From Luna to Solar: The Evolutions of the Compute-to-Storage Networks in Alibaba Cloud

Rui Miao*, Lingjun Zhu*, Shu Ma, Kun Qian, Shujun Zhuang, Bo Li, Shuguang Cheng, Jiaqi Gao, Yan Zhuang, Pengcheng Zhang, Rong Liu, Chao Shi, Binzhang Fu, Jiaji Zhu, Jiesheng Wu, Dennis Cai, Hongqiang Harry Liu

Alibaba Group

*Co-first authors

ABSTRACT

This paper presents the two generations of storage network stacks that reduced the average I/O latency of Alibaba Cloud's EBS service by 72% in the last five years: LUNA, a user-space TCP stack that corresponds the latency of network to the speed of SSD; and SOLAR, a storage-oriented UDP stack that enables both storage and network hardware accelerations.

LUNA is our first step towards a high-speed compute-to-storage network in the "storage disaggregation" architecture. Besides the tremendous performance gains and CPU savings compared with the legacy kernel TCP stack, more importantly, it teaches us the necessity of offloading both network and storage into hardware and the importance of recovering instantaneously from network failures.

SOLAR provides a highly reliable and performant storage network running on hardware. For avoiding hardware's resource limitations and offloading storage's entire data path, SOLAR eliminates the superfluous complexity and the overfull states from the traditional architecture of the storage network. The core design of SOLAR is unifying the concepts of network packet and storage data block – each network packet is a self-contained storage data block. There are three remarkable advantages to doing so. First, it merges the packet processing and storage virtualization pipelines to bypass the CPU and PCIe; Second, since the storage processes data blocks independently, the packets in SOLAR become independent. Therefore, the storage (in hardware) does not need to maintain receiving buffers for assembling packets into blocks or handling packet reordering. Finally, due to the low resource requirement and the resilience to packet reordering, SOLAR inherently supports large-scale multi-path transport for fast failure recovery. Facing the future, SOLAR demonstrates that we can formalize the storage virtualization procedure into a P4-compatible packet processing pipeline. Hence, SOLAR's design perfectly applies to commodity DPUs (data processing units).

CCS CONCEPTS

• Networks → Network protocol design; **• Information systems → Cloud based storage**;

KEYWORDS

Storage Network; In-network Acceleration; Data Processing Unit

ACM Reference Format:

Rui Miao*, Lingjun Zhu*, Shu Ma, Kun Qian, Shujun Zhuang, Bo Li, Shuguang Cheng, Jiaqi Gao, Yan Zhuang, Pengcheng Zhang, Rong Liu, Chao Shi, Binzhang Fu, Jiaji Zhu, Jiesheng Wu, Dennis Cai, Hongqiang Harry Liu. 2022. From Luna to Solar: The Evolutions of the Compute-to-Storage Networks in Alibaba Cloud. In *ACM SIGCOMM 2022 Conference (SIGCOMM '22), August 22–26, 2022, Amsterdam, Netherlands.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3544216.3544238

1 INTRODUCTION

Elastic Block Storage (EBS) is a fundamental service that provides persistent data hosting in virtualized disks (VDs) to cloud users [4, 5, 7, 8]. It has to be highly reliable (*e.g.,* "nine 9s" for data integrity [4]) and fast (*e.g.,* sub-millisecond for I/O latency), given that the VDs directly interact with the cloud users' operating systems in real-time. As the "compute-storage separation" or "storage disaggregation" architecture of EBS has been widely adopted by mainstream cloud providers, the network that interconnects the compute and storage clusters turns into an essential bottleneck of EBS' overall performance.

Nonetheless, it does not mean that the network solution with the best performance is always suitable to EBS because there are multiple dimensions of requirements on a storage network. For example, storage networks should also support a massive number of connections, long network distances, various types of hardware configurations, be compatible with the computation architecture, and limit the cost for cloud providers. Therefore, designing a storage network is highly challenging.

This paper presents the motivations, challenges, design choices, deployment experiences, and lessons of two significant upgrades on the EBS network of Alibaba Cloud ("AliCloud" for short) in recent five years: LUNA and SOLAR.

LUNA was designed to replace the kernel TCP stack for coordinating the speed shift of the storage medium from HDD (hard disk drive) to SSD (solid-state disk). Although we were deploying RDMA in the storage clusters' backend network, LUNA adopted a user-space TCP software stack instead of hardware for the network between compute and storage clusters due to the concerns on scalability and interoperability. With data from a large-scale deployment in production, we demonstrate that a software stack like LUNA can indeed achieve a significant improvement in end-to-end I/O latency

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SIGCOMM '22, August 22–26, 2022, Amsterdam, Netherlands

© 2022 Association for Computing Machinery.

ACM ISBN 978-1-4503-9420-8/22/08…$15.00

https://doi.org/10.1145/3544216.3544238

and CPU overhead reduction. Meanwhile, Luna also gives us two essential directions to develop the next generation of the compute-to-storage network. First, after the network stack is improved, the storage virtualization of EBS becomes a significant performance bottleneck, as it is on EBS' data path and runs on CPU. Second, the network stack must recover from failures in milliseconds to avoid noticeable user impacts.

Based on the lessons Luna taught us, Solar is a novel storage-oriented and reliable UDP stack to achieve the network-storage joint hardware accelerations and fast failure recovery simultaneously. There are three main challenges to achieve Solar's goals. First, the resource is scarce in hardware and implementing complex and stateful logic in hardware is also arduous; Second, the requirements of fast failure recovery will further damage the scalability of Solar; Third, pragmatically, FPGA hardware is more error-prone (*e.g.,* bit flipping) than software, which could significantly undermine the data integrity after offloading the storage.

The fundamental source of the preceding challenges is the complexity paid for the layered system architecture of the state-of-the-art storage network and the generality that the existing network stacks try to provide. Therefore, the core design of Solar lies in the restructures of the network stack to serve storage dedicatedly. Solar breaks the boundary of the network and storage layers by putting the storage processing into the packet processing pipelines. For realizing this, it makes a packet in the network represent a single data block in storage. With such a network-storage fusion, Solar has the following features:

Offloading the entire EBS data path: The packet processing pipeline can also parse and process the storage semantics embedded in the packet without involving CPU and memory to convert packets into data blocks or process the blocks.

Simplified and lightweight hardware implementation: The "one-block-one-packet" design can also significantly simplify the network and storage pipeline. Because the storage has no requirements on the order of block arrivals, Solar does not need to maintain receiving buffers, connection state machines, or handle packet reordering in the hardware.

High confidence on data integrity: For handling FPGA hardware errors during storage-level CRC computations, Solar performs a lightweight check on an aggregation of multiple blocks' CRC values in software. This design protects the high data integrity and the low CPU overhead simultaneously.

Multi-path transport for fast failure recovery: Solar is friendly to multi-path transport. First, since Solar does not maintain any connection related states in hardware, multi-path does not degrade its scalability; Second, Solar is inherently resilient to the packet reordering caused by multi-path.

We have deployed Solar to approximately 100K servers. For reliability, production data shows that Solar eliminated the I/O hangs caused by network failures. For performance, compared with Luna, Solar reduces the I/O latency by 20%-69% and increases the throughput with a single CPU core by up to 78%. Overall, combining Luna and Solar, we reduced the average I/O latency of AliCloud's EBS service by 72% and enabled the scale up of IOPS (I/O per second) on a single compute server by 220% in the past five years.

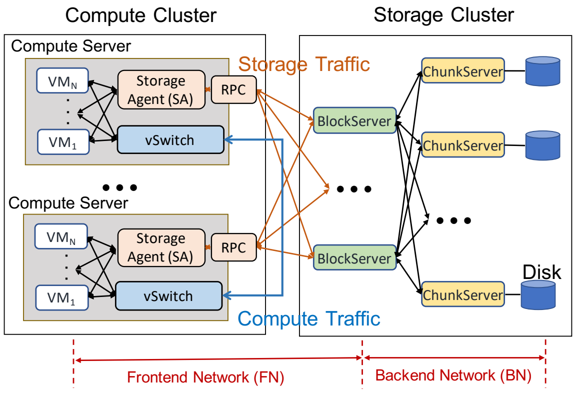


Figure 1: The overall architecture of EBS.

In summary, this paper has four major contributions:

- We introduce the state-of-the-art architecture and working traffic patterns of EBS networks with production experiences and data;
- We present Luna, a user-space TCP stack deployed on a large scale for EBS. We explain the insights on the design choices in practice and also share the performance gains and lessons in production; We point out that the storage virtualization is a new performance bottleneck in EBS's end-to-end performance;
- We present Solar, a novel fusion design of network stack and storage processing that has been deployed in large scales. Solar enables the entire data path of EBS to be offloaded to hardware and the fast failure recovery with a highly efficient multi-path transport;
- We show that Solar provides a general storage offloading solution for hardware hypervisors on the emerging commodity DPUs (data processing units).

