS is shown in Fig. 5. We create a command buffer for every NS in BMS-Engine. In performing QoS service, BMS-Engine will determine whether the current I/O performance

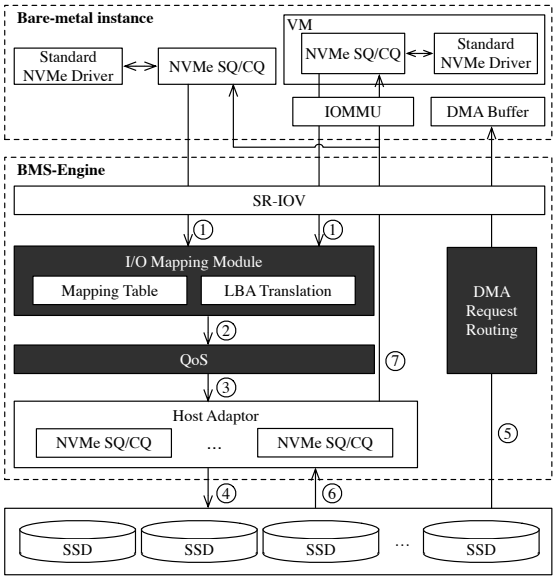


Fig. 6. I/O Request and Data Transfer Procedure

reaches the threshold limit in each NS. If so, it will send the current request into the Command Buffer, and then the Command Dispatcher will reschedule from the buffer.

DMA request routing for zero-copy. BM-Store aims to provide high-performance local storage. However, BM-Store adopts *direct-attached architecture* where the BMS-Engine is connected between the SSD and the host. Typically, the data must be transferred to the FPGA memory and then copied to the host memory. These duplicate data copies will seriously affect I/O performance.

To mitigate performance degradation, we propose the *DMA requests routing* mechanism that enables **zero-copy**. In this way, BM-Store enables direct data transfer between the SSD and the host memory. BMS-Engine does not buffer any data. BMS-Engine converts the host PRP entry to the global PRP entry, denoted as *global PRP*, to route DMA requests. As shown in Fig. 4(b), our *global PRP* utilizes the first 8 bits among the reserved 16 bits of the standard PRP. We set the first seven bits as the PCIe PF/VF device ID (Function ID) and the last one as the PRP list flag. When BMS-Engine fetches a command from the host, it will convert the *host PRP* to BM-Store's *global PRP* by inserting function ID to the host PRP entry and storing the *global PRP* into the chip-memory. When the SSD is ready, the SSD will initiate a TLP message to start the DMA operation. Then the BMS-Engine receives the TLP message, retrieves the function ID from the DMA destination memory address, and routes the request to the corresponding PF/VF. In this way, our BMS-Engine combines separated PCIe domains of the host and back-end devices into a uniform domain, enabling direct data transfer between the SSD and the host memory.

The I/O path. Fig. 6 presents details of the command path in BMS-Engine. We take the process of reading data as an example. In general, the command processing flow of the BMS-Engine has the following seven steps:

① The host submits its I/O command into its SQ as the SQ entry (SQE) and writes the doorbell register in the BMS-Engine. Then, the BMS-Engine fetches the command through the corresponding PF/VF.

② The BMS-Engine converts the *Host LBA* to *Physical LBA* by the LBA address mapping mechanism. After that, the QoS module determines whether the command needs to be put into the command buffer.

③ Then, the BMS-Engine will fetch commands to the SQ in the host adaptor and write the doorbell register to notify the back-end physical SSDs. If there is a PRP list in SQE, BMS-Engine will fetch the PRP list and modify the PRP address in the PRP list. After that, the BMS-Engine will write the doorbell register of the back-end SSD.

④ The SSD receives the signal from the doorbell register and fetches the command from the host adaptor's SQ.

⑤ The SSD initiates a TLP message to BMS-Engine. Then, our DMA request routing mechanism forwards the TLP message to the corresponding PF/VF. Afterward, the SSD successfully writes data into the host memory.

⑥ When the SSD completes data transmission, it writes the complete command (CQE) into the CQ in the host adaptor of the BMS-Engine.

⑦ Finally, after receiving the CQE, BMS-Engine writes the CQE back to the corresponding CQ in the host memory and initiates an interrupt to notify the host.

As a result, the FPGA-based BMS-Engine accelerates the I/O path, enabling BM-Store to provide extreme performance close to native disks.

D. Out-of-band management with enhanced availability

To solve the manageability challenge in bare-metal instances, we design an out-of-band management mechanism based on MCTP over PCIe for local storage.

Out-of-band management. We set the MCTP-endpoint and the NVMe Management Interface (NVMe MI) over the MCTP module in the BMS-Controller to receive management commands from the remote console. The MCTP Endpoint receives management commands from a remote MCTP console. Then, the NVMe MI protocol analyzer parses these commands and sends them to the corresponding modules in the BMS-Controller. The availability of local storage services is essential in cloud computing centers. Firmware upgrades of SSDs and faulty disk replacement are common in cloud computing centers. In traditional ways, these processes need to power off the machines and suspend cloud services, resulting in service interruption. Benefiting from the flexibility of the ARM SoC in the BM-Store architecture, we design I/O monitoring, hot-upgrade, and hot-plug functions for bare-metal instances. These functions significantly enhance the availability of local storage services in BM-Store.

I/O Monitor. We deploy the I/O monitor module in the BMS-Controller. The BMS-Engine monitors I/O status and saves relevant data in specific registers. The I/O monitor module would read the registers to get the I/O status information through the Advanced Extensible Interface (AXI) bus.

