More Than Capacity: Performance-oriented Evolution of Pangu in Alibaba

Qiang Li, *Alibaba Group;* Qiao Xiang, *Xiamen University;* Yuxin Wang, Haohao Song, and Ridi Wen, *Xiamen University and Alibaba Group;* Wenhui Yao, Yuanyuan Dong, Shuqi Zhao, Shuo Huang, Zhaosheng Zhu, Huayong Wang, and Shanyang Liu, Lulu Chen, Zhiwu Wu, Haonan Qiu, Derui Liu, Gexiao Tian, Chao Han, Shaozong Liu, Yaohui Wu, Zicheng Luo, Yuchao Shao, Junping Wu, Zheng Cao, Zhongjie Wu, Jiaji Zhu, and Jinbo Wu, *Alibaba Group;* Jiwu Shu, *Xiamen University;* Jiesheng Wu, *Alibaba Group*

https://www.usenix.org/conference/fast23/presentation/li-qiang-deployed

This paper is included in the Proceedings of the 21st USENIX Conference on File and Storage Technologies.

February 21–23, 2023 • Santa Clara, CA, USA

978-1-939133-32-8

Open access to the Proceedings of the 21st USENIX Conference on File and Storage Technologies is sponsored by

NetApp®

More Than Capacity: Performance-Oriented Evolution of Pangu in Alibaba

Qiang Li[◇], *Qiao Xiang*[†], *Yuxin Wang*[†◇], *Haohao Song*[†◇], *Ridi Wen*[†◇],
Wenhui Yao[◇], *Yuanyuan Dong*[◇], *Shuqi Zhao*[◇], *Shuo Huang*[◇], *Zhaosheng Zhu*[◇],
Huayong Wang[◇], *Shanyang Liu*[◇], *Lulu Chen*[◇], *Zhiwu Wu*[◇], *Haonan Qiu*[◇], *Derui Liu*[◇],
Gexiao Tian[◇], *Chao Han*[◇], *Shaozong Liu*[◇], *Yaohui Wu*[◇], *Zicheng Luo*[◇],
Yuchao Shao[◇], *Junping Wu*[◇], *Zheng Cao*[◇], *Zhongjie Wu*[◇], *Jiaji Zhu*[◇], *Jinbo Wu*[◇],
Jiwu Shu[†], *Jiesheng Wu*[◇],
[◇]*Alibaba Group,* [†]*Xiamen University*

Abstract

This paper presents how the Pangu storage system continuously evolves with hardware technologies and the business model to provide high-performance, reliable storage services with a 100-μs level of I/O latency. Pangu's evolution includes two phases. In the first phase, Pangu embraced the emergence of solid-state drive (SSD) storage and remote direct memory access (RDMA) network technologies by innovating its file system and designing a user-space storage operating system. As a result, Pangu substantially reduced its I/O latency while providing high throughput and IOPS. In the second phase, Pangu evolved from a volume-oriented storage provider to a performance-oriented one. To adapt to this business model change, Pangu upgraded its infrastructure with storage servers of much higher SSD volume and RDMA bandwidth from 25 Gbps to 100 Gbps. It introduced a series of key designs, including traffic amplification reduction, remote direct cache access, and CPU computation offloading, to ensure Pangu fully harvests the performance improvement brought by hardware upgrades. Other than technology innovations, we also shared our operating experiences during Pangu's evolution, and discussed important lessons learned from them.

1 Introduction

Since Alibaba started developing and deploying the Pangu storage system in 2009, Pangu has been serving as a unified storage platform for Alibaba Group and Alibaba Cloud. It provides a scalable, high-performance, and reliable storage service for Alibaba's core businesses (*e.g.*, Taobao, Tmall, AntFin, and Alimama). Many cloud services, such as Elastic Block Storage (EBS) [1], Object Storage Service (OSS) [2], Network-Attached Storage (NAS) [3], PolarDB [4], and MaxCompute [5], are built on top of Pangu. After over a decade, Pangu has become a global storage system with a volume of exabytes and manages trillions of files.

Pangu 1.0: volume-oriented storage service provision. The development and deployment of Pangu go through two generations. Pangu 1.0 spanned from 2009 to 2015. It was designed on an infrastructure composed of servers with commodity CPUs and hard disk drives (HDD), which have a *ms*-level I/O latency, and a Gbps-level datacenter network. Pangu 1.0 designed a distributed kernel-space file system based on Linux Ext4 [6] and kernel-space TCP [7], and gradually added support to multiple file types (*e.g.*, TempFile, LogFile, and random access file) as needed by different storage services. This period overlaps with the early stage of cloud computing. Although Pangu 1.0's performance (*i.e.*, throughput and I/O latency) reached the limit of HDD and Gbps-level networks, clients' primary focus was to get large volumes of space to store their data, rather than high performance.

New hardware technologies require new designs. Since 2015, we started to design and develop Pangu 2.0 to embrace the emerging SSD and RDMA technologies. The goal of Pangu 2.0 is to *provide high-performance storage services with a 100µs-level I/O latency*. Although SSD and RDMA can achieve high-performance, low-latency I/O in storage and network, we observe that (1) multiple file types used in Pangu 1.0, in particular file types that allow random access, perform poorly on SSD, which can achieve high throughput and IOPS on sequential operations; (2) the kernel-space software stack cannot keep up with the high IOPS and low I/O latency of SSD and RDMA, due to data copy and frequent interrupts; and (3) the paradigm shift from server-centric datacenter architectures to resource-disaggregated datacenter architectures poses additional challenges to achieving low I/O latency.

Phase one of Pangu 2.0: embracing SSD and RDMA by file system refactoring and user-space storage operating system. To achieve high-performance and low-latency I/O, in this phase, Pangu 2.0 first proposed new designs in key components of its file system. To simplify the development and management of the overall system, it designed a unified, append-only persistence layer. It also introduced a self-contained chunk layout to reduce the I/O latency of file write operations. Second, Pangu 2.0 designed a user-space storage operating system (USSOS). USSOS uses a run-to-completion thread model to realize efficient collaboration between the user-space storage stack and the user-space network stack. It also proposes a user-space scheduling mechanism for ef-

ficient CPU and memory resource allocation. Third, Pangu 2.0 deployed mechanisms to provide SLA guarantees under dynamic environments. With these innovations, Pangu 2.0 achieved a ms-level P999 I/O latency in phase one.

Phase two of Pangu 2.0: adapting to a performance-oriented business model with infrastructure updates and breaking through network/memory/CPU bottlenecks. Since 2018, Pangu gradually changed its business model from volume-oriented to performance-oriented. It is because enterprises are increasingly moving their businesses to Alibaba Cloud and they have stringent requirements on storage performance and latency. This shift became faster after the COVID-19 pandemic broke out in 2020. To adapt to this business model change and the fast expansion of clientele, Pangu 2.0 needed to keep upgrading the infrastructure.

Scaling the infrastructure with original servers and switches along a Clos-based topology (*e.g.*, FatTree [8]) is not economical, including a higher total cost of ownership (*e.g.*, rack space, power, cooling, and labor) and a higher environmental cost (*e.g.*, a higher carbon emission rate). As such, Pangu developed in-house high-volume storage servers (96 TB SSD per server) and upgrades network bandwidth from 25 Gbps to 100 Gbps.

To fully harvest the performance improvement brought by these upgrades, Pangu 2.0 proposed a series of techniques to cope with the performance bottleneck at network, memory, and CPU and fully utilize its massive resources. Specifically, Pangu 2.0 optimized network bandwidth by reducing the network traffic amplification ratio and dynamically adjusting the priorities of different traffic. It coped with the memory bottleneck by proposing remote direct cache access (RDCA). It addressed the CPU bottleneck by eliminating the data tax of data (de)serialization and introducing *CPU wait* instruction to synchronize hyper-threading.

High performance in production. By the end of phase one, Pangu 2.0 successfully supported the elastic SSD block storage service with a 100μs-level I/O latency and 1M IOPS. During Alibaba's Double 11 Festival in 2018, Pangu 2.0 supported the Alibaba database service with a latency of 280 μs. For the OTS storage service [9], with the same hardware, its I/O latency in Pangu 2.0 is lower than that in Pangu 1.0 by an order of magnitude. For write-intensive services (*e.g.*, EBS cloud drive), the P999 I/O latency is less than 1 ms. For read-intensive services (*e.g.*, online search), the P999 I/O latency is less than 11 ms.

In phase two, by upgrading the network from 2×25 Gbps to 2×100 Gbps and breaking through the bottlenecks of network, memory and CPU, the normalized effective throughput per Taishan storage server increases by 6.1×.

2 Background

2.1 Overview of Pangu

Pangu is a large-scale distributed storage system. It consists of Pangu Core, Pangu Service, and Pangu Monitoring System (Figure 1). Pangu Core consists of clients, masters, and chunkservers, and provides an append-only persistence semantic. The clients provide SDK to Pangu cloud storage services (*e.g.*, EBS and OSS), and are responsible for receiving file requests from these services and communicating with the masters and the chunkservers to fulfill these requests. Similar to other distributed file systems (*e.g.*, Tectonic [10] and Colossus [11]), the clients in Pangu are heavyweight and play a key role in the replica management, SLA guarantee, and data consistency management of Pangu.

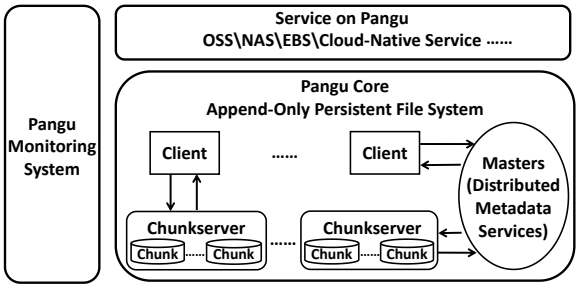

Figure 1: The architecture of Pangu.