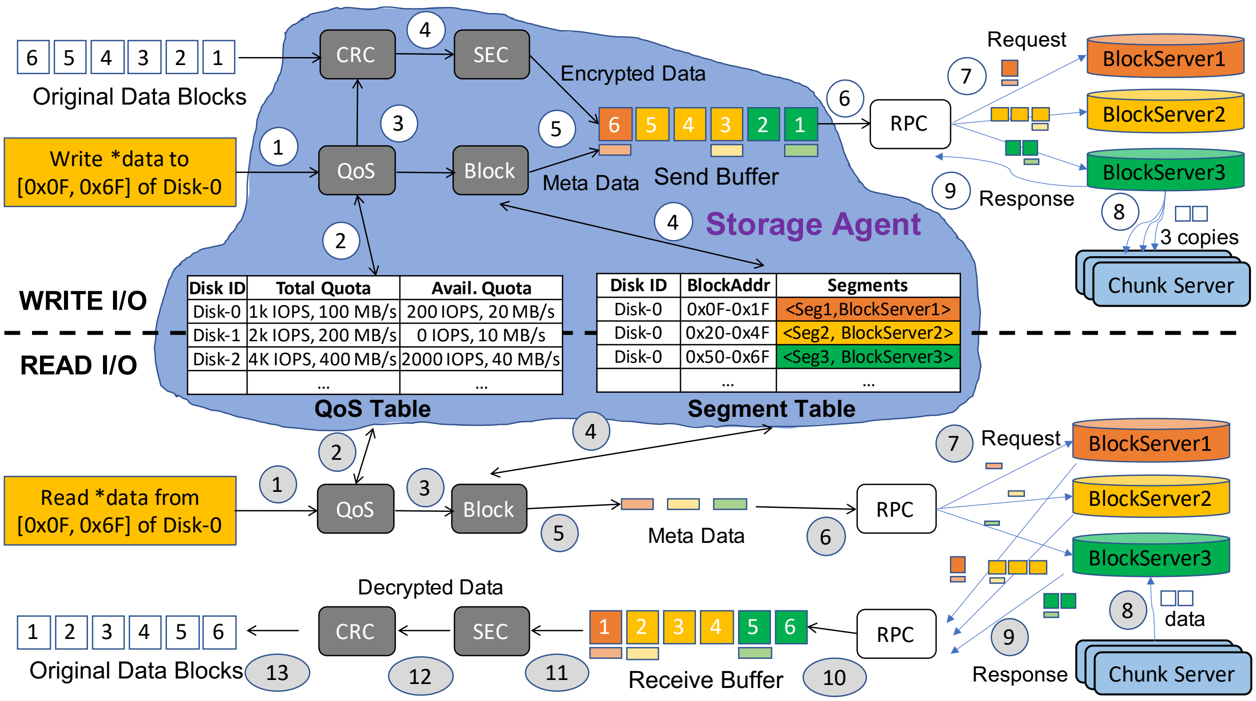
This work does not raise any ethical issues.

2 BACKGROUND

This section provides the background of elastic block storage (EBS) networks. We also show the data from large-scale production networks to offer a deeper understanding of the traffic patterns of EBS networks in the wild.

2.1 Overview of EBS Networks

EBS provides an essential virtual disk (VD) service to cloud guests. Figure 1 illustrates a typical "compute-storage separation" or "storage disaggregation" architecture of EBS. This architecture places compute servers (that host VMs) and storage servers (that host VDs) in separated clusters. There are two main advantages of this design. First, compute and storage servers can be designed independently such that either type of server is optimized for its target workloads, which is more cost-efficient than general-purpose servers; Second, the storage cluster is dedicated to ensuring the safety of data and manages the massive storage media with high utilization to lower



Disk ID	Total Quota	Avail. Quota
Disk-0	1k IOPS, 100 MB/s	200 IOPS, 20 MB/s
Disk-1	2k IOPS, 200 MB/s	0 IOPS, 10 MB/s
Disk-2	4K IOPS, 400 MB/s	2000 IOPS, 40 MB/s
	...	...

Disk ID	BlockAddr	Segments
Disk-0	0x0F-0x1F	<Seg1,BlockServer1>
Disk-0	0x20-0x4F	<Seg2, BlockServer2>
Disk-0	0x50-0x6F	<Seg3, BlockServer3>
	...	...

Figure 2: The internal structure and workflow of storage agent (SA).

the cost. Third, with persistent states stored in storage servers, it is easier and faster to migrate application services across compute servers. It becomes a standalone and common component in the cloud infrastructure that is easy to be integrated by different compute applications.

The compute-storage separation design of EBS augments the importance of the network since all I/O operations on the VDs will go through the network between compute and storage clusters. There are two types of networks in EBS: the frontend network (FN) that connects compute and storage servers; the backend network (BN) provides a fabric among storage servers (*e.g.,* block and chunk servers) in each storage cluster. FN and BN have distinctive properties. For instance, a FN usually expands across multiple clusters and data centers in the same region, while one BN is typically the fabric (*e.g.,* a two-layer Clos topology) of a PoD (point of delivery). In this paper, we mainly focus on FN.

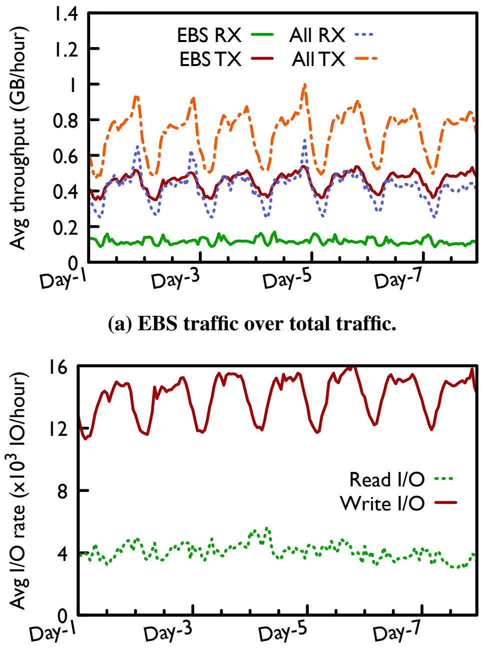
2.2 Storage agent (SA)

As a hypervisor function, SA converts storage operations into network transitions, which defines the traffic inputs of FN. Figure 2 illustrates the workflow of the storage agent (SA) with two examples of WRITE/READ I/O operations to/from a VD. SA maintains two tables: (i) Segment Table, which traces the mapping between the data block address on a VD and the corresponding data segment(s) on the physical disk(s) and the block servers in storage clusters. Segment Table is a core data structure that realizes the virtualization of storage; (ii) QoS (quality of service) Table maintains each virtual disk's service level and current usage, measured by both bandwidth and IOPS. Each I/O request will traverse the QoS and Segment tables on the data plane, whose entries are populated through storage management plane.

After receiving a WRITE request from the guest OS (operating system) (in the upper half of Figure 2), SA checks the Segment Table (Step ④) to find the segments and hosting block servers of the data piece to be written, and then it will call RPC one or multiple times (Step ⑦) to transfer the optionally encrypted data and the meta-data to the block servers in the storage cluster. After the block servers write the data into chunk servers with multiple (*e.g.,* 3) copies (Step ⑧), they will confirm the WRITE success to the SA (Step ⑨).

A READ operation (in the lower half of Figure 2) follows a similar process though it has more steps than WRITE. Note that all data is split into atomic units – data blocks whose size is 4K bytes to be consistent with SSD's sector size – and all operations in SA are in a per-block manner, though RPC may combine multiple blocks in a transition. Thus, the latency is critical in block transition because any straggler would postpond the completion of the entire I/O.

Alternatively, SA could potentially use remote storage protocols (e.g., iSCSI, NVMe-oF) for the network transition and remote disk access [27, 33, 44]. However, this design choice requires the migration of storage operations from the storage server to the compute server, which challenges our environment for three reasons. First, it requires extra processing and states to run storage operations (e.g., LSM-Tree compaction, periodical data scrubbing) and services (e.g., snapshot, fast failover), which incurs significant and often imbalanced overhead to precious CPU in compute servers. Second, to preserve consistency while sharing a VD among multiple VMs, it is more challenging to distribute the shared states across compute



(a) EBS traffic over total traffic.

(b) EBS I/O request rate.

Figure 3: Hourly-averaged throughput and I/O request rate per server over approximately 100K compute servers in a week.

servers with a strong consistency than today's approach to aggregate and sequentialize operations in a block server. Third, it hinders our flexibility and granularity for allocating elastic disk resources on demand.

2.3 Traffic patterns of FN

Traffic volume: Generally, there are two types of traffic in data centers: VPC (virtual private cloud) traffic and EBS traffic. Figure 3a shows that EBS traffic has become the majority traffic volume. EBS traffic accounts for 63% of the server's TX traffic or 51% of overall traffic. Figure 3a and 3b show that WRITE I/O requests are 3-4× as many as READ in both volume and I/O rate. Also, as shown in Figure 4, on average, a single compute server can have up to 200K IOPS (or the number of network flows per second) in production.

I/O and flow sizes: Figure 5 shows the distributions of the sizes of I/O operations and RPC data transfers. We can see that in FN, the RPC size (or flow size) is under 128K bytes, and about 40% RPCs are up to 4K bytes. Figure 5 also illustrates the reasons for the small flow size: the I/O size follows almost the same distribution because the guest applications (*e.g.,* databases) intentionally invoke small I/O requests to ensure their data integrity. Therefore, most I/O operations can finish with a single RPC in practice. Since small I/O operations are typically sensitive to latency, the flows in EBS are mostly latency-sensitive.

I/O latency: Figure 1 shows the four portions in the end-to-end I/O latency in EBS: SA, RPC over FN, RPC over BN, and chunk server with SSD. Each of the portions can be a bottleneck depending on the specific environments. For instance, as shown in Figure 6, kernel TCP has a long latency, and it becomes the performance bottleneck when the storage medium starts to shift from HDD to SSD. Especially, Figure 6 shows that write I/O has only tens of μs latency (one to two orders of magnitude faster than kernel TCP) in chunk server due to the use of SSD's write cache without touching its NAND medium[1]. With LUNA and RDMA deployed in FN and BN, respectively, the end-to-end I/O latency is primarily reduced to match the speed of the storage medium. As a result, SA becomes the bottleneck at the tail.