TABLE II
FPGA RESOURCE UTILIZATION FOR BM-STORE CONFIGURATION

Design	LUTs	Registers	BRAMs	URAMs	Clock Speed
1 SSDs	216711 (41%)	226309 (22%)	526 (53%)	49.4 (39%)	250MHz
2 SSDs	244711 (47%)	270309 (26%)	570 (58%)	59.4 (46%)	250MHz
4 SSDs	300711 (58%)	358309 (34%)	659 (67%)	79.4 (62%)	250MHz
6 SSDs	356711 (68%)	446309 (43%)	748 (76%)	99.4 (78%)	250MHz

Hot-upgrade. The device firmware upgrade is commonly required in the cloud computing center. In traditional instances, cloud vendors usually access the host OS to upgrade the firmware and reset the SSDs. However, cloud vendors cannot apply the previous method to upgrade the firmware in bare-metal instances because they cannot access the host OS.

Therefore, we adopt a hot-upgrade module on the BMS-Controller to enable the hot-upgrade of the firmware without disrupting local storage services. BMS-Controller notifies the BMS-Engine to store I/O context during firmware upgrading and reloads I/O context after upgrade completion. In this way, we successfully enable the firmware hot-upgrade without interrupting the local storage service.

Hot-plug. To replace faulty storage devices, shutting down the entire server is needed if SSDs do not support hot-plug. It wastes time and affects service availability. Thus BM-Store adopts the hot-plug module in the BMS-Controller for high availability. During device replacing, BM-Store reserves the front-end to the tenants, which means the logic drive identities in the host OS would not disappear. Then BM-Store can replace, reset, and reconnect the back-end SSDs with the front-end NVMe interfaces. Thus, the front-end NVMe devices need not be reset and rescanned. Tenants would benefit from the reserved logical drive of hot-plug that they do not redeploy applications after replacing faulty disks.

Above all, we offload several maintenance functions to the BMS-Controller to enhance local storage service availability bypassing the host OS. The hot-upgrade process only introduces extra I/O latency of a few seconds. For the hot-plug process, it reserves logic drives identities and only needs to pause the I/O but does not need to reboot machines or redeploy applications. BM-Store is a novel local storage architecture specially designed for bare-metal clouds, which supports hot-plug, hot-upgrade, and I/O monitor to enhance service availability.

E. Implementation

We implemented BM-Store on the Xilinx® Zynq® UltraScale+™ MPSoC ZU19EG device and developed BMS-Engine and BMS-Controller on the FPGA and embedded ARM SoC, respectively.

BMS-Engine. We implemented BMS-Engine in Verilog on the FPGA. BMS-Engine consists of six major modules: SR-IOV layer, Target Controller, LBA mapping module, DMA request routing module, QoS module, and host adaptor. BMS-Engine receives and returns host I/O requests through the Target Controller and then forwards requests to back-end SSDs. Specifically, the Target Controller will send general I/O requests to the I/O mapping module and forward the device management commands (admin command) to the BMS-Controller. The mapping information is stored in FPGA on-chip RAMs. We also implemented I/O counting functions in the BMS-Engine, which sends the number of requests to the I/O Monitor to supervise the performance and status of the BM-Store. We present the resource utilization of BM-Store FPGA with two and four SSDs in Table II, and BM-Store can support more SSDs with the remaining resources.

BMS-Engine exposes 4 PFs and 124 VFs to the host at the front-end, which can provide 128 independent NVMe devices in total. The existing BMS-Engine has two PCIe X8 interfaces at the back-end, which can attach four NVMe SSDs. We can increase the number of PCIe interfaces on the BMS-Engine to support more NVMe SSDs in the future.

BMS-Controller. For BMS-Controller on the embedded ARM SoC, we implemented several service modules to support interaction with BMS-Engine, out-of-band management, I/O monitor, hot-upgrade, and hot-plug. The BMS-Controller interacts with the FPGA-based BMS-Engine through the AXI bus. We implemented BMS-Controller with about 15000 LOCs.

TABLE III
DETAILS OF OUR EXPERIMENTAL SETUP

Host	Description
CPU	Intel Xeon Platinum 8163 CPU 2.50GHz
RAM	768GB DDR4
Host OS	CentOS 7.9.2009
VM OS	CentOS 7.9.2009
Kernel Version	3.10.0
SSD	2.0 TB Intel P4510 NVMe SSD

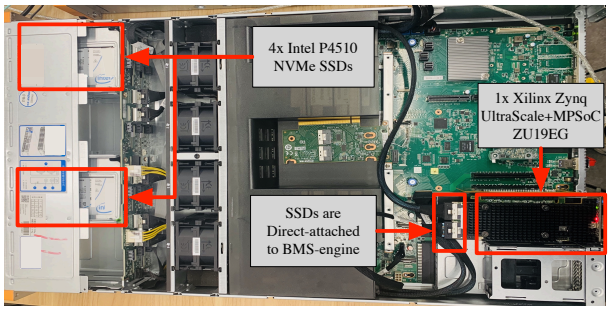


Fig. 7. BM-Store Deployment on Production Server

V. EVALUATION

This section evaluates BM-Store and compares its I/O performance with other local storage schemes through synthetic benchmarks and real-world applications in both bare-metal machines and virtual machines. We also evaluate the availability of BM-Store in terms of the upgrade time and the I/O pause time caused by hot-upgrade.