The masters manage all the metadata in Pangu and use a Raft-based protocol to maintain metadata consistency [12]. For better horizontal scalability and extensibility (*e.g.*, hundreds of billions of files), the Pangu masters decompose the monolithic metadata service into two separate services: the namespace service and the stream meta service, where a stream is an abstraction of a group of chunks. Both services first partition metadata by directory tree to achieve metadata locality, then further partition these groups using hashing to achieve good load balancing [13]. While the namespace service provides information on files (*e.g.*, directory tree and the namespace), the stream meta service provides the mapping from files to chunks (*i.e.*, the locations of chunks). Chunkservers store data in chunks and are equipped with a customized user-space storage file system (USSFS). The USSFS provides high-performance, append-only storage engines for different hardware (*e.g.*, SMRSTORE for HM-SMR drives [14]). In the early days (*i.e.*, phase one of Pangu 2.0), each file is stored in chunkservers with three replicas, and later a garbage collection worker (GCWorker) performs garbage collection and stores the file using erasure coding (EC). In phase two of Pangu 2.0, we gradually replace the 3-way replication with EC in key businesses (*e.g.*, EBS) to reduce the traffic amplification in Pangu (§4.1.2).

On top of the Pangu Core, the Pangu Service provides traditional cloud storage services (*e.g.*, EBS, OSS, and NAS) and cloud-native storage services through a cloud-native-oriented file system (*i.e.*, Fisc [15]). The Pangu Monitoring System (*e.g.*, Perseus [16]) provides real-time monitoring and AI-assisted root cause analysis services to both the Pangu Core and the Pangu Service. The infrastructure of Pangu Core, the Pangu Service, and the Pangu Monitoring System are interconnected using high-speed networks [17, 18].

2.2 Design Goals of Pangu 2.0

To cope with the emergence of hardware technologies and the shift of business model, Pangu 2.0 aims to achieve the following goals:

- **Low latency**: Pangu 2.0 aims to leverage the low latency characteristics of SSD and RDMA, to reach an average 100μs-level I/O latency in a computation-storage disaggregated architecture, and provide a ms-level P999 SLA even under environment dynamics such as network traffic jitters and server failures.
- **High throughput**: Pangu 2.0 aims to reach an effective throughput on storage servers that approaches their capacity.
- **Unified high-performance support to all services**: Pangu 2.0 aims to provide unified high-performance support to all services running on top of it, such as online search, data streaming analytics, EBS, OSS, and database.

2.3 Related Work

Many distributed storage systems have been designed and deployed [10, 19–21]. Some are open-sourced ones (*e.g.*, HDFS [20] and Ceph [22]), and some are proprietary ones used by different industry organizations (*e.g.*, GFS [19], Tectonic [10], and AWS [23]).

Pangu is a proprietary storage system of Alibaba Group. It provides storage infrastructure support for Alibaba's core businesses and Alibaba Cloud. In the past few years, we have shared our experiences in different aspects of Pangu, such as the large-scale deployment of RDMA [17], the key-value engine for scale-out cloud storage services [24], the co-design of network and storage software stack for the EBS storage service [18], and some key designs of the namespace metadata service [13]. This paper focuses on introducing our experience in evolving Pangu to provide unified low-latency, high-throughput storage services to support all Alibaba's businesses and Alibaba Cloud, in response to the emergence of hardware technologies and the shift of Pangu's business model.

3 Phase One: Embracing SSD and RDMA

In this section, we introduce how Pangu embraces the emergence of SSD and RDMA to provide high-performance, reliable storage services with low I/O latency. Compared with HDD and TCP, SSD and RDMA technologies substantially reduce the I/O latency in storage and network, respectively. However, integrating these two technologies into Pangu, a large, distributed storage system with a mature architecture, is not without challenges. To this end, Pangu introduces a series of new designs in key components of its file system (§3.1) and develops a user-space storage operating system (§3.2) to achieve a high-throughput, high IOPS performance with a 100μs-level I/O latency. It also deploys novel mechanisms to provide such SLA guarantees under dynamic environments, *e.g.*, straggler and transient/permanent failures (§3.3).

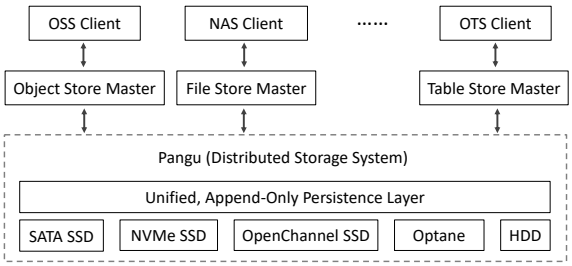

Figure 2: Various businesses are based on the unified persistence layer.

3.1 Append-Only File System

As shown in Figure 1, the core base layer of Pangu consists of the masters, chunkservers, and clients. Pangu first introduces a unified, append-only persistence layer with an append-only interface called FlatLogFile to simplify its architecture (§3.1.1). FlatLogFile is friendly to SSDs with high throughput and low latency. Then Pangu adopts a variety of designs to improve its performance. Specifically, the Pangu client is heavyweight to meet the requirements of different storage services (§3.1.2). Based on FlatLogFile, Pangu adopts the append-only chunks and uses a self-contained chunk layout to manage chunks on chunkservers (§3.1.3). Pangu implements distributed metadata management on the master to realize efficient metadata operation (§3.1.4).

3.1.1 Unified, Append-Only Persistence Layer

The persistence layer of Pangu provides interfaces to all Pangu's storage services (*e.g.*, EBS, OSS, and NAS). In the early development of Pangu, the persistence layer provides different interfaces to different storage services. For example, it provides the LogFile interface to low-latency NAS services, and the TempFile interface to high-throughput Maxcompute data analytics services. However, this design brings substantial development and management complexities. Specifically, for every storage service, Pangu developers must design and implement a new interface. That is a complex, labor-intensive and error-prone process.

To simplify the development and management of Pangu and make sure all storage services can achieve high-performance, low-latency I/O on SSD, motivated by the layered architecture of computer networks, Pangu introduces a unified file type called FlatLogFile (Figure 2). Specifically, FlatLogFile has an append-only semantic, and upper-layer services (*e.g.*, OSS) can equip a key-value-like mapping to update their data and a garbage collection mechanism to compress their historical data. FlatLogFile provides a simple, unified interface for storage services to perform data operations. Furthermore, Pangu developers must ensure that data operations via FlatLogFile, especially the write operations, can be executed efficiently and reliably on storage media. As such, all upgrades and changes to storage services are transparent to Pangu developers, substantially simplifying the development and management of Pangu.

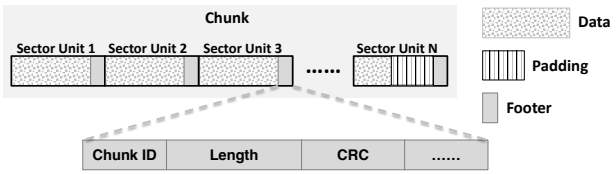

Figure 3: The self-contained chunk layout.

Under the hood, we observe that SSD can achieve high throughput and IOPS on sequential operations due to its intrinsic characteristics of the storage unit and flash transaction layer. To make sure data operations via FlatLogFile can be executed on SSD efficiently, we align the sequential operations on FlatLogFile to achieve high performance.

3.1.2 Heavyweight Client

We design Pangu's client as a heavyweight one. It is responsible for the data operations with chunkservers and the metadata information retrieval and updates with the masters. After getting the chunk information from the masters, a Pangu client is in charge of the corresponding replication protocol and EC protocol. The client is equipped with retry mechanisms (*e.g.*, backup read in §3.3) to cope with occasional jitters in Pangu (*e.g.*, network packet drop) in order to improve the I/O SLA. It also deploys probing mechanisms to periodically get the latest chunkserver status from the masters and evaluate the quality of services of chunkservers. Similar to the client of Facebook's Tectonic file system [10], the Pangu client can select appropriate write or read parameters to meet the specific requirements of different storage services (*e.g.*, EBS and OSS).

3.1.3 Append-only Chunk Management

Typical file systems (such as Ext4 [6]) store files in blocks. A file and its metadata are written to the storage media separately with two SSD write operations. Not only does it increase the latency of the file write, it also shortens the lifespan of SSDs. As such, adopting this design in Pangu 2.0 would result in high latency for file access, and lead to a high hardware replacement cost.

To address this issue, Pangu chooses to store files in chunks, which have the append-only semantic based on FlatLogFile, on chunkservers, and designs a self-contained chunk layout, where each chunk stores both data and its own metadata. As such, a chunk can be written into the storage media in one operation instead of two, substantially reducing the write latency and improving the lifespan of the storage media. Figure 3 shows the layout of the chunk. A chunk includes multiple sector units, where each sector unit includes 3 elements: data, padding, and footer. The footer stores chunk metadata, such as chunk ID, chunk length, and the CRC checksum.

The self-contained chunk layout also allows the chunkserver to recover from failures by itself. Specifically, when a client writes consecutive self-contained chunks to the storage media, the chunkserver stores a copy of the metadata of these chunks in the memory, and periodically takes checkpoints of this information to the storage media. When a failure happens and leads to some unfinished write operations, the chunkserver loads the metadata from the checkpoints and compares it with the metadata in chunks. If discrepancies happen, the chunkserver checks the CRC of the chunks to recover to the latest correct state.

3.1.4 Metadata Operation Optimization

The Pangu masters provide two metadata services: the namespace service is responsible for directory tree and file management, and the stream service is responsible for chunk information. Stream is an abstraction of a group of chunks. Chunks in the same stream belong to the same file. Both services use a distributed architecture for better scalability. They partition metadata by considering metadata locality and load balancing (*i.e.*, partition by directory tree first and then by hashing). We design multiple mechanisms to optimize the efficiency of metadata operations.