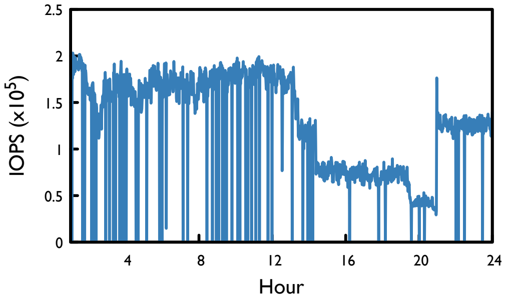


Figure 4: Average IOPS monitored in every minute over a day for a highly-loaded compute server in production.

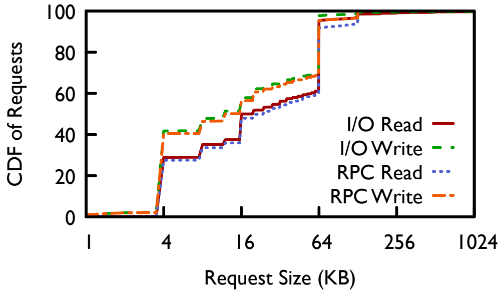
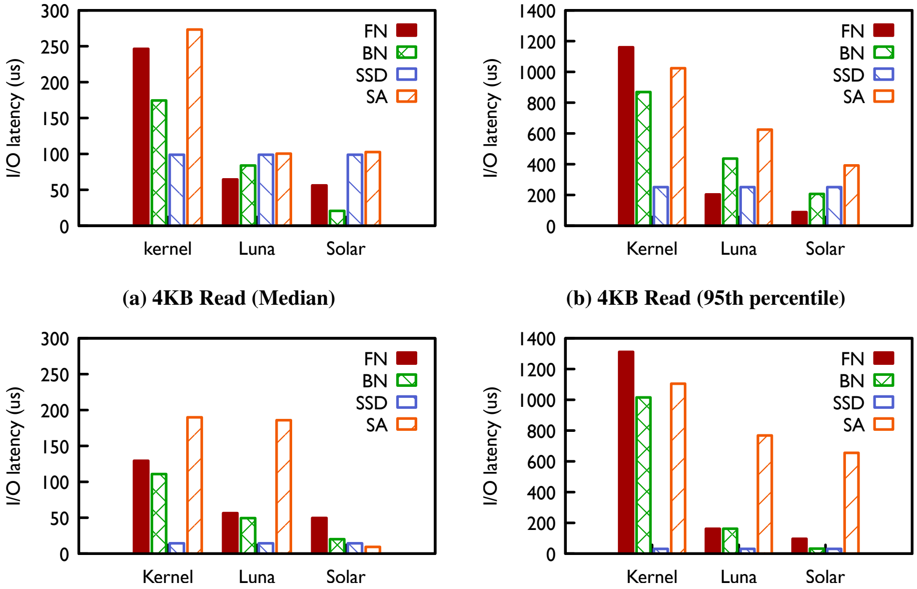


Figure 5: The distribution of I/O and FN RPC sizes. Typical sizes are 4K, 16K, and 64K bytes.

3 LUNA: MATCH SSD'S HIGH-SPEED

The emergence of SSD has profoundly impacted the expectations of EBS. Since SSD significantly reduces the I/O latency from HDD's tens of milliseconds to tens of microseconds, applications are designed with fewer concerns on the slowness of I/O operations. Moreover, applications tend to write data to SSD instead of memory for better data durability. Taking the typical database application as an example, it uses LRU (least-recently-used) queues to replace the cached data in memory. Each replacement leads to storage access at the granularity of pages (*e.g.,* 8 KB and 16 KB in size with Oracle SQL and MySQL, respectively). Therefore, it causes more strict SLAs (service level agreements) for EBS backed on SSD. For instance, AliCloud EBS released ESSD (Enhanced SSD) in 2018,

[1] The fragmentation is carefully trimmed by turning random writes into sequential writes with log-structured merged-tree (LSM tree) and commit aggregation [56].



(a) 4KB Read (Median)

(b) 4KB Read (95th percentile)

(c) 4KB Write (Median)

(d) 4KB Write (95th percentile)

Figure 6: I/O latency breakdown of 4KB size monitored in production over approximately 100K compute servers in a week. Latency in each component is monitored using distributed trace, where the policy-based queuing delays like QoS are excluded from capturing the actual system performance. "Kernel" denotes that both FN and BN are kernel TCP. The BN of LUNA and SOLAR is RDMA. "SSD" includes the processing time in chunk servers and I/O in physical SSDs.

providing 1,000,000 IOPS and 100μs average latency of a single request [4, 56].

3.1 The choice of software

The traditional kernel TCP cannot satisfy ESSD's requirements on the end-to-end I/O latency because the kernel stack latency can be hundreds of microseconds or even milliseconds. For low latency, there were two mainstream solutions from which to choose: software solutions with kernel bypassing technologies (*e.g.,* Intel's DPDK) or hardware-based solutions (*e.g.,* RDMA). While we had already been deploying RDMA in our BN, our choice for FN is software.

Compared with BN, FN puts scalability and interoperability in a much higher priority:

Scalability: Due to the massive number of compute nodes, a storage node often has tens of thousands of concurrent connections with the compute. Therefore the network stack of FN should keep high performance and high reliability even facing numerous connections; Moreover, FN should support inter-data center traffic to allow extensive access to storage data from compute clusters in different data centers. Conversely, BN has much less requirement on scalability because storage clusters are typically small (tens of servers). It is because (i) the disk density of a single chunk server can be high, so there is no need to scale out, and (ii) the scale of the storage clusters is controlled to limit the impact domain of failures.

Interoperability: Since different compute and storage clusters are built at different times, they usually have different hardware configurations (*e.g.,* the brands and models of a network interface card) in their nodes. Therefore, FN must provide strong compatibility to connect servers with different configurations. This property also ensures that compute and storage nodes can exchange data universally. However, BN's interoperability is almost always guaranteed because the hardware configuration in an individual storage cluster is often consistent.

In 2017, RoCEv2 (RDMA over Converted Ethernet Version 2) is still in an early stage. Due to the limited resources in hardware, the overall throughput of the RNIC (RDMA capable network interface card) we use went down quickly after the number of connections was beyond 5,000, which is too low for our scale (similar observation from [31]); Also, RoCE had severe compatibility issues among different vendors and even different models from the same vendors because different types RNICs speaks slightly different protocol semantics, which is opaque to cloud providers; For instance, Mellanox keeps adding new features to its newly released RNICs, which are either not backward compatible (*e.g.,* Selective Repeat v.s. Go-Back-N) or require completely different network setups (*e.g.,* lossless v.s. lossy and Zero-touch v.s. ECN/PFC) [10]. Thus, different models of Mellanox RNICs cannot easily communicate directly, not to mention talking to other vendors. Finally, RoCE uses PFC for flow control, which is a reachability risk because PFC storms or deadlocks in large-scale networks like FN can be disasters [26]. Even though other cloud providers like Microsoft have made region-level RDMA deployment [41], they typically use deep buffer switches in both DC core and regional switches, with constant buffer tuning in each tier of network switches and close collaboration with vendors to enable PFC in FN. Instead, our design choice is that FN should be

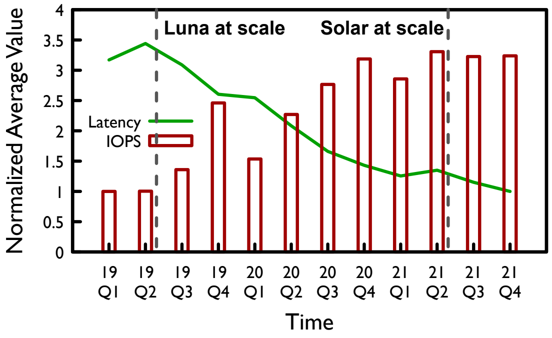


Figure 7: The evolution of average IOPS (normalized with Q4 2021) and latency (normalized with Q1 2019) per server across all compute servers using ESSD service. The network stack reduced the average I/O latency by 72% and tripled the IOPS in the last five years.

	Avg. RPC Latency (μs)		Consumed cores	
	Kernel	Luna	Kernel	Luna
Single 4KB RPC	70.1	13.1*	1	1
50 Gbps stress test	1782	900	4	1

(a) Tested using 2×25GE.

	Avg. RPC Latency (μs)		Consumed cores	
	Kernel	Luna	Kernel	Luna
Single 4KB RPC	43.4	12.4*	1	1
200 Gbps stress test	2923	465	12	4

(b) Tested using 2×100GE.

Table 1: FN RPC latency and CPU used under different load. Stress test uses concurrent RPCs. (*including the base RTT of 8.3 μs)

loss-tolerant, and shallow buffer switches are used within the region to save cost.

On the contrary, software-based solutions do not have the preceding issues in the context of FN, and its sub-optimal performance and relatively high CPU overhead is the price we have to pay at the time.

3.2 The deployment of Luna

Luna has a similar architecture as mTCP [28] which provides the Run-to-Complete (RTC) with zero-copy for network transport. Luna extends the RTC model into storage processing with two significant improvements besides many small ones. The first is that Luna achieves a zero-copy design across SA and RPC for saving CPU, by recycling and sharing buffers across layers. The second is that Luna achieves "lock-free and share-nothing" thread arrangements across network and storage processing, which removes locks and buffer sharing across CPU cores for high parallelism. In addition, Luna also uses features provided by NICs (*e.g.,* TCP/generic segmentation offload, or TSO/GSO) for partial hardware offloading to improve performance and reduce CPU overhead.

Luna has been deployed to almost all EBS clusters of AliCloud since released in 2019. Figure 7 shows that storage network stacks reduced the average I/O latency of AliCloud's EBS service by 72% in the last five years. We can see that the average latency and IOPS were continuously improved as LUNA was gradually deployed. By the time it was fully deployed (2021 Q1), Luna had cut the average I/O latency by more than 64% and helped to increase the average IOPS by 180%. Figure 6 shows that Luna reduces FN latency of kernel TCP by about 80% overall in production to match the speed of SSDs.