A. Experimental Setup

System Settings. Fig.7 shows our BM-Store deployed on the production server, and TABLE III presents the detailed

TABLE IV
FIO TEST CASE

Test Case	Description
rand-r-1	4K random read, iodepth=1, numjobs=4
rand-r-128	4K random read, iodepth=128, numjobs=4
rand-w-1	4K random write, iodepth=1, numjobs=4
rand-w-16	4K random write, iodepth=16, numjobs=4
seq-r-256	128K sequential read, iodepth=256, numjobs=4
seq-w-256	128K sequential write, iodepth=256, numjobs=4

TABLE V
AVERAGE LATENCY OF BARE-METAL PERFORMANCE WITH ONE DISK

Average Latency (μs)	Native Disk	BM-Store
rand-r-1	77.2	80.4
rand-r-128	786.7	792.6
rand-w-1	11.6	14.5
rand-w-16	179.8	179.9
seq-r-256	40579.3	40041.3
seq-w-256	92502.3	95030.0

experimental settings for BM-Store. We deploy BM-Store on the host machine with two 24-core Intel Xeon Platinum 8163 CPUs running at 2.5GHz, 768G DDR4 DRAM, and four 2.0TB Intel P4510 NVMe SSDs. We plugged the BM-Store card into the PCIe Gen 3.0 slot on the server and attached 4 NVMe SSDs directly to the BM-Store card. We also disabled hyperthreading and power-saving states for accurate performance measurements on the servers.

Baseline and Compared Methods. We took the performance of the native disks as the baseline and evaluated BM-Store in bare-metal machines. On the other hand, for the virtualization performance, We compared BM-Store with VFIO and SPDK vhost, the mainstream solution for device passthrough and the representative software solutions, respectively.

Benchmarks and Applications We leveraged FIO [1] to generate synthetic I/O workloads. FIO is a powerful tool for testing the performance of storage devices, which has been widely used by industry and academia. In particular, we used libaio as the FIO engine and ran the test cases in TABLE IV, including random and sequential read and write with various iodepth and numjobs. To evaluate BM-Store in the production environment, we adopted TPC-C [39] and Sysbench [22] to drive the MySQL database in the guest virtual machine, and YCSB [12] to drive RocksDB database. Furthermore, we set up the evaluation that mixes YCSB and Sysbench workloads in virtual machines. Finally, we evaluated the availability of BM-Store in terms of the hot-upgrade time and the I/O pause time caused by the hot-upgrade.

B. Bare-metal Performance of Single Disk

To evaluate the I/O performance in bare-metal machines, we ran FIO and measured the throughput and the average latency of BM-Store and the native disk. In BM-Store, We allocated a 1536 GB namespace from a back-end SSD and bound the namespace to the front-end VF.

Fig. 8 shows the IOPS and bandwidth performance of the native disk and BM-Store. For all test cases, BM-Store can achieve performance from 96.2% to 101.4% of native disk

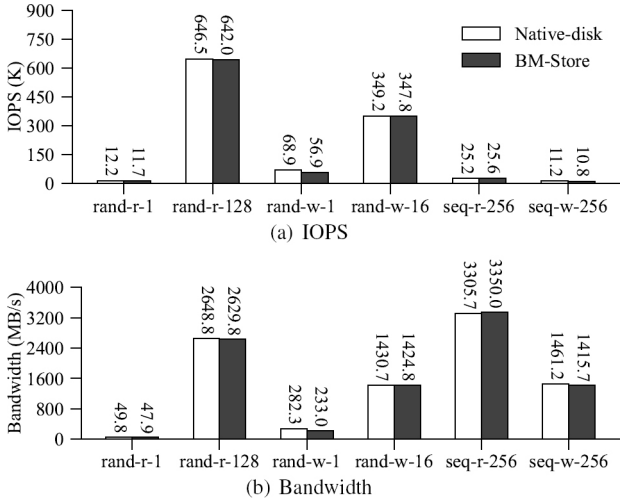


Fig. 8. Bare-metal Performance with 1 disk of Native Disks and BM-Store: (a) IOPS. (b) Bandwidth.

TABLE VI
THE I/O PERFORMANCE OF BM-STORE DEPLOYED ON DIFFERENT PLATFORMS.

OS Version	Kernel Version	IOPS	BW(MB/s)	AL(μs)
CentOS 7.4.1708	3.10.0	642K	2629	394.4
	4.19.127	642K	2629	395.9
	5.4.3	642K	2630	396.1
Fedora 33	4.9.296	603K	2468	207.0
	5.8.15	607K	2487	206.4

except for rand-w-1 (82.5%). TABLE V shows the latency performance of native disk and BM-Store. We can see that BM-Store constantly introduces about 3 us latency overhead due to the longer command path. The overhead is negligible in most cases. Nevertheless, in extremely low latency cases such as rand-w-1, the proportion of overhead is magnified. We observed that BM-Store could provide high-performance local storage thanks to the I/O path accelerated by the BMS-Engine.

BM-Store can be easily deployed on a large scale regardless of operating systems and kernel versions. We evaluate BM-Store on various operating systems and kernels by the 4k random read test case with iodepth=16 and numjobs=8, as shown in TABLE VI. The results show that BM-Store can successfully run on bare-metal instances without any modification to the host and provides stable local storage performance.

C. Virtualization Performance

In this section, we set up the virtual machine environment and evaluated the performance of VFIO, BM-Store, and SPDK vhost with a single disk. For all schemes, we allocated 4 CPU cores and 4GB system memory for the VM. Especially for SPDK vhost, we allocated one extra CPU core for the SPDK vhost virtualization layer.

IOPS and bandwidth performance. The IOPS and bandwidth performance of the three virtualization techniques on a single virtual machine are demonstrated in Fig. 9. We observe that BM-Store can achieve performance from 95.6% to 102.7%

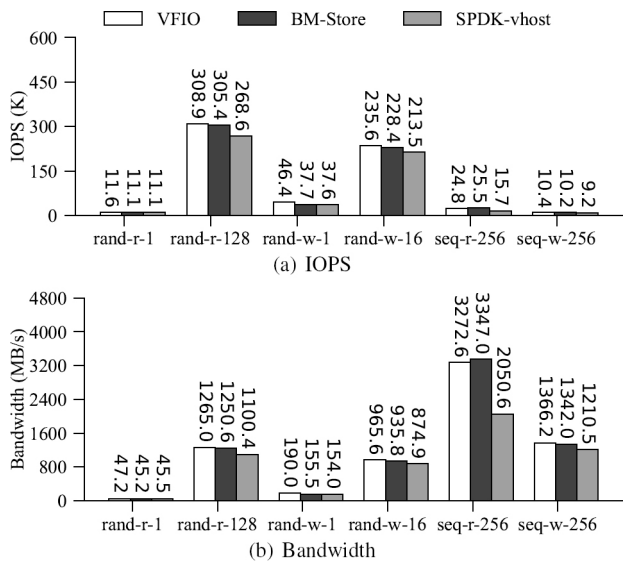


Fig. 9. Single VM Performance with one disk of VFIO, BM-Store, and SPDK vhost: (a) IOPS. (b) Bandwidth.