Parallel metadata processing. Both namespace and stream services adopt parallelization processing (*e.g.*, InfiniFS [13]) to meet the low-latency requirement of metadata access. Specifically, Pangu uses a hash algorithm to map highly cohesive metadata to different metadata servers. It also uses a new data structure that supports predictable directory IDs and allows clients to perform path parsing efficiently in parallel.

We also introduce several techniques to accelerate how clients retrieve chunk information from the stream service.

Big chunks with variable lengths. Pangu 2.0 chooses to use big chunks. This decision has three benefits. It reduces the number of metadata. It avoids unnecessary I/O latency caused by clients frequently requesting chunks. It also helps improve the lifespan of SSD. However, simply increasing the chunk size will increase the risk of fragmentation. As such, We introduce variable-length chunks (*e.g.*, sizes ranging from 1 MB to 2 GB). For example, the chunk size of the EBS service has a 95% quantile of 64 MB and a 99% quantile of 286.4 MB. Variable-chunk length reduces the probability of fragmentation and maintains compatibility with Pangu 1.0.

Caching chunk information at clients. Each client maintains a local metadata cache pool to reduce the number of metadata query requests. The pool is updated using an LRU-based mechanism. When an application wants to access the data, the client first queries its metadata cache. It issues a new request to the masters to get the up-to-date metadata when (1) its cache is not hit; or (2) the cache is hit, but in response to the request, the corresponding chunkserver notifies the client that its cached metadata becomes stale (*e.g.*, due to replica migration).

Chunk information request batching. We let each client aggregate multiple chunk information requests over a short interval and send them in a batch to the masters to improve the query efficiency. The masters process the batched requests in parallel, aggregate the results and send them back to the client. The client disaggregates the results and dispatches them to corresponding applications.

Speculative chunk information prefetching. We design a greedy, probabilistic prefetching mechanism to reduce the number of chunk information requests. When the masters receive a read request, they respond to the client with the metadata of the related chunk and the metadata of other chunks. When the masters receive a write request, the masters respond with more available chunks than the client requested. In this way, the client can switch chunks without requesting new chunks if it encounters write exceptions.

Data piggybacking to reduce one RTT. We are motivated by QUIC [25] and HTTP3 [26] to use data piggybacking to improve the write latency. Specifically, after a client retrieves the chunk address from the masters, it merges the chunk creation request and the data to write into one request and sends it to the chunkserver. As a result, we are able to reduce the write latency by one RTT.

3.2 Chunkserver USSOS

In Pangu, the chunkserver is responsible for carrying out all the data operations. As such, it is essential to carefully design the run-time operating system to ensure that data operations can be finished with low latency and high throughput. With the emerging high-speed network technology and storage media, sticking to the traditional design that puts data operations through kernel space is inefficient. In particular, this would incur not only frequent system interrupts, which consume CPU resources, but also unnecessary data duplication between the user space and kernel space.

To cope with these issues, we resort to the kernel-bypassing design to develop a high-performance user-space storage operation system [17] for the chunkserver, which provides a unified user-space storage software platform. Aside from realizing device management and run-to-completion thread model [17, 18] in USSOS, Pangu also realizes user-level memory management (§3.2.1), lightweight user-space scheduling strategy (§3.2.2), and a customized high-performance append-only user-space storage file system (USSFS) for SSDs (§3.2.3).

3.2.1 User-Level Memory Management

Chunkserver USSOS is built based on existing user-space technologies (*e.g.*, RDMA in the network stack, DPDK [27], and SPDK [28] in the storage stack). But we go beyond and unify these two stacks to further reduce the latency and improve the performance of data operations. First, we make use of the run-to-completion thread model. In the traditional pipeline thread model, a request is decomposed into individual stages and each stage runs on a thread. In contrast, in USSOS, the request is run on one thread from beginning to end in the run-to-completion model, reducing the overhead caused by context switch, inter-thread communication, and inter-thread synchronization. Second, the thread requests a huge-page memory space to serve as a shared memory between the network and the storage stacks. To be concrete, data received from the network can be stored in this shared huge-page memory using RDMA protocol. After sending the metadata of the huge-page memory (*e.g.*, its address and size), data can be written directly from the huge-page memory to the storage media via SPDK frame. This way, we achieve zero copy between the network and the storage stacks during the data transmission and storage procedure. Besides, through the user-level shared huge-page memory for I/O data, data transmission operations among different roles (*e.g.*, the chunkserver and the garbage collection worker) can also achieve zero copy.

3.2.2 User-Space Scheduling Mechanism

In a real production environment, we encounter performance glitches brought by problems like task scheduling. Here, we introduce three key designs to optimize CPU scheduling in USSOS to improve the performance of Pangu.

Preventing a task from blocking the subsequent ones. As explained in §3.2.1, the run-to-completion thread model helps achieve zero-copy between the user-space network and storage stacks by using shared huge-page memory. However, each chunkserver has a fixed number of working threads. A new request is dispatched to a working thread based on the hash value of the file in the request. Requests assigned to the same working thread are executed in a first-in-first-serve order. As such, given one request, if one of its tasks takes too much time (*e.g.*, table lookup, table search, traversal, memory allocation, monitoring, and statistics), it will hog resources and block subsequent tasks. This issue degrades the performance of this request and leaves other requests to starve. To solve this problem, we take different measures for different scenarios. For heavy tasks, Pangu introduces the heartbeat mechanism to monitor the execution time of tasks and set an alarm. If a task runs out of time slice, it would be put into a background thread to remove it from the critical path. For system overhead, Pangu uses TCMalloc's cache [29] to allow high-frequency operations to be completed in the cache.

Priority scheduling to guarantee high QoS. Pangu assigns different QoS objectives for different requests (*e.g.*, user requests are assigned a high-priority objective while GC requests are assigned a low one). However, a request with a low-priority objective may block a high-priority request that arrives later and is assigned to the same working thread. As a result, it is hard to guarantee that requests with higher QoS objectives can always receive higher priorities. To address this problem, USSOS creates priority queues. Then tasks can be put into the corresponding priority queue according to their QoS objectives.

Polling and event-driven switching (NAPI). USSOS adopts a switching mechanism between polling and event-driven modes to reduce the overhead of massive interrupt processing with a low CPU utilization [30]. Specifically, NIC provides a *fd* monitored by applications and notifies the applications of data arrival through the *fd* event. Applications are in the event-driven mode by default. When they receive a notification from NIC, they enter the polling mode. If they do not receive any I/O request for some time, they switch back to the event-driven mode and notify the NIC.

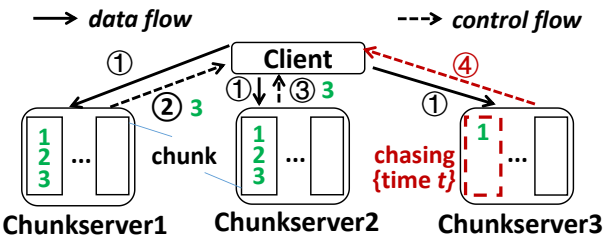

Figure 4: An illustrating example of chasing with $MaxCopy = 3$ and $MinCopy = 2$.

3.2.3 Append-Only USSFS

Previous file systems (*e.g.*, Ext4) could not make full use of the append-only FlatLogFile (§3.1.1) and SSDs with high throughput and low latency. Therefore, Pangu goes beyond and customizes the USSFS, a compact and high-performance user-space storage file system.

With the append-only semantic of FlatLogFile, USSFS supports append-only write and provides a set of chunk-based semantics, such as open, close, seal, format, read, and append, instead of standard POSIX semantics like Ext4. On this basis, it supports the append-only sequential write, which fully leverage of the sequential write-friendly feature of SSDs, and random read of successfully written data. Meanwhile, USSFS adopts different mechanisms to maximize the performance of SSDs. First, it can fully utilize the self-contained chunk layout (§3.1.3) to significantly reduce the number of data operations without using mechanisms such as page cache and journal. Second, it does not establish a hierarchical relationship between inodes and file directories like Ext4. All operations on files are recorded to log files. The corresponding metadata can be rebuilt by replaying the logs when mounting the file system. Third, we use the polling mode instead of an interrupt notification mechanism like Ext4 to maximize the performance of SSDs. Moreover, considering that the capacity of a single SSD node is tens or even hundreds of terabytes and the size of the chunk is usually 64 MB, we set the minimum space allocation granularity in USSFS as 1 MB. This choice considers both the size of memory used by space management metadata and SSD space utilization.

3.3 High Performance SLA Guarantee

Pangu introduces multiple mechanisms to provide high performance and a ms-level P999 SLA guarantee [31] in failure scenarios. *Chasing* copes with abnormal jitters (*e.g.*, network flash and packet retransmission caused by network incast). In these scenarios, exceptions occur in the cluster operating environment, but the system can recover to a normal state automatically in a short time. *Non-stop write* aims at unavailable chunks. *Backup read* reduces latency when the read request can not return within a limited time. *Blacklisting* isolates the chunkservers which provide poor service or fail.

Chasing. We design this mechanism to reduce the impact of system jitters on write latency. It allows the client to return success to the application when *MinCopy* out of *MaxCopy* replicas are successfully written in chunkservers, where $2 \times MinCopy > MaxCopy$. Figure 4 illustrates how chasing works with $MaxCopy = 3$ and $MinCopy = 2$. Suppose the application asks the client to write data $[1,2,3]$ to 3 chunk replicas. At time T, chunkserver 1 and 2 return success to the client's write operation but chunkserver 3 has not. The client returns success to the application. But it keeps the chunk in its memory and waits for another period t, an empirical *ms*-level threshold. If chunkserver 3 returns success to the client before $T+t$, the client releases the chunk from its memory. If chunkserver 3 does not finish the write, but the unfinished part is smaller than an empirical threshold k, the client issues a retry on chunkserver 3. If the unfinished part is larger than k, the client will seal this chunk at chunkserver 3 so that this chunk will not have subsequent append operation. The client then notifies the masters, who will replicate the data on a different chunk from chunkserver 1 or 2 to ensure there are a total of 3 replicas eventually.