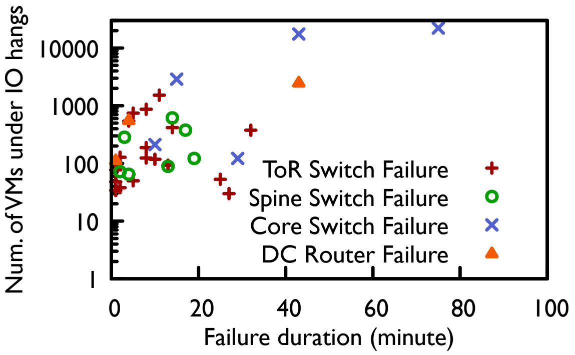


Figure 8: The impact of I/O hangs caused by around 100 network failures with locations over a two-year period.

Luna must sature the line-rate to fulfill the service-level agreement (SLA) for the peak capacity the user has purchased. Table 1a and 1b present the performance and CPU overhead comparisons between Luna and kernel TCP stack for running RPCs. For 2 × 25GE, Luna only needs one CPU core while the kernel TCP needs four; Luna cuts the latency by more than 80% without background traffic and by about a half when the throughput is approaching the capacity. The gaps of CPU overhead and latency under full throughput are even larger for 2 × 100GE.

3.3 The lessons from Luna

Luna is our first move towards high-speed network stacks for supporting storage. As well as the performance gains, we also learn critical lessons that can guide the next generation of storage network stack design.

SA is becoming the bottleneck. With Luna, the bottleneck of the end-to-end I/O latency has become SA as shown in Figure 6. It is because SA is on the data path of the EBS; it has to perform heavy computations (*e.g.,* CRC, Crypto) and per-I/O table lookups in CPU. Therefore, in the next step, the key to improving EBS' performance is mainly on SA and thus, offloading SA to hardware for saving CPU overhead is an option.

Software is unsustainable with the increasing speed. Even if Luna is not a latency bottleneck now, that does not mean it can sustain production for a long time. The main reason is its high CPU overhead. In Table 1a and 1b we see that Luna needs 4 CPU cores for 2×100Gbps network speed running RPC transfers (the I/O throughput is even lower). As the network bandwidth quickly ramps up to 2×200Gbps and 2×400Gbps, offloading the network stack into the hardware is indispensable.

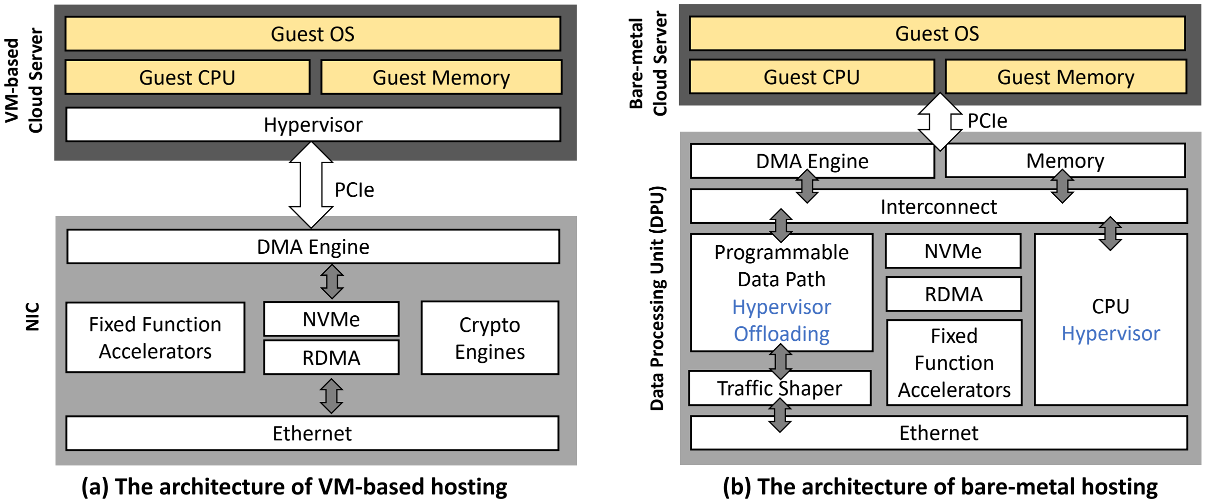


Figure 9: The hardware architecture of bare-metal cloud.

Common network failures can be disasters to storage. In production, network failures easily create traffic blackholes that impact the storage's performance [16, 58]. Figure 8 shows a large number of VMs experienced I/O hangs (defined as I/O gets no response for one minute or longer) due to failure incidents. Unexpected long I/O wait time during I/O hang may lead to VM crashes [58] since VM views EBS as a PCIe device. The key lesson is that even though storage offers high availability and durability using replication or erasure coding (EC), performing data recovery in the storage incurs a relateively high overhead, such as backup read in EC, fencing for data consistency, etc. Those reactions typically require multiple seconds to minutes to converge. In addition, despite the network fabric having enough redundancy (even with the ToR switch, we connect each server to a pair of it), LUNA has no option but to wait for the long recovery time (*e.g.,* minutes) via network operations.

Take an actual incident as an example. In one storage of LUNA's deployment, one line card of a Core switch failed, leading to 4% packet loss across the cluster, and more than one thousand VMs were affected. It takes 12 minutes for the network operation to locate and isolate the failing card, and the storage recovered after additional 30 minutes. Specifically, such a long connectivity loss can easily trigger failure reactions in the storage, and the whole system will experience an even longer convergence time. In our later study, which replayed the incident, we found the storage would have no visibility to the failure if LUNA could have found a good network path and avoided the failing core switch within one second.

4 SOLAR: OFFLOAD EBS' DATA PATH

Starting from 2018, we develop our new generation of EBS network "SOLAR" after LUNA. Compared with LUNA, SOLAR is a fundamental redesign of the FN stack not only to solve the issues left in LUNA but also to embrace a profound transform in the compute architecture: virtual machine (VM) based hosting to bare-metal hosting.

4.1 Background: the rise of bare-metal cloud

Offloading cloud infrastructure to hardware is a long-term endeavour led by top cloud providers [2, 3] and hardware vendors [6, 9, 11, 12]. Generally, the cloud infrastructure includes virtualizations of compute (*e.g.,* VM), network (*e.g.,* virtual private cloud (VPC)), and storage (*e.g.,* EBS). The first component that was offloaded to hardware was VPC, which promoted the development of SmartNICs [22]. The success of network offloading stimulated the idea of offloading the compute and the storage, which finally forms the concept of bare-metal clouds.

The core idea of bare-metal cloud is putting the cloud infrastructure into dedicated hardware (as shown in Figure 9)(a), such that the cloud user can occupy the entire physical machine without sharing resources (*e.g.,* CPU and memory) as the cloud provider or other guests. Bare-metal cloud has remarkable advantages:

Flexibility: Cloud tenants require the option of bare-metal servers to run either their virtualization (*e.g.,* VMware Cloud [14]) directly on the hardware without the performance hit of nested virtualization or applications intended to access the low-level hardware features (*e.g.,* Intel RDT [13]) or require licensed supports in non-virtualized environments. Meanwhile, they still want to access the elastic, reliable, and highly efficient infrastructure services (*e.g.,* EBS). On the other hand, the type of resources (*e.g.,* CPU) a guest can get is defined by the cloud providers because the guest's programs have to run on the same hardware platform with the cloud infrastructure. Though such bounding seems to be working relatively well in the traditional public cloud, it is more troublesome to have such a constraint in private/hybrid clouds where customers desire to move their off-cloud environments to clouds without significant changes.

Agility: Bare-metal cloud is favorable in the edge scenarios. This is because the infrastructure overhead becomes significant when the cloud size is small [54], and the overhead will be well limited if the entire cloud infrastructure is in dedicated hardware. Moreover,

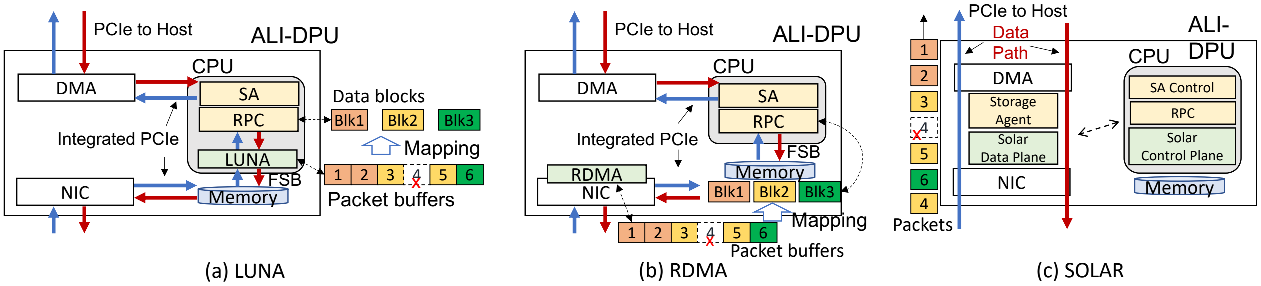


Figure 10: High CPU overhead and PCIe bottleneck for either LUNA or RDMA; SOLAR achieves data path offloading.

the deployment of bare-metal clouds becomes lightweight and fast (ideally insert a PCIe card), which is critical for edge clouds.

Isolation and security: Since the guests' hardware is isolated from the cloud infrastructure, it is easier to guarantee performance isolation. Also, attacks based on resource sharing (*e.g.,* side-channel attacks) will be challenging to realize.