TABLE VII
AVERAGE LATENCY OF SINGLE VM PERFORMANCE WITH ONE DISK

Average Latency (µs)	VFIO	BM-Store	SPDK vhost
rand-r-1	79.7	83.7	82.7
rand-r-128	1647.0	1666.0	1893.4
rand-w-1	14.9	19.6	19.2
rand-w-16	264.7	275.5	305.3
seq-r-256	40990.4	40075.6	65197.1
seq-w-256	98819.2	100615.0	112245.7

of VFIO except for the rand-w-1 (81.2%). However, SPDK vhost can only achieve from 63.0% to 96.0% of VFIO. In the best case (seq-r-256), the performance of BM-Store is 62.9% higher than that of SPDK vhost. We find that SPDK vhost suffers from severe performance degradation in seq-r-256 in CentOS7 with 3.10.0 kernel, which indicates that SPDK vhost cannot get stable performance in some OS and kernel versions. In contrast, BM-Store is independent of the host, achieving near-native performance. VFIO provides entire storage devices to the virtual machine, which achieves native performance. However, the virtual machine monopolizes the storage device, losing the sharing capability. In BM-Store, the back-end storage disks are divided into multiple namespaces, bounded to PF/VFs, and directly accessed by the virtual machine, achieving high-performance I/O under the premise of namespace split and isolation.

To summarize, the I/O processing accelerated by FPGA can achieve better performance than software solutions. SPDK vhost not only suffered from performance degradation but also consumed 25% more CPU resources than BM-Store.

Latency performance. TABLE VII shows that the average latency performance of BM-Store is better than SPDK vhost in most test cases, indicating that BM-Store's hardware-accelerated path can provide lower latency compared with SPDK's polling-based scheme. We can observe that BM-Store introduces only about 4 µs constant latency in the virtual machine in rand-r-1 and rand-w-1. The I/O path accelerated by

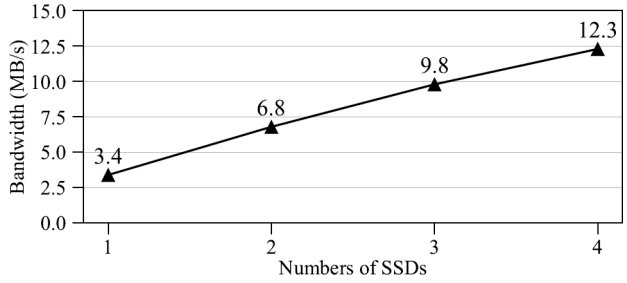


Fig. 10. Total Bandwidth of BM-Store in Different Numbers of SSDs in the Bare-metal Machine.

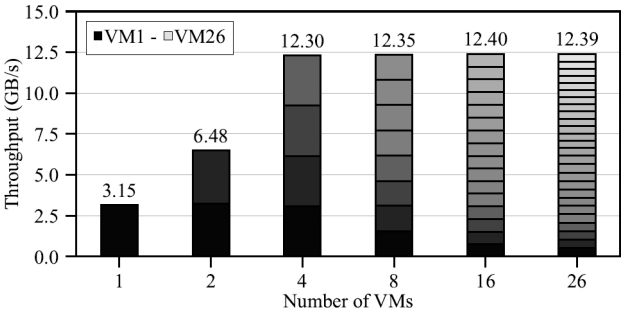


Fig. 11. Total Bandwidth of BM-Store with 4 SSDs in the Multiple Virtual Machines.

BMS-Engine has obvious advantages, obtaining constant extra latency and higher performance with no CPU usage. Under the test case of rand-r-1 and rand-w-1, BM-Store shows slight poor performance compared to SPDK vhost. The latency of VFIO is only 79.7 µs in rand-r-1 and 14.9 µs in rand-w-1. This is because the constant latency of BM-Store accounts for a large proportion in these two test cases. We can observe that BM-Store has a more obvious advantage in handling large I/O requests and deep queue depth. As a result, BM-Store can accelerate virtual local storage while ensuring minimal and constant overhead.

D. Scalability and Fairness

For scalability, we evaluated the bare-metal performance of BM-Store on different numbers of SSDs. First, we tested the total bandwidth of 1 to 4 SSDs in the bare-metal machine.

Then we ran different numbers of VMs on BM-Store with 4 SSDs to evaluate the fairness. We created 1, 2, 4, 8, 16, and 26 VMs, where each VM was assigned with a 256GB namespace in a Round-Robin style from four SSDs. The 26 is the maximum number of virtual machines running in a single server in our production environment. We bind these namespaces to the 26 VFs provided by BM-Store. VMs directly access these VFs to access virtualized local storage.