Our analysis shows that with a careful choice of t and k, chasing substantially reduces the write tail latency without increasing the risk of data loss. Specifically, taking the case of $MaxCopy = 3$ as an example, after two replicas are successfully written to chunkservers, three replicas are in the system, whereas the third one is in the memory of the client. During $[T, T+t]$, data loss can only happen when the SSDs of the two chunkserver replicas are damaged and the last replica in the client memory also fails or the cluster powers down. Because SSDs' annual failure rate (*i.e.*, $\sim 1.5\%$ [32–37]) and servers' annual downtime rate (*i.e.*, $< 2\%$ [38, 39]) are close, the probability of data loss when two replicas are written successfully and the third replica is chasing is approximately the same as that when all three replicas are written successfully. Pangu has deployed chasing for over a decade and has not experienced any data loss caused by chasing. Recent studies also started to investigate this early-write-acknowledgment mechanism [40–42].

Non-stop write. We design this mechanism to reduce the write latency when a chunk write fails. When the failure happens, the client seals the chunk and reports the successfully written data length to the masters. It then uses a new chunk to continue writing the unfinished data. If the data written to the sealed chunk is corrupted, we use other replicas to duplicate a copy of this data to the new chunk in the background traffic. If no replica is available, the client writes this data to the new chunk again.

Backup read. To reduce the read latency under dynamic environments, the client sends additional read requests to other chunkservers as backups before receiving the response of the previously sent read request. This mechanism has two key parameters, the number and waiting time of sending backup read requests. To this end, Pangu calculates the latency of different disk types and I/O sizes and uses this information to dynamically adjust the time to send backup read requests. It also limits the number of backup read requests to control the system's load.

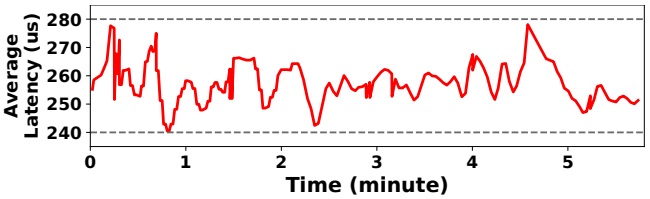

Figure 5: The average latency of database access to Pangu on Double 11 Festival in 2018.

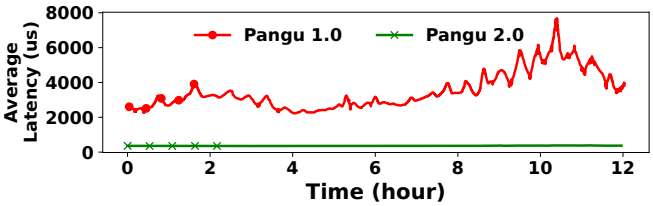

Figure 6: The average latency of OTS querying of Pangu 1.0 and Pangu 2.0 under the same stress test.

Blacklisting. To avoid sending I/O requests to chunkservers with poor service quality, Pangu introduces two blacklists, deterministic blacklist and non-deterministic blacklist. When Pangu determines that a chunkserver is unserviceable (*e.g.*, SSDs of a chunkserver are damaged), this server will be added to the deterministic blacklist. If a chunkserver can provide service, but its latency exceeds a certain threshold, it will be added to the non-deterministic blacklist with a probability that increases with its service latency. If the server's latency exceeds the median latency of all servers by several times, it is directly added to the non-deterministic blacklist with a probability of one. To release servers from these blacklists, clients send I/O probes to those servers periodically (*e.g.*, every second). If a server on the deterministic blacklist successfully returns the response of this request, it will be removed from this blacklist. For a server on the non-deterministic blacklist, Pangu decides whether to remove the server from the blacklist based on the time it takes to receive the response to this request.

Pangu limits the total number of servers on the blacklists to ensure system availability. For each server, it introduces a grace period for adding/removing it to/from the blacklist to maintain system stability. In addition, because the failed servers in the TCP and RDMA links may be different, Pangu maintains separate blacklists for TCP and RDMA links, respectively, and takes I/O probes on both links to update them.

3.4 Evaluations

Figure 5 shows the latency of database (DB) access to Pangu under the peak of 550,000 transactions per second during the Double 11 Festival in 2018. This peak value is at least one order of magnitude larger than that of non-festival days. The DB consists of millions of databases (*e.g.*, databases of Taobao merchants), each of which contains millions of e-commerce users' data. During this process, the DB needs to query orders and record transactions for those users. Under such a peak transactions rate, the access latency is less than 280 μs, which proves the high performance of Pangu 2.0.

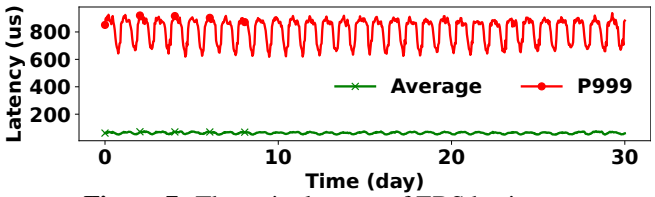

Figure 7: The write latency of EBS business.

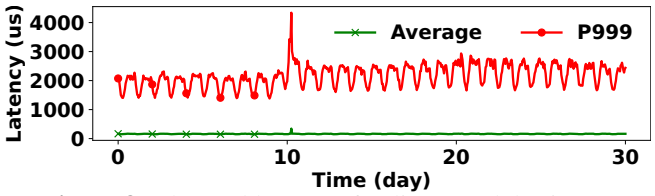

Figure 8: The read latency of online search business.

Figure 6 shows the latency of the cloud product OTS [9] querying with the same SSD and 25 Gbps network. Under the same query pressure, after upgrading from Pangu 1.0 to Pangu 2.0, the query latency is reduced by nearly an order of magnitude. This is mainly due to the low latency of read and write operations and the improved processing capability of a single thread in Pangu 2.0.

Figures 7 and 8 show the average latency and long tail latency of two clusters (online EBS and online search business) within one month. Online EBS is write-intensive and its online read/write ratio is nearly 1:10, which expects to achieve high throughput on write. As shown in Figure 7, its long tail latency is less than 1 ms. Online search business mainly services for the query and recommendation of Alibaba e-commerce (*e.g.*, search for goods on Taobao and Tmall), which expects to achieve high throughput on read. As shown in Figure 8, the long tail latency of read is less than 5 ms within one month. The results exhibit that Pangu 2.0 has a good guarantee for latency SLA.

4 Phase Two: Adapting to Performance-Oriented Business Model

Since 2018, Pangu gradually changes its role from a volume-oriented storage provider to a performance-oriented provider. This change in business model and the fast expansion of Pangu's clientele require Pangu to keep upgrading the infrastructure. However, scaling the infrastructure with original servers and switches along a Clos-based topology is not economical in many ways, including a higher financial and environmental cost (*e.g.*, a higher carbon emission rate). As such, Pangu develops its in-house storage server Taishan. A Taishan server is equipped with 2×24 core Skylake CPUs, 12×8 TB commodity SSDs, 128 GB DDR memory, and 2×dual-port 100 Gbps NICs. Although we could continue to increase its storage volume at the moment, we choose not to do so to maintain a high-level write IOPS/GB to suit the need of the performance-oriented business model. With optimizations made by SSD manufacturers (*e.g.*, caching and channel), the SSD throughput of a single Taishan server can reach more than 20 GB/s.

With such high-performance storage servers, it is natural to observe that other resources (*e.g.*, network, memory, and CPU) become the performance bottleneck of Pangu. As such, we upgrade the network of Pangu from 25 Gbps RDMA to 100 Gbps. However, contrary to common perception, it is *non-trivial* to provide high-performance, low-latency I/O in such an upgraded infrastructure. That is because new hardware also comes with new technical challenges in large-scale deployment. To this end, we propose and deploy a series of novel techniques to optimize the operation of Pangu's massive network (§4.1), memory (§4.2), and CPU resources (§4.3).

4.1 Network Bottleneck

Pangu optimizes the network in two aspects: network bandwidth expansion (§4.1.1) and traffic optimization (§4.1.2).

4.1.1 Bandwidth Expansion

To match the throughput of all SSDs on a single storage node, Pangu upgrades the network bandwidth from 25 Gbps to 100 Gbps to increase its network capability. The success of network bandwidth expansion depends on the improving of software and hardware. For hardware, Pangu adopts high-performance NIC/RNIC, optical modules (QSFP28 DAC, QSFP28 AOC, QSFP28 [43]), single-mode/multi-mode fiber, and high-performance switches. For the network software stack, Pangu first adopts lossless RDMA and proposes various mechanisms to achieve large-scale RDMA deployment, such as shutting down NIC ports or temporialy switching from RDMA to TCP for a short time (*e.g.*, several seconds) when there are too many pause frames on the RDMA network [17]. However, these mechanisms cannot handle other issues of the pause frame based flow control (*e.g.*, deadlocks [44] and head-of-line blocking [45, 46]). As such, Pangu upgrades to lossy RDMA, in which pause frames are disabled, to avoid these problems and improve performance. More details about this bandwidth expansion (*e.g.*, how we address the interoperability issue of heterogeneous network hardware and software) can be found in our early paper [17].