4.2 Data processing unit (DPU)

Traditional cloud servers offer guest VM hosting and run cloud hypervisor in host CPUs (Figure 9(a)). DPU is the foundation of bare-metal clouds. As shown in Figure 9(b), a DPU provides CPU, memory, programmable hardware (*e.g.,* FPGA or ASIC), traffic shaper, and fixed-function accelerators for offloading the hypervisor. It also provides a DMA engine that can read/write data directly from/to the guest memory via PCIe. There is also an internal interconnect channel among the CPU, memory, and accelerators.

Alibaba is one of the pioneers of bare-metal clouds. As our early version of DPU, ALI-DPU, which integrates FPGA for the progammable data plane with interconnection options, has some critical resource limitations by that time. First, the FPGA's resource is minimal due to the hardware cost and power consumption concerns (e.g., DPU is typically limited to 100~300 Watt), and the infrastructure CPU on ALI-DPU only has six cores. In addition, due to the delay of PCIe 4.0, the internal interconnect of ALI-DPU has not enough bandwdith capacity. While the Ethernet can be 2 × 25Gbps, the internal PCIe channel is far less than 100Gbps. Hence, the internal PCIe channel could be a throughput and latency bottleneck if traffic passes through it back and forth.

As the service model extends from a public cloud host to private and edge clouds, the evolution of the bare-metal cloud is accelerating, backed up by the whole cloud and hardware industry. Nowadays, multiple major hardware vendors have released the designs and road maps of their commodity DPU chips (*e.g.,* Intel's Mount Evans [9], Nvidia's BlueField [11], Fungible's F1 [6], Pensando's DSC [12], *etc.*). In these chips, the internal interconnect channel may not the bottleneck anymore; the programmable data path is ASIC (instead of FPGA) that supports P4 languages, but the CPU resource is still limited to control the cost and power consumption.

In this section, we focus on the design of SOLAR on top of ALI-DPU and we will discuss how to run SOLAR in commodity DPUs after they are finally released shortly.

4.3 EBS networks in bare-metal clouds

As shown in Figure 2, SA realizes the storage virtualization in the host CPU. Hence, in the bare-metal cloud, SA has to run inside the infrastructure CPU of ALI-DPU. This structure results in two significant challenges to EBS in the bare-metal clouds.

CPU overhead: As shown in Figure 10(a), EBS' entire data path, including both LUNA's network stack and the SA, have to go through ALI-DPU's CPU, resulting in a heavy CPU load that increases with the network throughput. Furthermore, unfortunately, RDMA's improvement is limited (Figure 10(b)) because it merely offloads the network stack, but not SA. Thus, the data path still goes through ALI-DPU's CPU.

Limited throughput: Figure 10 also illustrates that either LUNA or RDMA will suffer from the bottleneck in ALI-DPU's integrated PCIe because they require the traffic to transverse PCIe twice.

Therefore, limited by the new compute architecture and the hardware capabilities by the time, we have to design a new FN stack for EBS to embrace the era of bare-metal cloud.

4.4 The system design of SOLAR

The design goal of SOLAR has two folds. On the one hand, for adapting to the new compute architecture, it should remarkably reduce the CPU overhead on ALI-DPU and avoid the internal PCIe channel by allowing both the network stack and the SA offloaded to hardware (as shown in Figure 10(c)); On the other hand, for solving the reliability problems we observed in LUNA, it should be able to detect and avoid in-network failures with path changing actively. Our design should accommodate the two independent requirements harmoniously.

Challenges: There are three main challenges to achieving the goals of SOLAR. First of all, both the network stack and the SA are complex and stateful software systems, such that it costs tremendous development efforts and hardware resources to implement the current system architectures of them into the FPGA. Especially because the FPGA on ALI-DPU is also used for other hypervisor functions (*e.g.,* virtual switch), the resources left for EBS are very limited.

Next, even if we can re-implement the current LUNA and SA with FPGA, its scalability will become a new concern. Furthermore, our design choice is a simple and highly efficient multi-path transport [48] for fast failure recovery. However, it requires states on each traversing path, which will make the scalability issue even worse.

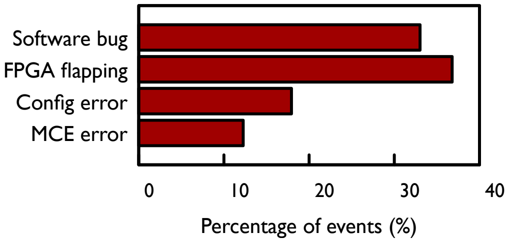


Figure 11: The root causes of about 100 data corruption events mitigated by software CRC in past two years. MCE (Machine Check Exception) error detected by the processor is the hardware error in the processor, cache, memory, and system bus.

Finally, according to our practical experiences, FPGA is error-prone due to random hardware failures (*e.g.,* bit flipping). Among the data corruption incidents detected by CRC mismatches in production, FPGA error is the major contributor by 37% as shown in Figure 11. Our experience reveals that bit flipping in FPGA can corrupt data and table entries in memory and distort the execution logic towards an unexpected outcome. Therefore, a concern to offload SA's CRC into the FPGA is that it could potentially undermine the data integrity because of errors in the CRC computation on the FPGA itself. Furthermore, since EBS has an extremely high SLA for data integrity (*e.g.,* 99.9999999% [4]), SOLAR must provide at least the same confidence as software.

Observations: SOLAR has to eliminate unnecessary complexity in both network stack and SA to fit the ALI-DPU hardware accelerator. We observe that most of the complexity comes from the overgeneralization and layered system structure. Specifically, for instance, according to the byte-stream abstraction, either TCP or RDMA in RC (reliable connection) maintains complex state machines for connections and packet buffers for handling packet loss or reordering, but it is unnecessary to EBS. In an I/O READ as shown in Figure 10, EBS accepts any arriving orders of the blocks if only all of them arrive on time while in-order buffering creates unnecessary head-of-line blocking; For another example, to map packets on the network layer to the data blocks on the storage layer, SA has to maintain data structures to remember and update the mapping records in real-time. As the system scale and the exceptions go up, maintaining the preceding states is a huge burden that prevents hardware offloading.

Idea – the fusion of storage and network: The key to reducing the complexity and the number of states maintained in FN is eliminating the packet buffering and packet-to-block converting. Therefore, SOLAR makes a one-to-one mapping between a packet and a data block, which essentially breaks the boundary between network and storage. Specifically, while the network is a packet processing pipeline, SA is essentially a pipeline of data chunk processing (Figure 2). Hence, we can merge the pipelines of the network and the storage if we make a single packet represent a single data block.

In summary, the "one-block-one-packet" strategy also has the following excellent properties:

Low packet buffer . Since the storage pipeline can directly process a packet in the hardware, packet buffering for assembling packets into data blocks is not needed anymore.

Zero-touch to CPU or memory . The handover between the network and storage pipelines can be done within the hardware accelerator without using the memory as a hub.

Few maintained states . There is no need to maintain mappings between packets and data blocks anymore.

Compatibility with multi-path . As all packets become independent, the whole transport is insensitive to packet reordering, which significantly simplifies the multi-path transport.

Easy to realize in practice . Fortunately, the "one-block-one-packet" also does not require significant storage or network changes. It is because currently, a data block is typically 4K bytes in size, and a packet can be up to 9K bytes in a jumbo frame. Therefore, the design is pragmatic if only the network supports the jumbo frame, which is prevalent in state-of-the-art data centers.

4.5 The workflow of SOLAR

SOLAR achieves I/O operations using "one-block-one-packet" network transitions. First, The storage I/O is split into one or multiple outgoing RPCs towards different storage servers. Then, each RPC packet delivers and retrieve one data block for WRITE I/O and READ I/O, respectively. Figure 12 and 13 demonstrate the system architecture of SOLAR and the workflow of an I/O WRITE and READ operation, respectively. Compared with Figure 2, we can see that SOLAR put the data path of SA into the FPGA.

WRITE: When the guest issues a WRITE to a VD, the I/O operation is directly forwarded to the FPGA on ALI-DPU via an NVMe command. QoS and Block are two typical "match-action" table checking steps: QoS performs admission control of I/O operations to enforce bandwidth constraints for maintaining the service-level agreement (SLA) of each VD; Block translates the VD's block address (i.e., Logical Block Addressing or LBA) to its corresponding segment address of the physical disk in the remote block server. A large I/O (e.g., a 256KB write I/O) may consist of LBA addresses located in multiple block servers, and therefore Block splits the I/O into smaller I/Os, one for each block server, by adjusting the LBA address. Note that the chance of I/O splitting is typically low since the I/O size is generally small (Figure 5), and each segment hosted in a block server contains relatively large (e.g., 2MB) and continuous LBA addresses to avoid such fragmentation.

After the block table lookup, ALI-DPU's CPU polls the I/O operation to issue an RPC for data delivery of each (possibly split) I/O. The final metadata obtained from the block table will be inserted into the packet generator (PktGen) as the EBS header in the packet. In addition, the RPC and Path&CC modules in the CPU will insert the RPC header and the UDP header (using the source UDP port as the path identifier) into the packet, respectively. Meanwhile, FPGA fetches the block data via DMA, computes its CRC value in the CRC module, and optionally encrypts the data in the SEC module. Finally, the PktGen module will send out the generated packet with the sending rate from the Path&CC module in the CPU of ALI-DPU. After the RPC generated, the CPU will get the acknowledgments with the path condition(i.e., timeout, RTT) and congestion feedbacks (i.e., INT (in-band network telemetry)) for path selection and congestion control for each RPC independently.