Scalability of BM-Store. Fig. 10 shows the result of BM-Store in seq-r-256. We can see that the total bandwidth increases linearly with the number of SSDs. BM-Store can saturate 4 SSDs while consuming only half of the FPGA resources (as shown in TABLE II). Therefore, we can conclude that *BM-Store can ensure promising scalability*. Fig. 11 shows the FIO bandwidth result of BM-Store when running multiple VMs on four NVMe SSDs. We can see the total throughput linearly

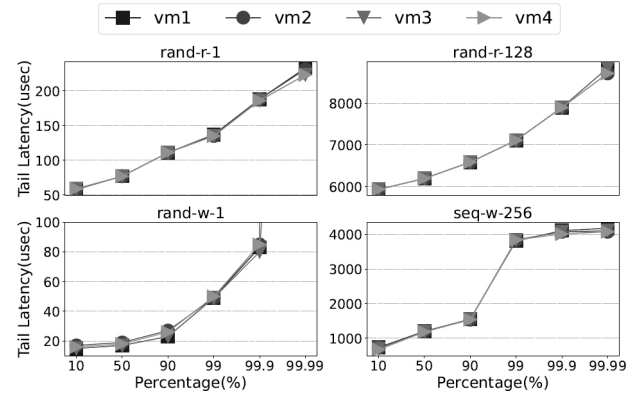


Fig. 12. Tail Latency Distribution of BM-Store under Four Virtual Machines

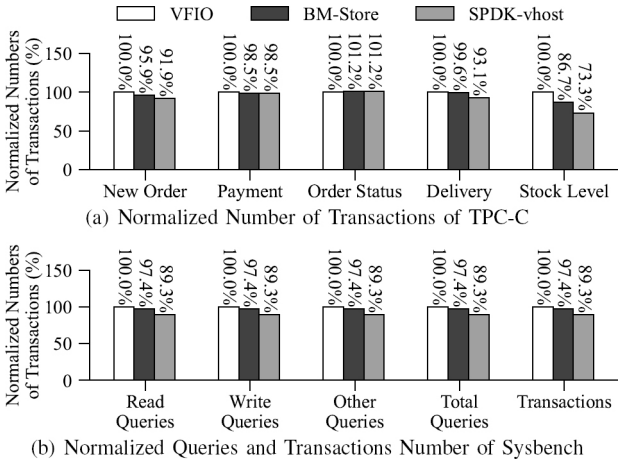


Fig. 13. MySQL Results: (a) TPC-C: Normalized Number of Transactions. (b) Sysbench: Normalized Queries and Transactions Number.

scaling with the number of the VMs. With 16 VMs, BM-Store achieves up to 12.40 GB/s, reaching the maximum bandwidth of four P4510 SSDs.

Fairness between multiple VMs. Fig. 11 shows that BM-Store achieves the balanced bandwidth allocation among VMs without any performance loss. We also demonstrated the tail latency of four virtual machines in Fig. 13. We can see that the distribution of the tail latency of each virtual machine is close to each other in all test cases. The results demonstrate that BM-Store can maintain the fairness of each virtual machine as well as the overall performance of I/O and prevent the host side resources from tilting to some virtual machines.

E. Application

To evaluate the performance of BM-Store in the production environment, we selected MySQL and RocksDB as real-world applications. We also deployed native disks through VFIO, SPDK vhost, and BM-Store to provide local storage to them. We installed MySQL-5.7.35 and RocksDB-6.4.6 on virtual machines. To generate workloads, we deployed TPC-C and Sysbench for MySQL, and YCSB for RocksDB.

TPC-C benchmark. For TPC-C, we configured the relevant parameters, created test databases in MySQL, and filled the database with data. The warehouse number in TPC-C is set

TABLE VIII
AVERAGE LATENCY RESULT OF SYSBENCH

	VFIO	BM-Store	SPDK vhost
Average Latency (ms)	8.32	8.54	9.32
Extra Overhead	0	2.6%	11.2%

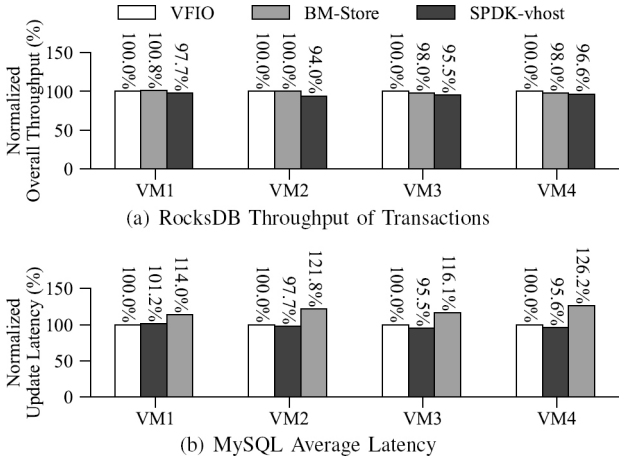


Fig. 14. Mix Workloads Results: (a) RocksDB Throughput of Transactions. (b) MySQL Average Latency

to 100. Meanwhile, we set the number of concurrent threads to 32. After warming up for 300s, we started the test for 60 minutes, with the corresponding results shown in Fig. 13(a).

We observed that BM-Store could reach near-native performance when serving TPC-C workloads. Compared to the baseline, BM-Store outperforms 13.4% higher transactions than SPDK vhost in the best case, indicating that BM-Store architecture provides high-performance I/O virtualized local storage services for applications in virtual machines.

Sysbench benchmark. Then, we ran OLTP workloads of Sysbench on MySQL for 120s. As shown in TABLE VIII, BM-Store architecture incurs only 2.6% extra latency compared to native disks with VFIO. In contrast, SPDK vhost incurs 11.2% more latency. When comparing the numbers of queries and transactions, as shown in Fig. 13(b), the performance of the BM-Store architecture is only 2.59% lower than that of the native disks. Compared to the baseline, it is 8.1% better than SPDK vhost in all query cases.

Mixed workload in multiple VMs. For better evaluating the BM-Store in the production environment, we implemented Sysbench and YCSB to generate mixed workloads in multiple VMs and took MySQL and RocksDB as storage applications, respectively. From the results in Fig. 14, we can see that BM-Store achieves near-native performance even under complex workloads. The consistent performance advantages of different VMs reveal that BM-Store guarantees extreme performance and excellent isolation among multiple VMs.