4.1.2 Traffic Optimization

Other than increasing the network capability, we also tackle the network bottleneck by reducing the traffic amplification ratio. Specifically, the traffic amplification ratio is computed as the amount of data transmitted through the network divided by the actual file size. Take the workflow of the EBS service as an example (Figure 9). First, the EBS client sends a file (1x) to the Pangu client (step (a)). Second, the Pangu client transfers the file to 3 storage nodes to write 3 replicas (3x). Third, the garbage collection worker (GCWorker) reads the file (1x) and performs GC on it. For ease of exposition, we ignore the file size change before and after GC. In the end, the file is written back to storage nodes in the form of EC(8,3) (1.375x), which provides at least the same level of fault tolerance as 3-replica but uses less storage space. As such, the traffic amplification ratio of a file write can be up to 6.375x (1x+3x+1x+1.375x). In other words, the maximum data access bandwidth of EBS

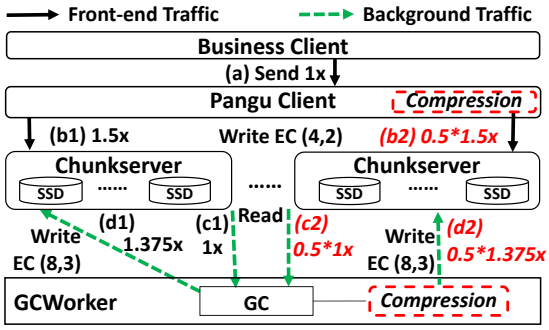

Figure 9: Pangu optimizes network traffic with 3 techniques: EC, compression, and balance between background and front-end traffic. The entire life cycle of a file we define is as follows. First, the business client sends a file (step a) to the Pangu client. Second, the Pangu client writes the file to the chunkserver in way b1 (b2 after compression). Third, the GCWorker reads the file from the chunkserver in way c1 (c2 after client compression) and performs garbage collection. Finally, the GCWorker writes the file in way d1 (d2 after GCWorker compression).

in a 100-Gbps network is less than 16 Gbps. To cope with the issue of the high traffic amplification ratio limiting the service capability of Pangu, we introduce two optimizations: EC and data compression.

Use EC to replace 3-replica. Using EC to replace the 3-replica mechanism can substantially reduce the network traffic amplification while achieving a good level of fault tolerance. Take Figure 9 as an example. If we use EC (4,2) (step (b1)) to replace the 3-replica step, the network traffic amplification ratio can be reduced from 6.375x to 4.875x, the sum of step (a), (b1), (c1), and (d1).

Two challenges arise during this replacement. First, storing small files in EC is expensive because of the large number of zero-paddings needed to perform EC on data with a fixed length. We introduce multiple mechanisms to cope with this waste of space, including small write request aggregation and dynamic switching between EC and 3 replicas. Second, computing EC introduces a non-negligible latency. To this end, Pangu adopts Intel ISA-L [47], which reduces the latency of computing EC by 2.5 to 3 times compared with Jerasure [48].

Compressing FlatLogFile. We observe that FlatLogFile is highly redundant. As such, both the Pangu client and GCWorker compress the file before writing it to further reduce the traffic (*e.g.*, step (b2) and (d2) in Figure 9). We choose the LZ4 algorithm [49] to achieve efficient (de)compression. Empirical data in Pangu shows that the average compression rate can reach 50%. As such, in the example above, the traffic amplification ratios of step (b2), (c2) and (d2) can all be reduced by half. As a result, the traffic amplification ratio can be further reduced from 4.875x to 2.9375x, *i.e.*, the sum of step (a), (b2), (c2), and (d2).

Dynamic bandwidth allocation between front-end and background traffic. We dynamically adjust the *threshold*

of usable network bandwidth for background traffic. For example, if there is sufficient empty space in the whole storage cluster, we temporally decrease the threshold to limit the bandwidth of background traffic (*e.g.*, the GC traffic), and let the frontend traffic use more bandwidth. For Taobao, Pangu sets a low threshold from daytime to midnight to cope with the large number of front-end access requests. After midnight, Pangu increases it because the front-end traffic decreases.

4.2 Memory Bottleneck

The fundamental memory bottleneck in Pangu lies in the high contention of memory bandwidth between network processes (*i.e.*, NIC performing DMA operations) and application processes (*e.g.*, data copy, data replication, and garbage collection) in the receiver host. Because NIC cannot acquire enough memory bandwidth, severe PCIe back-pressure is generated to the NIC. As a result, the NIC buffer is filled with in-flight packets. Eventually, it drops the overflowed ones, triggering the congestion control mechanism in the network and leading to overall performance degradation (*i.e.*, 30% network throughput drop, 5%-10% latency increase and 10% IOPS drop per server). This phenomenon is not unique to Pangu. Google also recently reported this issue [50].

We tackle this memory bandwidth bottleneck in three steps. First, we add more small-capacity DRAMs to the server to fully utilize the memory channels. Second, we switch background traffic from TCP to RDMA to reduce servers' memory bandwidth consumption (§4.2.2). Third, we design remote direct cache access (RDCA) to move memory out of the receiver host datapath and let senders access the receiver's cache directly (§4.2.3).

4.2.1 Adding Small-Capacity DRAMs

Because the bottleneck is memory bandwidth instead of memory capacity, we add more DRAMs with small capacity (*e.g.*, 16 GB) to servers to fully utilize the memory channels and increase the available memory bandwidth per server. We also enable non-uniform memory access (NUMA) to avoid across-sockets memory accesses being constrained by the ultra path interconnect [51].

4.2.2 Shifting Background Traffic From TCP to RDMA

In the 25-Gbps network, Pangu's background traffic was transmitted using TCP. It is to guarantee the QoS of front-end traffic because there is only one hardware queue for RDMA transmission on 25-Gbps switches.

With the network being updated to 100 Gbps, Pangu starts to transmit background traffic using RDMA to reduce the memory bandwidth consumption of network processes. It is because TCP needs at least four more memory copies than RDMA. By switching to RDMA, the memory bandwidth spent by background traffic is reduced by about 75%. To guarantee the QoS of front-end traffic, we design a host rate control mechanism similar to Linux *tc* [52, 53] to control the rate of the background traffic being injected into the network.

4.2.3 Remote Direct Cache Access

In addition to increasing the available memory bandwidth and decreasing unnecessary memory bandwidth consumption, we propose the remote direct cache access (RDCA) architecture to let senders bypass the receiver's memory and access its cache directly. It is supported by an important observation in Pangu's production workload: the timespan data spent in memory after leaving NIC is very short (*i.e.*, hundreds of μs on average). Assuming a 200 μs average post-NIC timespan, for a dual-port 100 Gbps NIC, we only need 5 MB to temporally store the data leaving NIC. Although other cache accessing technologies (*e.g.*, DCA [54, 55] and DDIO [56]) have been proposed, they suffer from the leaky DMA problem (*i.e.*, frequent cache eviction triggered by new arrival messages [57, 58]). In contrast, RDCA goes beyond substantially to show that we can recycle a small area of LLC to support NIC operations at line rate with three components:

Cache-resident buffer pool. The pool uses a shared receiver queue (SRQ) for receiving small messages and a READ buffer equipped with a window-based rate control mechanism for receiving large messages, such that the memory buffer needed for RDMA operations can fit into the cache.

Swift cache recycle. In order to support 100-Gbps NIC operating at line rate with as few LLC as possible, we design the swift cache recycle mechanism to reduce data's post-NIC timespan by (1) processing data in parallel along a pipeline, and (2) optimizing processing using hardware offloading and lightweight (de)serialization.

Cache-pressure-aware escape mechanism. To deal with occasional jitters (*e.g.*, SSD slow write and application exceptions), the escape mechanism monitors the usage of reserved LLC and takes corresponding actions, including (1) replacing the cache buffer of straggler data by adding a new buffer to the cache-resident buffer pool, such that the size of the usable cache in RDCA to accommodate newly arriving requests remains unchanged; (2) actively copying the data of slow-running applications to memory if too many replacements happened, such that other applications can use the RDCA buffer pool and the pool does not take up too much cache; and (3) let the NIC mark explicit congestion notification (ECN) in congestion notification packets to indicate congestion if copying to memory fails or is insufficient in releasing the cache pressure.

We leverage Intel's DDIO [56] to implement RDCA on commodity hardware. Results of extensive evaluation in some clusters of Pangu show that for typical storage workloads, RDCA consumes a 12MB LLC cache (20% of the total cache) per server, decreases the average memory bandwidth consumption by ∼89% and improves network throughput by 9%. We find that RDCA is also effective in non-storage workloads, *e.g.*, it reduces the average latency of collective communications in latency-sensitive HPC applications by up to 35.1%. RDCA is rolled out in Pangu at the end of 2022.

4.3 CPU Bottleneck

Even with optimizations to break the network and memory bottleneck, the throughput of Pangu in a 100-Gbps network can still reach only 80% of its theoretical value. It is because operations such as data serialization and deserialization, data compression and data CRC computation consume many CPU resources, making CPU another bottleneck of Pangu. To this end, Pangu introduces a hybrid design to reduce (de)serialization operations (§4.3.1), a special hardware instruction *CPU Wait* to make full use of the CPU (§4.3.2), and a hardware/software co-design [59] to offload CRC computing and data compression to hardware (§4.3.3).

4.3.1 Hybrid RPCs

Serializing and deserializing RPC requests using Protobuf [60] in Pangu costs about 30% of CPU overhead. We observe that this overhead mostly happens in the data path over a small number of RPC types. As such, we take a hybrid design to handle this issue. We switch our data path operations to use a raw structure similar to FlatBuffer [61], to send and receive data directly without serialization. Pangu continues to use Protobuf for control operations due to its flexibility and complexity. As a result, network throughput for each CPU core increased by about 59%.

4.3.2 Supporting Hyper-Threading Using CPU Wait

Pangu initially did not use hyper-threading (HT) [62], but started to adopt it as the CPU core resources become scarce. However, HT has two main performance issues. First, two HTs on one physical core need to switch contexts. Second, one HT affects the execution of the other HT when they execute tasks simultaneously, resulting in increased latency on both tasks. For example, when a network idle-polling thread is running on one HT, and a compression thread running on the other HT from the same physical core is compressing 4 KB data with the LZ4 algorithm [49] and lzbeach [63] at the same time, the latency of data compression increases by 25%, compared with the case that the compression thread exclusively occupies the physical core.