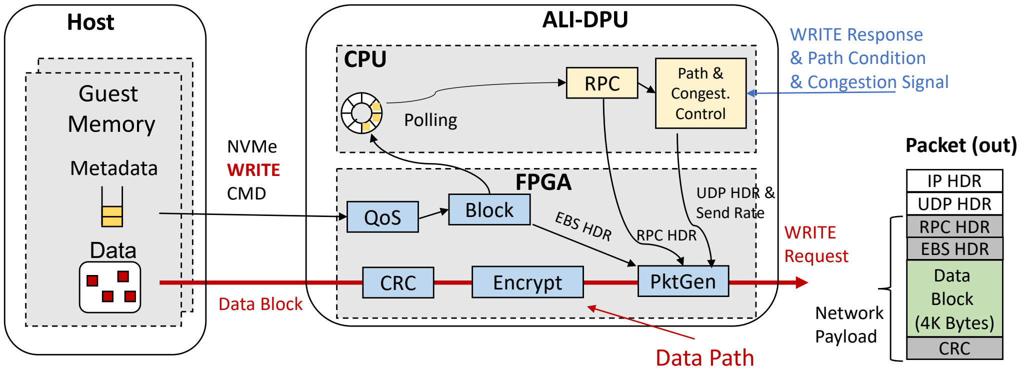
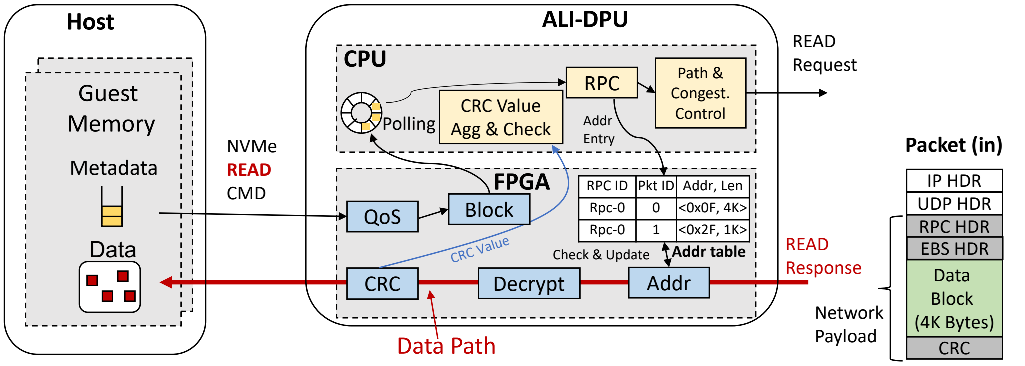


Figure 12: The workflow of a WRITE request in SOLAR.



RPC ID	Pkt ID	Addr, Len
Rpc-0	0	<0x0F, 4K>
Rpc-0	1	<0x2F, 1K>

Figure 13: The workflow of a READ request in SOLAR.

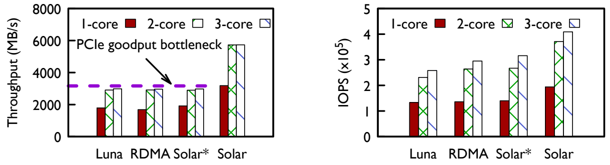
READ: To accommodate incoming block data, SOLAR maintains an Addr table that records RPC and packet IDs and the corresponding guest memory addresses. Like the WRITE I/O, the READ I/O will traverse QoS and Block tables to perform request admission, segment lookup, and potential I/O splitting. When sending an RPC request for each (possibly split) I/O, the RPC populates an entry in the Addr table for each incoming packet and removes it after the reply arrives. Since each packet in the READ response is an independent data block and the IO size is generally small (Figure 5), SOLAR performs packet processing at line-rate without buffering or extra state maintained in the Addr table.

The READ request (a UDP packet) is sent from the CPU of ALI-DPU, and the SOLAR control plane decides the sending path and sending rate. When the response comes in, it will directly go to FPGA, given that it has a data block payload. Since the RPC has already recorded the actions in the Addr table, the response packet can be directly processed, and its entry is cleaned afterward without interrupting the CPU.

After the last step in the hardware pipeline (CRC check), the hardware will put the data into the guest memory via DMA. Meanwhile, the hardware sends the headers and metadata of the packet to the CPU for the final data integrity check and congestion window updates. Then, the SA control plane in the CPU will ring the doorbell to the guest OS after the data integrity check passes.

Hardware errors v.s. data integrity: EBS heavily relies on CRC to check the errors in the hardware. However, as we mentioned early, the CRC check performed on FPGA cannot guarantee a high standard of data integrity. Alternatively, moving the CRC computation back to the CPU will increase the CPU overhead. SOLAR introduces the CRC aggregation and checking in the CPU for both data path offloading and data integrity. In this design, CRC32 is deployed in FPGA, and the CPU merely verifies segment level CRC with the CRC values for each data block in the segment. It essentially takes advantage of CRC32's divide-and-conquer property – $CRC(A \otimes B) = CRC(A) \otimes CRC(B)$ [1].

Multi-path transport: SOLAR's control plane maintains multiple (*e.g.,* 4) persistent paths towards each block server. We use different UDP ports as path IDs and leverage the randomness of ECMP's consistent hashing to select paths. For each path, SOLAR maintains path conditions (*e.g.,* congestion window, sending rate, RTT, *etc.*), which will be updated by congestion signals and acknowledgment. SOLAR disseminates packets over multiple paths in favor of the path with low average RTT. Packet loss is detected via out-of-order



(a) Throughput of 64KB I/O.

(b) IOPS of 4KB I/O

Figure 14: Fio Read test with 32 I/O depth under different number of 2.1 GHz cores.

arrivals or timeout happened in the same path for selective retransmission. Solar uses consecutive timeouts to infer a path failure and shifts traffic to other paths accordingly. This design is simple but effective in practice. There are indeed some cases in which the recovery is slow because multiple paths go through the same failure points(Table 2), and we plan to make the path selection more explicit with INT probing.

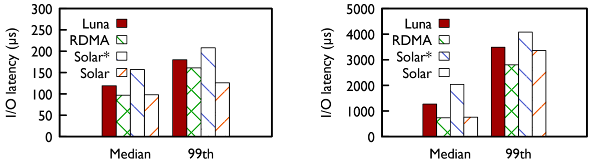
4.6 Solar in commodity DPUs

Despite that the first target hardware of Solar is Ali-DPU, we keep its design for future commodity DPU chips. When designing Solar, it was clear that ASIC will finally replace FPGA after the solution of hypervisor offloading converges because DPU needs to optimize the cost and the power consumption; it was also predictable that DPU would have a programmable packet processing pipeline (and use languages like P4) for offloading networking functions (*e.g.,* virtual switch, firewall, load balancer, *etc.*) Therefore, Solar has the advantage of leveraging the programmable packet processing pipeline to offload storage virtualization. Since in Solar, each data block is a single packet, and the functions in SA are essentially block reading, data computation, block writing, and table checking/maintaining, the data path of SA can be expressed with the P4 language and executed on the P4-compatible pipeline.

4.7 The deployment of Solar

Solar has been deployed in our production by over about tens of compute clusters and approximately 100K servers since 2020. We use monitoring data from production clusters and testbed results to evaluate the performance and resource consumption of Solar. We use Solar * to denote the Solar scheme with data plane offloading disabled.

Solar improves the I/O performance. Figure 7 shows that the overall latency in EBS production is reduced by 25% since Solar's large-scale deployment. The measurement from production clusters shows that Solar reduces latency for small I/O compared with Luna. For instance, as shown in Figure 6, Solar reduces the median latency of SA by 95% and end-to-end by 69% for 4 KB Write. Solar also decreases the FN latency, especially by more than a half in the 95th percentile of 4 KB Read. The latency reduction comes from hardware offloading and state reduction from its "one-block-one-packet" strategy. Note that Figure 6(d) also shows that Solar still has a long tail latency in SA, though its gain on average latency is remarkable. This is because Solar still relies on the CPU to perform path selection and congestion control (especially for WRITE), which can create high CPU loads under intensive I/Os.



(a) Light load.

(b) Heavy load.

Figure 15: I/O latency test of a single 4 KB Write under different background load.

Failure scenarios	Luna	Solar
ToR switch port failure	0	0
ToR switch failure	216	0
Spine switch failure	0	0
Packet drop rate=75%	10 per Second	0
ToR switch reboot/isolation	123	0
Blackhole in a ToR switch	611	0
Blackhole in a Spine switch	1043	0

Table 2: Number of I/Os with no response in one second or longer under failure scenarios in our testbed. The experiment is performed by 90 compute servers and 82 storage servers with generated traffic from block sizes of 4-32 KB, I/O depth of 4, and read/write ratio of 1:4.

Testbed results in Figure14 show that Solar achieves better resource efficiency than Luna and RDMA, as the throughput and IOPS of a single core increased by 78% and 46%, respectively. Solar also circumvents Ali-DPU's onboard PCIe bottleneck through completely offloading data plane to hardware at line-rate. Figure15 shows that Solar achieves a low I/O latency close to RDMA. We also test Luna with jumbo frame and the result is the same due to the inevitable CPU handover and states.

Solar avoids IO hangs from network failures. Guest VMs used to suffer from tens of IO hang incidents per year due to network failures (Figure 8). We reduced this number to zero over its two-year deployment when Solar deployed. Table 2 shows more cases in testbed results to reduce slow IO responses time during failures. The critical insight is that even though the "fail-stop" failures on a device or port can be quickly converged via ECMP routing, the traffic blackhole on a subset of traffic is hard to detect and mitigate via network operations. Furthermore, EBS is increasingly sensitive to latency hit and reachability loss because guest applications view EBS as a PCIe device. Hence, Solar fulfills the need to quickly route around the failure point to achieve high availability.

Hardware consumption Table 3 shows Solar's hardware resource consumption. Solar achieves its performance and reliability goals under Ali-DPU's stringent resource constraints.