F. Availability

In this section, we evaluated the enhanced availability of BM-Store in terms of the hot-upgrade time and the I/O pause time caused by the hot-upgrade. We ran FIO and performed twice hot-upgrades of the SSD's firmware during the I/O

TABLE IX
AVERAGE TIME FOR HOT-UPGRADE OF SSD'S FIRMWARE

	Virtual Machine		Bare Metal	
Test Case	rand-r	rand-w	rand-r	rand-w
I/O Pause Time (s)	6.2	8.8	6.2	9.2

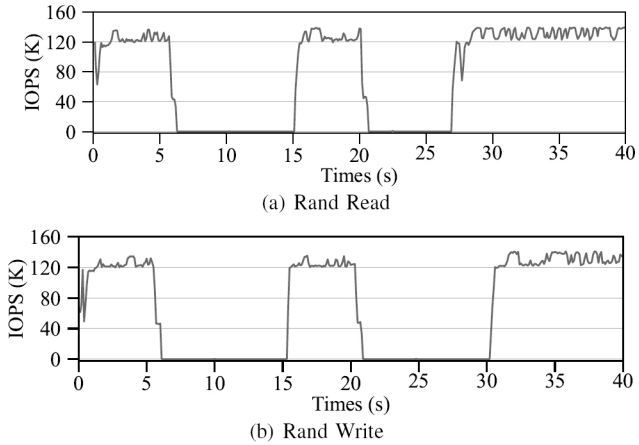


Fig. 15. The IOPS Performance of Virtual Machine When do Hot-upgrade of SSD's Firmware: (a) Rand Read. (b) Rand Write.

operations. In particular, we ran the test case of random read/write with 4K block size. Then we record the IOPS of FIO and show the result in Fig.15. Meanwhile, we calculated the I/O pause time and presented the results in TABLE IX. In BM-Store, the total time for SSD firmware hot-upgrade is about 6-9 seconds. Among them, the processing time of BM-Store is about 100ms, and the remaining part depends on the time of the SSD firmware upgrade. During the hot-upgrade process, tenants do not need to stop I/O operations and will not receive I/O errors. This is because BM-Store can complete SSD's firmware hot-upgrade before I/O timeout. Therefore, BM-Store can provide more availability for local storage services in the production environment.

VI. DISCUSSION

A. Compatibility of BM-Store Architecture

BM-Store can further easily support various storage devices such as SATA HDDs and ZNS SSDs [3]. For example, to support SATA HDD, no modification is needed to the hardware architecture due to the programmability of the BMS-Engine and BMS-Controller. First, we have to add the logic of the SATA controller to the Host Adaptor in BMS-Engine to implement interfaces for SATA HDDs. Then we should develop a module in BMS-Controller to process SATA protocol and management. As such, BM-Store is flexible to support SATA HDDs. BM-Store can also easily be deployed on the traditional virtual machine instances because BM-Store is transparent to the host. Virtual machine tenants will also benefit from the high performance and cost-efficiency of BM-Store.

Hence, BM-Store architecture is compatible with both storage devices and existing host systems. BM-Store architecture can provide virtual local storage with a standard NVMe interface independent of the types of back-end storage devices. Moreover, BM-Store is independent of the host system and OS, which is transparent to tenants.

B. Experience and Lessons

We want to share some practical lessons from the development of BM-Store.

First, TCO is one of the critical factors in choosing solutions for cloud vendors. In particular, TCO analysis includes too many variables, such as raw hardware cost, power consumption, IDC cost, sellable resources per server, compatibility with prior and subsequent architecture, server configuration, etc. Adopting an FPGA in the initial stages (deployment scale less than 10000) allows us to develop and optimize BM-Store quickly. Therefore, we can benefit from the flexibility of FPGA to quickly get stable performance and verify the effectiveness of BM-Store architecture. As the development scale grows, we will consider ASIC to further reduce hardware costs.

Second, hot-upgrade and management capabilities in cloud services are as essential as service performance. Cloud tenants are extremely sensitive to the availability of cloud services. Therefore, BM-Store solves the problem of local storage management in bare-metal instances and enhances the availability of services.

Third, industry standards represent a technological trend. However, we encountered many problems and bottlenecks in the actual implementation, which required many iterations and efforts to adapt to the practical application. For example, the MCTP over PCIe adopted in BM-Store enables out-of-band local storage management in bare-metal instances. However, we found issues with the stability and performance of MCTP. To enable BM-Store deployment in the production environment, we optimized the protocol to improve performance and stability and applied it for related patents.

C. TCO Analysis

We calculated the TCO of our solution and confirmed that the deployment of BM-Store can reduce at least 11.3% TCO because we can sell 14.3% more instances per server. In our deployment, the cost of a single server increased 3% from BM-Store hardware, where each server is equipped with 4 BM-Store hardware and 16 NVMe SSDs. Our typical server configuration is 128 HT/1024 GB Memory/16 SSDs. Adopting SPDK vhost scheme inevitably faces the problem of unused resource fragments (128GB/2 SSDs) due to 16 dedicated CPU cores for polling. BM-Store can eliminate these fragments and sell two more instances (8 HT/64 GB Memory/1 SSD) with these fragments than the SPDK solution.

D. Future Works

BM-Store provides high-performance virtual local storage and high compatibility to the existing system that is transparent to the host. Due to the programmability of BM-Store architecture, we can easily further offload any storage functions to BMS-Engine and BMS-Controller, providing various storage services without host CPU usage. While BM-Store currently focuses on local storage virtualization, we plan to add remote

storage support to cope with more storage scenarios. At the same time, we are working hard to deploy BM-Store in one of the largest public cloud data centers to provide high-performance virtualized local storage to bare-metal clouds.