To solve these two issues, Pangu introduces the *CPU wait* instruction. It consists of *monitor* and *mwait*. Pangu needs less than 5 ms to call them. Revisit the example above. After introducing CPU wait, the network idle-polling thread will *mwait* at the monitored memory address and does not wake up until the memory address is written by other threads. During the *mwait* process, the HT it runs on enters an idle sleep state, one of the C-States except for C0 [64, 65], without interfering with the other HT. In addition, the time of waking up HTs with system calls is of ms-level. As such, Pangu can fully utilize the CPU cores with high performance. In this example, the network throughput increases by 31.6%, compared with the case where *CPU wait* is not used.

4.3.3 Hardware and Software Co-design

For high performance, Pangu offloads some tasks from CPU to programmable hardware. First, data compression is offloaded to FPGA-based computational storage drives [66]. At a 3-GB/s throughput, hardware-based FPGA compression can save about 10 physical cores (Intel(R) Xeon(R) Platinum 2.5 Ghz). Second, CRC computation is offloaded to RDMA-capable NICs, which calculates the CRC of each data block. CPU then aggregates these CRC and performs a lightweight check [18]. This design ensures low CPU overhead and high application-level data integrity. At the same throughput, compared with using software, using hardware for CRC computation can save 30% of CPU overhead.

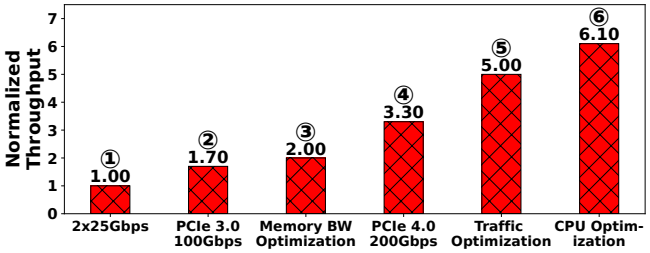

Figure 10: The normalized effective throughput per storage node of Pangu in the evolution process.

4.4 Evaluations

Figure 10 shows the normalized effective throughput per storage node (NETPSN) on the evolution roadmap of the storage node on EBS and high-performance ESSD [67].

Stage 1. To better illustrate the development of the performance of Pangu 2.0, we initialize NETPSN as 1 when Pangu introduces 2 × 25-Gbps network and 4-TB SSDs.

Stage 2. As SSD's capacity increases to 8 TB with higher performance, Pangu introduces a 100-Gbps network. The network bandwidth is 100 Gbps because the server supports PCIe gen3. Though the network bandwidth doubles, NETPSN increases from 1 to 1.7 instead of 2. It is because the memory bandwidth for read and write reaches 90 GB/s, close to the maximal memory bandwidth of the server (105 GB/s).

Stage 3. Insufficient memory bandwidth leads to packet drops at NICs, resulting in a large throughput drop. With the memory bandwidth optimizations in §4.2, we successfully increase NETPSN to 2.

Stage 4. After introducing PCIe gen4, the network throughput increases to 2×100 Gbps. The network bandwidth doubles, but NETPSN does not, *i.e.*, increased to 3.3 rather than 4, due to the traffic amplification problem (about 6x).

Stage 5. After the Pangu client adopts EC(4, 2), data compression, and other optimizations, the traffic amplification ratio reduces by half (∼3x) (§4.1). However, NETPSN increases to 5 instead of 6.6. The main reason is that data compression and other operations consume many CPU resources.

Stage 6. In order to solve the CPU bottleneck, Pangu offloads tasks (*e.g.*, data compression and CRC) from CPU to hardware (§4.3), eventually increasing NETPSN to 6.1.

These results show that by breaking the bottlenecks of network, memory, and CPU, Pangu achieves high performance and adapts to the new performance-oriented business model.

5 Operation Experiences

After introducing the design innovation in Pangu 2.0, we next introduce the basic operation cycle of Pangu, focusing on the Pangu monitoring system, and share our experiences in addressing several important issues during Pangu's operation.

5.1 Pangu's Operation Cycle

Pangu's basic operation cycle consists of five stages: planning, development, testing, deployment, and monitoring. Before entering development, new hardware and software solutions go through a rigorous planning phase (*i.e.*, feasibility analysis, benefit/cost analysis, and social and regulation studies). We conduct extensive tests on the solutions under various scenarios between development and deployment. In particular, Pangu copes with the interoperability issue among heterogeneous hardware from different vendors using an in-house, template-based admission test. New solutions are rolled out in Pangu's production environment cluster by cluster. After they are online, the Pangu monitoring system watches their behaviors closely via fine-grained monitoring, thorough root-cause analysis, fast response, and post-mortem documentation.

Fine-grained monitoring and intelligent diagnosis. In Pangu 2.0, we improve the Pangu monitoring system with two key designs to keep up with the high-performance requirement (*i.e.*, 100 μs-level I/O latency). First, we increase the time granularity of monitoring from 15 seconds to 1 second and extend the Log Service [68] to design an on-demand tracing system. Compared with the coarse-grained monitoring in Pangu 1.0, this allows us to perform tracings on a per-file-operation basis to accurately capture fine-grained abnormal events (*e.g.*, memory allocation exception and log printing timeout). Second, we embrace AI to better capture the causal relationship between abnormal events and their root causes. The inferred root causes are rated by operating teams and fed back to the trained model to improve its accuracy. This design substantially improves diagnostic accuracy and reduces the required human efforts.

5.2 Case Studies

Extensive data integrity checking. Pangu extensively employs CRC to ensure data integrity, such as end-to-end CRC along the data path, monthly CRC on all replicates, CRC on random sampled replicates and CRC on one extra replicate during EC building. Among all the data integrity issues we encountered, the CPU silent error is a representative one. Specifically, we find some end-to-end CRC errors and pinpoint their root causes as the silent errors on certain CPUs. To prevent such errors, we work with Intel to deploy silent error testing tools that run in production environments when the overall workload is low, and achieve some success.

Handling SLA jitters in USSOS. During Pangu's operation, we encounter and address several issues that contribute to SLA jitters in USSOS (§3.2), such as memory allocation, periodic heavyweight tasks and increased USSOS CPU utilization. First, we find existing memory allocation mechanisms (*e.g.*, TCMalloc [29]) become too time-consuming if they enter a global memory allocation phase (*e.g.*, when a new thread requires memory) or a memory organization phase (*e.g.*, when too many RDMA queue pairs try to reserve high-order memory space). To this end, we introduce a user-space memory allocation pool and optimize RDMA drivers to use anonymous pages. Second, for periodic heavyweight tasks (*e.g.*, log printing), we move them to asynchronous threads to avoid affecting the SLA of data operations. Third, we find that the CPU utilization of USSOS significantly increases when the memory occupation is high and USSOS needs to perform memory recollection. As such, we adjust the threshold of memory recollection to reduce the chance of USSOS entering memory recollection. We also recollect memory allocated to the buffer and cache in the background.

Handling correctable machine check exceptions (MCE) in the USSOS to improve availability. Initially, USSOS (§3.2) can monitor such hardware failures, but it cannot perceive how the kernel migrates the physical memory for exception isolation. As such, errors would happen when USSOS tries to access the already migrated physical memory based on its outdated virtual-physical memory address mapping.

To this end, we add a handler to the MCE monitor daemon in USSOS. Once the number of found correctable MCEs exceeds a threshold, the user-space process related to them will pause and let the handler notify the kernel to migrate the memory. After the migration, the process resumes and updates its mapping table before accessing memory pages. This design improves the availability of Pangu with negligible performance degradation. For example, we observe <330 correctable MCEs in a 2300-server cluster in 22 days.

Heterogeneous memory bandwidths from different vendors. Pangu deploys memories from different vendors. Our tests show that under a 1:1 memory read/write ratio, the achievable bandwidths of 128 GB memory from three different vendors are 94 GB/s, 84 GB/s, and 60 GB/s, respectively (*i.e.*, a 57% difference). Such heterogeneous memory bandwidths would cause performance degradation in clusters. This observation taught us to pay more attention to the performance of memory, instead of the capacity. It is also our earliest evidence of congestions in the receiver host datapath and a direct motivation for RDCA (§4.2.3).

Coping with tail latency surge during Double 11 Festival. This festival is Alibaba's largest annual online shopping event. Guaranteeing Pangu's high performance under such high-pressure traffic requires real-time monitoring, diagnosis, and response to ensure all technical features work in harmony. On the Double 11 Festival in 2019, we noticed a surge of read tail latency in the Relational Database Service (RDS) built on top of Pangu. By analyzing the system traces, we identify the root cause as the increased simultaneous occurrence of rebalancing migration of 2 chunks and the failure of chunkserver storing the remaining unmigrated chunk. As such, clients have to try and fail to access all three replicas before requesting the

IP addresses of the latest chunkservers, increasing latency. To fix this issue, we let the clients periodically fetch chunk information of abnormal chunkservers from the masters and immediately request the latest chunk metadata if their needed chunks are on abnormal servers. This action works well in coping with the surge of tail latency back then.

However, this mechanism has its limitation. On the Double 11 Festival 2020, clients experience the surge of tail latency again. We find that the root cause is that many chunkservers are determined as abnormal due to an internal issue. Clients then receive a large amount of information about abnormal chunkservers and spend many resources processing them (*e.g.*, deserialization and address resolution), resulting in long-time I/O hang. Facing the upcoming peak traffic, we temporarily disable this mechanism and upgrade it after the festival with a series of improvements, including independent threads for abnormal server operations and limiting the number of abnormal servers requested each time.