4.8 The lessons from Solar and discussions

The speeds of network and PCIe: One of Solar's advantages is bypassing PCIe, which is the throughput bottleneck of Ali-DPU.

Modules	LUT(%)	BRAM (%)
Addr	5.1	8.1
Block	0.2	8.6
QoS	0.1	0.4
SEC	2.8	0.9
CRC	0.3	0.0
Total	8.5	18.2

Table 3: SOLAR's hardware resource consumptions.

Thus, SOLAR manages to handle about 150K IOPS per CPU core without incuring queueing delay. While it might seem very specific to our hardware limitation, it essentially reflects that the network's speed has become comparable to PCIe. It spent close to ten years from PCIe 3.0 to PCIe 4.0, but at the same time, the network interface was upgraded from 10Gbps to 100Gbps (or more). Therefore, we believe bypassing PCIe will be a long-term requirement for all high-performance network protocols. Here, the key techniques require to offload both network transition and application's post-processing before DMA.

Pros and cons of jumbo frame: Large packet sizes can increase the chance of congestion in the switch that uses store-and-forward pipelines, especially running with multi-path exaggerates the incast scenario. We take two steps to reduce the risk of high latency brought by the jumbo frame in SOLAR. First, we use a per-packet ACK to perform a fine-grained congestion control algorithm (*e.g.,* HPCC [38]) with a dedicated queue in the switch for SOLAR; Second, we use 4K bytes instead of 8K bytes for the jumbo frame to balance the congestion risk and the benefit from the end-to-end perspective.

Integrated EBS with DPU: The compute-storage separation architecture fits the large-scale storage requirements well but introduces a high network communication overhead. In edge or private clouds where the network scale is limited but bare-metal hosting and high-performance are still needed, we can consider merging the SA and the block server into DPU and implement them in the hardware P4-capable pipeline.

5 RELATED WORKS

High-performance network stacks: The challenges to the network brought by the advances of the storage medium is a common problem in the cloud, while different cloud providers have their strategies and solutions. As this paper focuses on FN, AliCloud also reported its RDMA deployment experiences in BN [23]. Microsoft is using RDMA for the latency-sensitive services (*e.g.,* search, storage) [26, 59], and Huawei also adopts RDMA for its storage [57]. Different from the preceding hardware-based solutions, Google chooses user-space software solutions [34, 42] and demonstrates their performance is close to the hardware with a high velocity of update and customization. Furthermore, Google improves RDMA's scalability and predictability using a software-aided architecture [53]. Compared to the preceding reports from the industry, we provide more insights on the differences of FN and BN, the considerations and tradeoffs for FN, the lessons learned from a massive scale deployment of user-space TCP, and the changes of performance bottlenecks from the end-to-end perspective of I/O latency.

Recently, cloud providers started to build their high-speed network stacks according to their experiences of operating existing solutions. For instance, Amazon designs EFA (Elastic Fabric Adapter) [51] specifically for the HPC (High-Performance Computing) scenarios on its self-designed NIC "Nitro". EFA has some similar properties with SOLAR, such as a reliable UDP protocol and a multi-path transport, but it does not target on solving FN's scalability and interoperability concerns or achieving hardware offloading for the entire storage data path.

In-network accelerations: With the emergence of the programmable network data plane, people found it might be beneficial to the applications if they could offload (partial) loads into the switches or SmartNICs. For instance, Microsoft Azure [22] and AliCloud [46] offload various network functions into FPGA-based SmartNIC and P4 switches, respectively; Mellanox [25] builds an in-network aggregation protocol using InfiniBand switches. SOLAR's focus is EBS, which is complementary to these works. There are also many works from academia in this direction. For example, NetCache [30] and NetChain [29] accelerate key-value stores and distributed locks with programmable switches respectively; Gimbal [44] designs a multi-tenancy framework for offloading NVMe storage to SmartNICs. Many other works use programmable switches to accelerate caching [39], machine learning [21, 24, 35, 50], consensus protocols [18, 19, 36, 37, 47], load balancing [40, 43], database [55], and file system [32].

Joint design of network and applications: Co-designing of networks and applications is not a new idea. RedN [49] integrates self-modifying RDMA chains of existing RDMA as Turing machines for key-value stores. RMC [15] is an extension of RDMA to install primitives in SmartNICs to optimize key-value stores. PRISM [17] provides a library of RDMA extensions for better support applications like key-value stores, block storage, and distributed transaction protocols. It points out the CPU involvement problem in post-RDMA processing, similar to the scenarios in Figure 10(b). Works like Cliquemap [52], FaRM [20], and Pilaf [45] studies how to make better usage of the existing RDMA interfaces to improve specific applications' performance.

6 CONCLUSION

This paper shows how the design of the EBS network evolves with the developments of storage media, service scales, computation architectures, hardware configurations, *etc.* in AliCloud. The evolution will continue driven by the desire for better performance, lower cost, and more cloud scenarios (*e.g.,* private and edge clouds). In addition, LUNA demonstrates the tremendous gains brought by kernel bypassing, and SOLAR further improves the performance, reliability, and efficiency by offloading the entire EBS data path into hardware. One promising direction of the future EBS network is hardware offloading in newly emerging chips like DPU, and we demonstrate that EBS' data path can be an excellent fit to the programmable packet processing pipeline originally designed for offloading VPC.

ACKNOWLEDGMENT

We would like to thank our shepherd Ming Liu and SIGCOMM reviewers for their valuable feedback.

REFERENCES

[1] 2016. Dancing on the Lip of the Volcano: Chosen Ciphertext Attacks on Apple iMessage. https://www.usenix.org/sites/default/files/conference/protected-files/security16_slides_garman.pdf. (2016).

[2] 2017. Amazon EC2 Bare Metal Instances. https://aws.amazon.com/blogs/aws/new-amazon-ec2-bare-metal-instances-with-direct-access-to-hardware/. (2017).

[3] 2018. Alibaba Cloud ECS Bare Metal Instance. https://www.alibabacloud.com/product/ebm/. (2018).

[4] 2020. Alibaba Cloud EBS performance. https://www.alibabacloud.com/help/en/doc-detail/25382.html. (2020).

[5] 2020. Amazon EBS features. https://aws.amazon.com/ebs/features/. (2020).

[6] 2020. Fungible F1. https://www.fungible.com/product/dpu-platform/. (2020).

[7] 2020. Google Cloud Block Storage Performance. https://cloud.google.com/compute/docs/disks/performance. (2020).

[8] 2020. Microsoft Azure Disk Storage. https://azure.microsoft.com/en-us/services/storage/disks/. (2020).

[9] 2021. Intel Mount Evans. https://www.intel.com/content/www/us/en/products/platforms/details/mount-evans.html. (2021).

[10] 2021. Mellanox User Manual. https://docs.nvidia.com/networking/pages/viewpage.action?pageId=19812618. (2021).

[11] 2022. Nvidia BlueField. https://www.nvidia.com/en-us/networking/products/data-processing-unit/. (2022).

[12] 2022. Pensando Distributed Services Card. https://pensando.io/products/dsc/. (2022).

[13] 2022. Resource Director Technology. https://www.intel.com/content/www/us/en/architecture-and-technology/resource-director-technology.html. (2022).

[14] 2022. VMware Cloud. https://www.vmware.com/cloud-solutions/multi-cloud.html. (2022).

[15] Emmanuel Amaro, Zhihong Luo, Amy Ousterhout, Arvind Krishnamurthy, Aurojit Panda, Sylvia Ratnasamy, and Scott Shenker. Remote Memory Calls. In *HotNets, 2020*.

[16] Behnaz Arzani, Selim Ciraci, Boon Thau Loo, Assaf Schuster, and Geoff Outhred. Taking the Blame Game out of Data Centers Operations with NetPoirot. In *SIGCOMM, 2016*.

[17] Matthew Burke, Sowmya Dharanipragada, Shannon Joyner, Adriana Szekeres, Jacob Nelson, Irene Zhang, and Dan RK Ports. PRISM: Rethinking the RDMA Interface for Distributed Systems. In *SOSP, 2021*.

[18] Huynh Tu Dang, Marco Canini, Fernando Pedone, and Robert Soulé. Paxos made switch-y. *CCR, 2016*.

[19] Huynh Tu Dang, Daniele Sciascia, Marco Canini, Fernando Pedone, and Robert Soulé. Netpaxos: Consensus at network speed. In *SOSR, 2015*.

[20] Aleksandar Dragojević, Dushyanth Narayanan, Miguel Castro, and Orion Hodson. FaRM: Fast remote memory. In *NSDI, 2014*.

[21] Jiawei Fei, Chen-Yu Ho, Atal N Sahu, Marco Canini, and Amedeo Sapio. Efficient sparse collective communication and its application to accelerate distributed deep learning. In *SIGCOMM, 2021*.

[22] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, Harish Kumar Chandrappa, Somesh Chaturmohta, Matt Humphrey, Jack Lavier, Norman Lam, Fengfen Liu, Kalin Ovtcharov, Jitu Padhye, Gautham Popuri, Shachar Raindel, Tejas Sapre, Mark Shaw, Gabriel Silva, Madhan Sivakumar, Nisheeth Srivastava, Anshuman Verma, Qasim Zuhair, Deepak Bansal, Doug Burger, Kushagra Vaid, David A. Maltz, and Albert Greenberg. Azure accelerated networking: SmartNICs in the public cloud. In *NSDI, 2018*.

[23] Yixiao Gao, Qiang Li, Lingbo Tang, Yongqing Xi, Pengcheng Zhang, Wenwen Peng, Bo Li, Yaohui Wu, Shaozong Liu, Lei Yan, Fei Feng, Yan Zhuang, Fan Liu, Pan Liu, Xingkui Liu, Zhongjie Wu, Junping Wu, Zheng Cao, Chen Tian, Jinbo Wu, Jiaji Zhu, Haiyong Wang, Dennis Cai, and Jiesheng Wu. When Cloud Storage Meets RDMA. In *NSDI, 2021*.