VII. Related Work

Bare-metal Clouds. Most public cloud vendors have provided bare-metal instances [8, 11, 29] and many previous works [26, 31, 43] focus on optimizing bare-metal services. BMcast [31] improves the agility and elasticity in bare-metal clouds. BM-Hive [43] provides a multi-tenants bare-metal architecture to enable secure, flexible, and interoperable bare-metal cloud service. To better evaluate big data processing on bare-metal clouds, Lee et al. [26] provide performance analysis on different bare-metal platforms.

Local Storage Virtualization. Virtio [34] is a set of virtualized devices for general I/O devices, but it suffers from severe performance degradation of device simulation. Yang et al. [41] analyze the overhead caused by interrupts in the NVMe protocol when accessing high-speed storage devices and propose that using polling instead of the interrupt can significantly improve storage performance. SPDK [42] adopts a polling mechanism, which provides a user-mode efficient I/O interface, and it provides a vhost acceleration method called SPDK vhost for accelerating Virtio-scsi or Virtio-blk. Similarly, MDev-NVMe [32] proposes a mediated passthrough mechanism with an active polling mode. To mitigate host CPU inefficiency, FVM [24] offloads NVMe virtualization to the hardware, which is a fast and scalable solution. However, it needs customized drivers, which is tough for bare-metal instances and large-scale deployment.

Software stack offloading and near-storage processing. As the speed of accelerators such as FPGA continues to increase, offloading I/O stack and I/O-related computations becomes popular, which is host CPU efficient. LeapIO [27] offloads the entire storage stack for local and remote storage to the ARM SoC, but it suffers from severe performance degradation. Linefs [19] offloads part of the distributed file system to the idle SmartNIC. Recent near-storage processing studies [14, 16, 17, 21, 33, 35, 37] proposed flexible overlay FPGA architectures or offloaded part of computing tasks to the smart SSDs and the SSD controllers, which enhanced the usability of computational storage devices. DCS [7, 23] and Lynx [38] offloads the server data and control plane to the accelerator and enables direct communication between accelerators and the smart-NIC efficiently. These works optimize the I/O performance while greatly reducing the pressure on the CPU.

VIII. Conclusion

We propose BM-Store, a novel transparent and high-performance virtual local storage architecture for bare-metal clouds. BM-Store adopts the device direct-attached to the BMS-Engine and the out-of-band management on the BMS-Controller to make BM-Store transparent to the host. We offload the I/O path to the FPGA-based BMS-Engine to provide high-performance virtual local storage. We also integrate management and maintenance functions in BMS-Controller to enhance the availability of BM-Store in the production environment. BM-Store does not rely on any custom drivers, so other cloud vendors can benefit from our design and experience presented in this paper.

IX. Acknowledgment

This research was supported by the key research and development plan of Zhejiang province (Grant No.2021C03140). This research was also supported by the advanced computing resources provided by the Supercomputing Center of Hangzhou City University. Chengkun Wei and Wenzhi Chen are co-corresponding authors.

References

[1] "Flexible i/o tester," https://github.com/axboe/fio.git, 2021.

[2] "Intel ssd dc p4510 series," https://ark.intel.com/content/www/us/en/ark/products/series/122570/intel-ssd-dc-p4510-series.html, 2021.

[3] "Nvme zoned namespaces," https://zonedstorage.io/docs/introduction, 2021.

[4] "Sata data storage protocol," https://www.intel.com/content/www/us/en/programmable/solutions/technology/transceiver/protocols/pro-sata-sas.html, 2021.

[5] T. A, "Storage and pci express a natural combination," http://www.avagotech.com/applications/datacenters/enterprise-storage, 2016.

[6] D. Abramson, J. Jackson, S. Muthrasanallur, G. Neiger, G. Regnier, R. Sankaran, I. Schoinas, R. Uhlig, B. Vembu, and J. Wiegert, "Intel virtualization technology for directed i/o." *Intel technology journal*, vol. 10, no. 3, 2006.

[7] J. Ahn, D. Kwon, Y. Kim, M. Ajdari, J. Lee, and J. Kim, "Dcs: a fast and scalable device-centric server architecture," in *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2015, pp. 559–571.

[8] AWS, "Amazon ec2 m5.metal instances," https://aws.amazon.com/ec2/instance-types/m5/, 2022.

[9] M. Canini, I. Salem, L. Schiff, E. M. Schiller, and S. Schmid, "A self-organizing distributed and in-band sdn control plane," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 2656–2657.

[10] A. Cloud, "Local disks in elastic compute service of alibaba cloud," https://www.alibabacloud.com/help/en/elastic-compute-service/latest/local-disks, 2022.

[11] A. Cloud, "Alibaba cloud elastic compute service (ecs) bare metal instance families," https://www.alibabacloud.com/help/en/elastic-compute-service/latest/ecs-bare-metal-instance-types-overview, 2022.

[12] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with

ycsb," in *Proceedings of the 1st ACM symposium on Cloud computing*, 2010, pp. 143–154.

[13] P. S. I. Group, "Welcome to pci-sig," https://pcisig.com/, 2020.

[14] B. Gu, A. S. Yoon, D.-H. Bae, I. Jo, J. Lee, J. Yoon, J.-U. Kang, M. Kwon, C. Yoon, S. Cho *et al.*, "Biscuit: A framework for near-data processing of big data workloads," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 153–165, 2016.

[15] Z. Guz, H. Li, A. Shayesteh, and V. Balakrishnan, "Nvme-over-fabrics performance characterization and the path to low-overhead flash disaggregation," in *Proceedings of the 10th ACM International Systems and Storage Conference*, 2017, pp. 1–9.

[16] Y. Jin, H.-W. Tseng, Y. Papakonstantinou, and S. Swanson, "Kaml: A flexible, high-performance key-value ssd," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2017, pp. 373–384.