6 Lessons

Lessons on user-space systems. We develop Pangu's chunkserver USSOS (§3.2) to keep up with the high speed of new network and storage technologies. During its development and operation, we learned three lessons. First, user-space systems are simpler to develop and operate. For example, our data shows that bug fixing takes about two months in a kernel-space file system, but a couple of weeks in USSFS (§3.2.3). Developing new features (*e.g.*, zoned namespace [69]) in the user space requires fewer developers and a shorter time. Furthermore, it is also easier to monitor and trace behaviors of user-space systems and adjust their parameters accordingly.

Second, developing user-space systems should learn from the design of the kernel space. In particular, to build a high-performance USSOS, not only we need to unify the storage and network stack, we also need to design user-space modules for memory management, CPU scheduling and hardware failure handling. The kernel space is pretty good at these functionalities. As such, user-space systems can benefit by learning from it.

Third, the performance gain of user-space systems is not exclusive to high-speed storage such as SSD. Specifically, we provide a series of mechanisms in Pangu's USSFS to accelerate the performance of HDD. For example, USSFS takes advantage of the self-contained chunk layout (§3.1.3) to save the number of metadata operations. It leverages the differences between internal and external tracks of disks to improve HDD's write efficiency.

Lesson on performance-cost tradeoff. To meet new business requirements, Pangu usually first chooses to add more hardware to improve its performance based on total cost of ownership (TCO) balance (*e.g.*, upgrading the network from 25 Gbps to 100 Gbps, increasing the number of memory channels by placing more small-volume DRAMs and upgrading servers with more powerful CPUs). Hardware expansion effectively improves Pangu's performance, but is not sustainable due to the cost incurred. As such, Pangu also spends substantial efforts, such as traffic optimization (§4.1.2), improving its resource utilization and efficiency.

Lesson on persistent memory (PMem). PMem has many advantages, such as fast data persisting, RDMA friendliness, low read latency (6 μs on PMem vs. 80 μs on SSD), low tail latency, and cache friendliness. As such, we developed a 30-μs PMem-based EBS service [1] in Pangu. However, Intel's decision to kill off its PMem business [70] forces us to rethink this service. We need to plan more thoroughly when developing new services (*e.g.*, considering substitutability, sustainability, and cost tradeoffs). But we are optimistic that new storage class memory [71] will emerge with better solutions to these issues.

Lesson on hardware offloading. The cost vs. benefits tradeoff is a fundamental issue for hardware offloading (§4.3.3), and has been an ever-lasting debate topic in Pangu since 2018. The entire development of hardware offloading compression takes a 20-person team two years, during which we resolve many issues such as the FPGA hardware cost, the integrity of compressed data, and the co-existence with other functions in hardware. In the end, the outcome benefits outweigh this cost substantially. Hardware offloading significantly reduces the compression's average and tail latency, effectively reducing the network traffic within a low latency. As a result, we can improve the service provision capability of our infrastructure by $\sim$50%. We rolled out hardware compression in Pangu in 2020 at a slow pace, first in internal services (*e.g.*, log/monitoring services) and then gradually expanding to core external services (*e.g.*, EBS). To prevent potential bugs in hardware from harming data integrity, we perform data decompression and CRC checking on hardware and conduct routine spot software CRC checking. Since 2022, all 200 Gbps clusters in Pangu enable hardware compression by default, and incidents happen less and less often.

7 Conclusion

We introduce how we embrace the emerging hardware technologies and adapt to the shift of business model to evolve the Pangu to provide high-performance, reliable storage services with a 100μs-level I/O latency. We also share our experiences operating Pangu 2.0 to shed light on future research in large-scale, high-performance storage systems.

Acknowledgments. We are extremely grateful for our shepherd, Peter Macko, and the anonymous FAST'23 reviewers for their wonderful feedback. Qiao Xiang, Haohao Song, Yuxin Wang, and Ridi Wen are supported in part by the National Key R&D Program of China 2022YFB2901502, Alibaba Innovative Research Award, NSFC Award 62172345, Open Research Projects of Zhejiang Lab 2022QA0AB05, MOE China Award 2021FNA02008, NSF-Fujian-China 2022J01004, and IKKEM Award HRTP-2022-34.

References

[1] Alibaba Group. Alibaba Cloud Products & EBS. https://www.aliyun.com/product/disk. Accessed Sept 18, 2022.

[2] Alibaba Group. Alibaba Cloud Products & OSS Services. https://www.alibabacloud.com/zh/product/object-storage-service. Accessed Sept 18, 2022.

[3] Alibaba Group. Alibaba Cloud Products & NAS Services. https://www.alibabacloud.com/zh/product/nas. Accessed Sept 18, 2022.

[4] Alibaba Group. Alibaba Cloud Products & PolarDB. https://www.alibabacloud.com/zh/product/polardb. Accessed Sept 18, 2022.

[5] Alibaba Group. Alibaba Cloud Products & MaxCompute. https://www.alibabacloud.com/zh/product/maxcompute. Accessed Sept 18, 2022.

[6] Avantika Mathur, Mingming Cao, Suparna Bhattacharya, Andreas Dilger, Alex Tomas, and Laurent Vivier. The New Ext4 Filesystem: Current Status and Future Plans. In *Proceedings of the Linux symposium*, pages 21–33. Citeseer, 2007.

[7] Wright Stevens. TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms. Technical report, 1997.

[8] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A Scalable, Commodity Data Center Network Architecture. In *SIGCOMM'08*, pages 63–74. ACM, 2008.

[9] Alibaba Group. Alibaba Cloud Products & OTS. https://www.alibabacloud.com/zh/product/table-store. Accessed Sept 18, 2022.

[10] Satadru Pan, Theano Stavrinos, Yunqiao Zhang, Atul Sikaria, Pavel Zakharov, Abhinav Sharma, Shiva Shankar P., Mike Shuey, Richard Wareing, Monika Gangapuram, Guanglei Cao, Christian Preseau, Pratap Singh, Kestutis Patiejunas, J. R. Tipton, Ethan Katz-Bassett, and Wyatt Lloyd. Facebook's Tectonic Filesystem: Efficiency from Exascale. In *FAST'21*, pages 217–231. USENIX Association, 2021.

[11] Colossus under the Hood: A Peek into Google's Scalable Storage System. https://cloud.google.com/blog/products/storage-data-transfer/a-peek-behind-colossus-googles-file-system. Accessed Sept 18, 2022.

[12] Diego Ongaro and John Ousterhout. In Search of an Understandable Consensus Algorithm. In *ATC'14*, pages 305–319. USENIX Association, 2014.

[13] Wenhao Lv, Youyou Lu, Yiming Zhang, Peile Duan, and Jiwu Shu. InfiniFS: An Efficient Metadata Service for Large-Scale Distributed Filesystems. In *FAST'22*, pages 313–328. USENIX Association, 2022.

[14] Su Zhou, Erci Xu, Hao Wu, Yu Du, Jiacheng Cui, Wanyu Fu, Chang Liu, Yingni Wang, Wenbo Wang, Shouqu Sun, Xianfei Wang, Bo Feng, Biyun Zhu, Xin Tong, Weikang Kong, Linyan Liu, Zhongjie Wu, Jinbo Wu, Qingchao Luo, and Jiesheng Wu. Deployed System: SMRSTORE: A Storage Engine for Cloud Object Storage on HM-SMR Drives. In *FAST'23*. USENIX Association, 2023.

[15] Qiang Li, Lulu Chen, Xiaoliang Wang, Shuo Huang, Qiao Xiang, Yuanyuan Dong, Wenhui Yao, Minfei Huang, Puyuan Yang, Shanyang Liu, et al. Fisc: A Large-Scale Cloud-Native-Oriented File System. In *FAST'23*. USENIX Association, 2023.

[16] Ruiming Lu, Erci Xu, Yiming Zhang, Fengyi Zhu, Zhaosheng Zhu, Mengtian Wang, Zongpeng Zhu, Guangtao Xue, Jiwu Shu, Minglu Li, and Jiesheng Wu. Deployed System: Perseus: A Fail-Slow Detection Framework for Cloud Storage Systems. In *FAST'23*. USENIX Association, 2023.

[17] Yixiao Gao, Qiang Li, Lingbo Tang, Yongqing Xi, Pengcheng Zhang, Wenwen Peng, Bo Li, Yaohui Wu, Shaozong Liu, Lei Yan, et al. When Cloud Storage Meets RDMA. In *NSDI'21*, pages 519–533. USENIX Association, 2021.

[18] 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, and Hongqiang Harry Liu. From Luna to Solar: the Evolutions of the Compute-to-Storage Networks in Alibaba Cloud. In *SIGCOMM'22*, pages 753–766. ACM, 2022.

[19] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google File System. In *SOSP'03*, pages 29–43. ACM, 2003.

[20] Dhruba Borthakur. HDFS Architecture Guide . https://hadoop.apache.org/docs/r1.2.1/hdfs_design.htmll. Accessed Sept 15, 2022.

[21] Microsoft. Azure Storage. https://azure.microsoft.com/en-us/products/category/storage/. Accessed Sept 18, 2022.

[22] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long, and Carlos Maltzahn. Ceph: A Scalable, High-Performance Distributed File System. In *OSDI'06*, pages 307–320. USENIX Association, 2006.

[23] AWS. Cloud Storage on AWS. https://aws.amazon.com/products/storage/. Accessed Sept 18, 2022.

[24] Zhu Pang, Qingda Lu, Shuo Chen, Rui Wang, Yikang Xu, and Jiesheng Wu. ArkDB: A Key-Value Engine for Scalable Cloud Storage Services. In *SIGMOD'21*, pages 2570–2583. ACM, 2021.