[24] Nadeen Gebara, Manya Ghobadi, and Paolo Costa. In-network aggregation for shared machine learning clusters. *MLSys, 2021*.

[25] Richard L. Graham, Devendar Bureddy, Pak Lui, Hal Rosenstock, Gilad Shainer, Gil Bloch, Dror Goldenerg, Mike Dubman, Sasha Kotchubievsky, Vladimir Koushnir, Lion Levi, Alex Margolin, Tamir Ronen, Alexander Shpiner, Oded Wertheim, and Eitan Zahavi. Scalable hierarchical aggregation protocol (SHArP): a hardware architecture for efficient data reduction. In *COMHPC, 2016*.

[26] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, and Marina Lipshteyn. RDMA over commodity ethernet at scale. In *SIGCOMM, 2016*.

[27] Jaehyun Hwang, Qizhe Cai, Ao Tang, and Rachit Agarwal. TCP ≈ RDMA: CPU-efficient Remote Storage Access with i10. In *NSDI, 2020*.

[28] EunYoung Jeong, Shinae Wood, Muhammad Jamshed, Haewon Jeong, Sunghwan Ihm, Dongsu Han, and KyoungSoo Park. mtcp: a highly scalable user-level TCP stack for multicore systems. In *NSDI, 2014*.

[29] Xin Jin, Xiaozhou Li, Haoyu Zhang, Nate Foster, Jeongkeun Lee, Robert Soulé, Changhoon Kim, and Ion Stoica. Netchain: Scale-free sub-rtt coordination. In *NSDI, 2018*.

[30] Xin Jin, Xiaozhou Li, Haoyu Zhang, Robert Soulé, Jeongkeun Lee, Nate Foster, Changhoon Kim, and Ion Stoica. Netcache: Balancing key-value stores with fast in-network caching. In *SOSP, 2017*.

[31] Anuj Kalia, Michael Kaminsky, and David Andersen. Datacenter RPCs can be general and fast. In *NSDI, 2019*.

[32] Jongyul Kim, Insu Jang, Waleed Reda, Jaeseong Im, Marco Canini, Dejan Kostić, Youngjin Kwon, Simon Peter, and Emmett Witchel. LineFS: Efficient SmartNIC Offload of a Distributed File System with Pipeline Parallelism. In *SOSP, 2021*.

[33] Ana Klimovic, Heiner Litz, and Christos Kozyrakis. Reflex: Remote flash ≈ local flash. *ASPLOS, 2017*.

[34] Gautam Kumar, Nandita Dukkipati, Keon Jang, Hassan M. G. Wassel, Xian Wu, Behnam Montazeri, Yaogong Wang, Kevin Springborn, Christopher Alfeld, Michael Ryan, David Wetherall, and Amin Vahdat. Swift: Delay is Simple and Effective for Congestion Control in the Datacenter. In *SIGCOMM, 2020*.

[35] ChonLam Lao, Yanfang Le, Kshiteej Mahajan, Yixi Chen, Wenfei Wu, Aditya Akella, and Michael M Swift. ATP: In-network Aggregation for Multi-tenant Learning.. In *NSDI, 2021*.

[36] Jialin Li, Ellis Michael, and Dan RK Ports. Eris: Coordination-free consistent transactions using in-network concurrency control. In *SOSP, 2017*.

[37] Jialin Li, Ellis Michael, Naveen Kr Sharma, Adriana Szekeres, and Dan RK Ports. Just say NO to paxos overhead: Replacing consensus with network ordering. In *OSDI, 2016*.

[38] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, and Minlan Yu. HPCC: High Precision Congestion Control. In *SIGCOMM, 2019*.

[39] Ming Liu, Liang Luo, Jacob Nelson, Luis Ceze, Arvind Krishnamurthy, and Kishore Atreya. Incbricks: Toward in-network computation with an in-network cache. In *ASPLOS, 2017*.

[40] Zaoxing Liu, Zhihao Bai, Zhenming Liu, Xiaozhou Li, Changhoon Kim, Vladimir Braverman, Xin Jin, and Ion Stoica. Distcache: Provable load balancing for large-scale storage systems with distributed caching. In *FAST, 2019*.

[41] Dave Maltz. Scaling challenges in cloud networking *(Microsoft Research Summit 2021)*.

[42] Michael Marty, Marc de Kruijf, Jacob Adriaens, Christopher Alfeld, Sean Bauer, Carlo Contavalli, Michael Dalton, Nandita Dukkipati, William C Evans, Steve Gribble, Nicholas Kidd, Roman Kononov, Gautam Kumar, Carl Mauer, Emily Musick, Lena Olson, Mike Ryan, Erik Rubow, Kevin Springborn, Paul Turner, Valas Valancius, Xi Wang, and Amin Vahdat. Snap: A microkernel approach to host networking. In *SOSP, 2019*.

[43] Rui Miao, Hongyi Zeng, Changhoon Kim, Jeongkeun Lee, and Minlan Yu. Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics. In *SIGCOMM, 2017*.

[44] Jaehong Min, Ming Liu, Tapan Chugh, Chenxingyu Zhao, Andrew Wei, In Hwan Doh, and Arvind Krishnamurthy. Gimbal: enabling multi-tenant storage disaggregation on SmartNIC JBOFs. In *SIGCOMM, 2021*.

[45] Christopher Mitchell, Yifeng Geng, and Jinyang Li. Using One-Sided RDMA Reads to Build a Fast, CPU-Efficient Key-Value Store. In *ATC, 2013*.

[46] Tian Pan, Nianbing Yu, Chenhao Jia, Jianwen Pi, Liang Xu, Yisong Qiao, Zhiguo Li, Kun Liu, Jie Lu, Jianyuan Lu, Enge Song, Jiao Zhang, Tao Huang, and Shunmin Zhu. Sailfish: Accelerating cloud-scale multi-tenant multi-service gateways with programmable switches. In *SIGCOMM, 2021*.

[47] Dan RK Ports, Jialin Li, Vincent Liu, Naveen Kr Sharma, and Arvind Krishnamurthy. Designing distributed systems using approximate synchrony in data center networks. In *NSDI, 2015*.

[48] Costin Raiciu, Sebastien Barre, Christopher Pluntke, Adam Greenhalgh, Damon Wischik, and Mark Handley. Improving datacenter performance and robustness with multipath TCP. *SIGCOMM, 2011*.

[49] Waleed Reda, Marco Canini, Dejan Kostić, and Simon Peter. RDMA is Turing complete, we just did not know it yet! *NSDI, 2022*.

[50] Amedeo Sapio, Marco Canini, Chen-Yu Ho, Jacob Nelson, Panos Kalnis, Changhoon Kim, Arvind Krishnamurthy, Masoud Moshref, Dan RK Ports, and Peter Richtárik. Scaling distributed machine learning with in-network aggregation. *NSDI, 2021*.

[51] Leah Shalev, Hani Ayoub, Nafea Bshara, and Erez Sabbag. A Cloud-Optimized Transport Protocol for Elastic and Scalable HPC. *IEEE Micro* 40, 6 (2020), 67–73.

[52] Arjun Singhvi, Aditya Akella, Maggie Anderson, Rob Cauble, Harshad Deshmukh, Dan Gibson, Milo MK Martin, Amanda Strominger, Thomas F Wenisch, and Amin Vahdat. Cliquemap: Productionizing an rma-based distributed caching system. In *SIGCOMM, 2021*.

[53] Arjun Singhvi, Aditya Akella, Dan Gibson, Thomas F Wenisch, Monica Wong-Chan, Sean Clark, Milo MK Martin, Moray McLaren, Prashant Chandra, Rob Cauble, Hassan M. G. Wassel, Behnam Montazeri, Simon L. Sabato, Joel Scherpelz, and Amin Vahdat. 1RMA: Re-Envisioning Remote Memory Access for Multi-Tenant Datacenters. In *SIGCOMM, 2020*.

[54] Bingchuan Tian, Jiaqi Gao, Mengqi Liu, Ennan Zhai, Yanqing Chen, Yu Zhou, Li Dai, Feng Yan, Mengjing Ma, Ming Tang, Jie Lu, Xionglie Wei, Hongqiang Harry Liu, Ming Zhang, Chen Tian, and Minlan Yu. Aquila: a practically usable verification system for production-scale programmable data planes. In *SIGCOMM,*

2021.

[55] Muhammad Tirmazi, Ran Ben Basat, Jiaqi Gao, and Minlan Yu. Cheetah: Accelerating database queries with switch pruning. In *SIGMOD, 2020*.

[56] Qiuping Wang, Jinhong Li, Patrick PC Lee, Tao Ouyang, Chao Shi, and Lilong Huang. Separating Data via Block Invalidation Time Inference for Write Amplification Reduction in Log-Structured Storage. In *FAST, 22*.

[57] Siyu Yan, Xiaoliang Wang, Xiaolong Zheng, Yinben Xia, Derui Liu, and Weishan Deng. ACC: Automatic ECN tuning for high-speed datacenter networks. In *SIGCOMM, 2021*.

[58] Qiao Zhang, Guo Yu, Chuanxiong Guo, Yingnong Dang, Nick Swanson, Xinsheng Yang, Randolph Yao, Murali Chintalapati, Arvind Krishnamurthy, and Thomas Anderson. Deepview: Virtual Disk Failure Diagnosis and Pattern Detection for Azure. In *NSDI, 2018*.

[59] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang. Congestion control for large-scale RDMA deployments. *SIGCOMM, 2015*.