[17] Y. Kang, Y.-s. Kee, E. L. Miller, and C. Park, "Enabling cost-effective data processing with smart ssd," in *2013 IEEE 29th symposium on mass storage systems and technologies (MSST)*. IEEE, 2013, pp. 1–12.

[18] A. Khatri, "Offline gaming vs cloud gaming (online gaming)," *National Journal of System and Information Technology*, vol. 11, no. 2, p. 99, 2018.

[19] J. Kim, I. Jang, W. Reda, J. Im, M. Canini, D. Kostić, Y. Kwon, S. Peter, and E. Witchel, "Linefs: Efficient smartnic offload of a distributed file system with pipeline parallelism," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, 2021, pp. 756–771.

[20] A. Klimovic, H. Litz, and C. Kozyrakis, "Reflex: Remote flash = local flash," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 1, pp. 345–359, 2017.

[21] G. Koo, K. K. Matam, I. Te, H. K. G. Narra, J. Li, H.-W. Tseng, S. Swanson, and M. Annavaram, "Summarizer: trading communication with computing near storage," in *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2017, pp. 219–231.

[22] A. Kopytov, "Sysbench: a system performance benchmark." https://github.com/akopytov/sysbench, 2021.

[23] D. Kwon, J. Ahn, D. Chae, M. Ajdari, J. Lee, S. Bae, Y. Kim, and J. Kim, "Dcs-ctrl: a fast and flexible device-control mechanism for device-centric server architecture," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 491–504.

[24] D. Kwon, J. Boo, D. Kim, and J. Kim, "{FVM}: Fpga-assisted virtual device emulation for fast, scalable, and flexible storage virtualization," in *14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20)*, 2020, pp. 955–971.

[25] D. Kwon, W. Lee, D. Kim, J. Boo, and J. Kim, "Smartfvm: A fast, flexible, and scalable hardware-based virtualization for commodity storage devices," *ACM Transactions on Storage (TOS)*, vol. 18, no. 2, pp. 1–27, 2022.

[26] H. Lee and G. Fox, "Big data benchmarks of high-performance storage systems on commercial bare metal clouds," in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. IEEE, 2019, pp. 1–8.

[27] H. Li, M. Hao, S. Novakovic, V. Gogte, S. Govindan, D. R. Ports, I. Zhang, R. Bianchini, H. S. Gunawi, and A. Badam, "Leapio: Efficient and portable virtual nvme storage on arm socs," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 591–605.

[28] P. Liu, J. Yoon, L. Johnson, and S. Banerjee, "Greening the video transcoding service with low-cost hardware transcoders," in *2016 {USENIX} Annual Technical Conference ({USENIX}{ATC} 16)*, 2016, pp. 407–419.

[29] Mircosoft, "Baremetal infrastructure on azure," https://docs.microsoft.com/en-us/azure/baremetal-infrastructure/concepts-baremetal-infrastructure-overview, 2022.

[30] NVMEXPRESS, "Nvm express specification," http://www.nvmexpress.org/specifications/, 2011.

[31] Y. Omote, T. Shinagawa, and K. Kato, "Improving agility and elasticity in bare-metal clouds," in *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2015, pp. 145–159.

[32] B. Peng, H. Zhang, J. Yao, Y. Dong, Y. Xu, and H. Guan, "Mdev-nvme: A nvme storage virtualization solution with mediated pass-through," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, 2018, pp. 665–676.

[33] Z. Ruan, T. He, and J. Cong, "{INSIDER}: Designing in-storage computing system for emerging high-performance drive," in *2019 {USENIX} Annual Technical Conference ({USENIX}{ATC} 19)*, 2019, pp. 379–394.

[34] R. Russell, "virtio: towards a de-facto standard for virtual i/o devices," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, pp. 95–103, 2008.

[35] R. Schmid, M. Plauth, L. Wenzel, F. Eberhardt, and A. Polze, "Accessible near-storage computing with fpgas," in *Proceedings of the Fifteenth European Conference on Computer Systems*, 2020, pp. 1–12.

[36] G. Soundararajan, M. Mihailescu, and C. Amza, "Context-aware prefetching at the storage server." in *USENIX Annual Technical Conference*, 2008, pp. 377–390.

[37] D. Tiwari, S. Boboila, S. Vazhkudai, Y. Kim, X. Ma, P. Desnoyers, and Y. Solihin, "Active flash: Towards energy-efficient, in-situ data analytics on extreme-scale machines," in *11th {USENIX} Conference on File and Storage Technologies ({FAST} 13)*, 2013, pp. 119–132.

[38] M. Tork, L. Maudlej, and M. Silberstein, "Lynx: A smartnic-driven accelerator-centric architecture for network servers," in *Proceedings of the Twenty-Fifth Inter-*

national Conference on Architectural Support for Programming Languages and Operating Systems, 2020, pp. 117–131.

[39] TPC, "Tpc-c, an on-line transaction processing benchmark," http://www.tpc.org/tpcc/, 2021.

[40] Xilinx, "Zynq ultrascale+ mpsoc: Heterogeneous multiprocessing platform for broad range of embedded applications," https://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html, 2021.

[41] J. Yang, D. B. Minturn, and F. Hady, "When poll is better than interrupt." in *FAST*, vol. 12, 2012, pp. 3–3.

[42] Z. Yang, C. Liu, Y. Zhou, X. Liu, and G. Cao, "Spdk vhost-nvme: accelerating i/os in virtual machines on nvme ssds via user space vhost target," in *2018 IEEE 8th International Symposium on Cloud and Service Computing (SC2)*. IEEE, 2018, pp. 67–76.

[43] X. Zhang, X. Zheng, Z. Wang, H. Yang, Y. Shen, and X. Long, "High-density multi-tenant bare-metal cloud," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 483–495.