[25] Chromium Contributor. Jim Roskind. QUIC: Design Document and Specification Rationale. https://docs.google.com/document/d/1RNHkx_VvKWyWg6Lr8SZ-saqsQx7rFV-ev2jRFUoVD34/edit. Accessed Jan 3, 2023.

[26] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, et al. The QUIC Transport Protocol: Design and Internet-Scale Deployment. In *SIGCOMM'17*, pages 183–196. ACM, 2017.

[27] Intel. Data Plane Development Kit. https://dpdk.org/. Accessed Aug 25, 2022.

[28] SPDK. Storage Performance Development Kit. https://www.spdk.io/. Accessed Aug 25, 2022.

[29] Sanjay Ghemawat and Paul Menage. TCMalloc: Thread-Caching Malloc. http://goog-perftools.sourceforge.net/doc/tcmalloc.html. Accessed Aug 25, 2022.

[30] J. Yang, D. B. Minturn, and F. Hady. When Poll Is Better than Interrupt. In *FAST'12*, pages 1–7. USENIX Association, 2012.

[31] Luiz Barroso, Mike Marty, David Patterson, and Parthasarathy Ranganathan. Attack of the Killer Microseconds. *Communications of the ACM*, 60(4):48–54, 2017.

[32] Backblaze. The SSD Edition: 2022 Drive Stats Mid-year Review. https://www.backblaze.com/blog/ssd-drive-stats-mid-2022-review/. Accessed Sept 18, 2022.

[33] Iyswarya Narayanan, Di Wang, Myeongjae Jeon, Bikash Sharma, Laura Caulfield, Anand Sivasubramaniam, Ben Cutler, Jie Liu, Badriddine Khessib, and Kushagra Vaid. SSD Failures in Datacenters: What? When? and Why? In *SYSTOR'16*, pages 1–11. ACM, 2016.

[34] Mingzhe Hao, Gokul Soundararajan, Deepak Kenchammana-Hosekote, Andrew A Chien, and Haryadi S Gunawi. The Tail at Store: A Revelation from Millions of Hours of Disk and SSD Deployments. In *FAST'16*, pages 263–276. USENIX Association, 2016.

[35] Bianca Schroeder, Raghav Lagisetty, and Arif Merchant. Flash Reliability in Production: The Expected and the Unexpected. In *FAST'16*, pages 67–80. USENIX Association, 2016.

[36] Erci Xu, Mai Zheng, Feng Qin, Yikang Xu, and Jiesheng Wu. Lessons and Actions: What We Learned from 10K SSD-Related Storage System Failures. In *ATC'19*, pages 961–976. USENIX Association, 2019.

[37] Mai Zheng, Joseph Tucek, Feng Qin, and Mark Lillibridge. Understanding the Robustness of SSDs under Power Fault. In *FAST'13*, pages 271–284. USENIX Association, 2013.

[38] Alibaba Group. Alibaba Cloud Products & ECS. https://www.aliyun.com/product/ecs. Accessed Sept 18, 2022.

[39] Amazon. Amazon EC2. https://aws.amazon.com/cn/ec2/?nc2=h_ql_prod_fs_ec2. Accessed Sept 18, 2022.

[40] K. V. Rashmi, M. Chowdhury, J. Kosaian, I. Stoica, and K. Ramchandran. EC-Cache: Load-Balanced, Low-Latency Cluster Caching with Online Erasure Coding. In *OSDI'16*, pages 401–417. USENIX Association, 2016.

[41] Takayuki Fukatani, Hieu Hanh Le, and Haruo Yokota. Delayed Parity Update for Bridging the Gap between Replication and Erasure Coding in Server-Based Storage. In *ADMS@ VLDB*, pages 1–9, 2021.

[42] Youngmoon Lee, Hasan Al Maruf, Mosharaf Chowdhury, Asaf Cidon, and Kang G Shin. Hydra: Resilient and Highly Available Remote Memory. In *FAST'22*, pages 181–198. USENIX Association, 2022.

[43] QSFP. https://community.fs.com/search?key_word=QSFP. Accessed Sept 18, 2022.

[44] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, and Marina Lipshteyn. RDMA over Commodity Ethernet at Scale. In *SIGCOMM'16*, pages 202–215. ACM, 2016.

[45] Mark J. Karol, S. Jamaloddin Golestani, and David Lee. Prevention of Deadlocks and Livelocks in Lossless Backpressured Packet Networks. *IEEE/ACM Trans. Netw.*, 11(6):923–934, 2003.

[46] Brent E. Stephens, Alan L. Cox, Ankit Singla, John B. Carter, Colin Dixon, and Wes Felter. Practical DCB for Improved Data Center Networks. In *INFOCOM'14*, pages 1824–1832. IEEE, 2014.

[47] Intel(R) Intelligent Storage Acceleration Library. https://github.com/intel/isa-l. Accessed Sept 18, 2022.

[48] Jerasure: A Library in C Facilitating Erasure Coding for Storage. https://jerasure.org. Accessed Sept 18, 2022.

[49] LZ4 - Extremely fast compression. https://github.com/lz4/lz4. Accessed Sept 18, 2022.

[50] Saksham Agarwal, Rachit Agarwal, Behnam Montazeri, Masoud Moshref, Khaled Elmeleegy, Luigi Rizzo, Marc Asher de Kruijf, Gautam Kumar, Sylvia Ratnasamy, David Culler, et al. Understanding Host Interconnect Congestion. In *HotNets'22*, pages 198–204. ACM, 2022.

[51] Intel. Intel Stratix 10 FPGAs & SoC FPGA. https://www.intel.com/content/www/us/en/products/details/fpga/stratix/10.html. Accessed Sept 18, 2022.

[52] Bert Hubert et al. Linux Advanced Routing & Traffic Control HOWTO. *Netherlabs BV*, 1:99–107, 2002.

[53] Alok Kumar, Sushant Jain, Uday Naik, Nikhil Kasinadhuni, Enrique Cauich Zermeno, C. Stephen Gunn, Jing Ai, Björn Carlin, Mihai Amarandei-Stavila, Mathieu Robin, Aspi Siganporia, Stephen Stuart, and Amin Vahdat. BwE: Flexible, Hierarchical Bandwidth Allocation for WAN Distributed Computing. In *SIGCOMM'15*, pages 1–14. ACM, 2015.

[54] Ram Huggahalli, Ravi Iyer, and Scott Tetrick. Direct Cache Access for High Bandwidth Network I/O. In *ISCA'05*, pages 50–59. IEEE, 2005.

[55] Amit Kumar, Ram Huggahalli, and Srihari Makineni. Characterization of Direct Cache Access on Multi-Core Systems and 10GbE. In *HPCA'09*, pages 341–352. IEEE, 2009.

[56] Intel® Data Direct I/O Technology (Intel® DDIO): A Primer. https://www.intel.com/content/www/us/en/io/data-direct-i-o-technology.html Accessed Aug 28, 2022.

[57] Amin Tootoonchian, Aurojit Panda, Chang Lan, Melvin Walls, Katerina Argyraki, Sylvia Ratnasamy, and Scott Shenker. ResQ: Enabling SLOs in Network Function Virtualization. In *NSDI'18*, pages 283–297. USENIX Association, 2018.

[58] Amy Ousterhout, Joshua Fried, Jonathan Behrens, Adam Belay, and Hari Balakrishnan. Shenango: Achieving High CPU Efficiency for Latency-Sensitive Datacenter Workloads. In *NSDI'19*, pages 361–378. USENIX Association, 2019.

[59] Giovanni De Michell and Rajesh K Gupta. Hardware/Software Co-Design. *Proceedings of the IEEE*, 85(3):349–365, 1997.

[60] Protocol Buffers are a Language-Neutral, Platform-Neutral Extensible Mechanism for Serializing Structured Data. https://developers.google.com/protocol-buffers/. Accessed Sept 18, 2022.

[61] FlatBuffers is an Efficient cross Platform Serialization Library. https://google.github.io/flatbuffers/. Accessed Sept 18, 2022.

[62] William Magro, Paul Petersen, and Sanjiv Shah. Hyper-Threading Technology: Impact on Compute-Intensive Workloads. *Intel Technology Journal*, 6(1):1–9, 2002.

[63] Lzbench, an in-Memory Benchmark of Various Compressors. https://openbenchmarking.org/test/pts/lzbench. Accessed Sept 18, 2022.

[64] Intel. Intel C-states. https://www.intel.com/content/www/us/en/develop/documentation/vtune-help/top/reference/energy-analysis-metrics-reference/c-state.html. Accessed Sept 18, 2022.

[65] Intel. C-states. https://www.thomas-krenn.com/en/wiki/Processor_P-states_and_C-states. Accessed Sept 18, 2022.

[66] Wei Cao, Yang Liu, Zhushi Cheng, Ning Zheng, Wei Li, Wenjie Wu, Linqiang Ouyang, Peng Wang, Yijing Wang, Ray Kuan, et al. POLARDB Meets Computational Storage: Efficiently Support Analytical Workloads in Cloud-Native Relational Database. In *FAST'20*, pages 29–41. USENIX Association, 2020.

[67] Alibaba Group. Alibaba Cloud Products & ESSD. https://www.aliyun.com/storage/storage/essd/. Accessed Sept 18, 2022.

[68] Alibaba Group. Alibaba Cloud Products & SLS. https://www.aliyun.com/product/sls. Accessed Sept 18, 2022.

[69] Matias Bjørling. From Open-Channel SSDs to Zoned Namespaces. In *Proc. Linux Storage Filesyst. Conf.(Vault)*, pages 1–18, 2019.

[70] Intel. Intel Reports Second-Quarter 2022 Financial Results. https://download.intel.com/newsroom/2022/corporate/Intel-CEO-CFO-2Q22-earnings-statements.pdf. Accessed Sept 18, 2022.

[71] Chung H Lam. Storage Class Memory. In *ICSICT'10*, pages 1080–1083. IEEE, 2010.