Introduction of Baidu Chitu Storage with SPDK NVMe-oF

Abstract: In the deep learning scenario, video and graphics are trained by the GPU which need to randomly read large numbers of small data blocks with high concurrency. Under the common HDFS storage architecture, small files are handled with low performance and also led to low storage utilization. High-speed GPU computing components should be matched with high-speed NVMe[1] storage, but under the HDFS storage architecture, it is difficult to unleash the full capability of the NVMe storage performance.

Baidu Chitu storage provides a shared NVMe storage with high throughput and low latency for the GPU clusters, using a layered architecture. The upper layer is the parallel file system called Baidu Parallel FS (BPFS) using logical volumes, while the lower layer is the NVMe-oF based distributed block layer with high availability. The small file and its metadata information is packaged and stored in the logical volume, and the client's small file access is directly connected to the remote storage server via NVMe-oF.

SPDK NVMe-oF[2] is a key module of the data path. Using SPDK NVMe-oF decreases 8x CPU overhead over the Linux kernel module under a million IOPS pressure on the storage server. The development, operation and

maintenance efficiency of the SPDK user space program [3] is much better than that of the kernel module.

Overall, Baidu Chitu storage's average latency is only 10us higher than the local NVMe, and the multi-threaded random IOs from a single client's for 16K small file could saturate the 100GbE network. A single client can achieve 500K IOPS for small files. Before reaching the hardware bottleneck, multiple servers' aggregated throughput is increased linearly, while at the same time, the aggregated throughput of the multiple clients' random read is also increased linearly and the latency is consistent and not changed. The test performance of the video and graphics training service is equivalent to the RAID0 composed by local multiple NVMes.

Contents

1. Introduction of Baidu Chitu Storage Project

Baidu Chitu storage is a shared NVMe storage pool with hardware and software combined. The goal is to match the company's internal high-performance AI computing service with low latency and high throughput parallel shared storage services, improve the efficiency of GPU and NVMe, and shorten the AI training time.

Baidu has its own self-developed internal HDFS alike storage. Its research and development background is for large files, mechanical disks, and gigabit networks. Thus, it cannot exploit the performance of RDMA and NVMe very well. It has limitations in both throughput and latency. RPC[4] based asynchronous IO performance ranges from 100,000 ~ 200,000 QPS(Query Per Second), while consuming more CPU resource. The poor performance of asynchronous IO determines that the total throughput of the storage server cannot reach a higher level. It is more difficult to optimize the latency than the throughput. In the AI training scenario, there are lots of random reads so that the data cache hit rate is very low, and the IO latency depends on the efficiency of the underlying storage system. The user training program usually adopts a synchronous read and write interface, and the IO latency is directly dominated as the time of the IO phase in the upper application logic processing. On the critical IO path, application, RPC, IO engine, file system, disk process scheduling, memory copy, interrupt processing and so on, all of them increase the IO latency.

The starting point of the Baidu Chitu storage is to decouple the computing and storage with an extremely efficient fabric. The information flow of the entire storage system is divided into three

categories: control flow, metadata flow, and data flow. RPC is more suitable for control flow with variable interfaces. Metadata flow and data flow based on RPC mainly use two data interfaces of reading and writing, which can be only adapted to a simpler and more efficient storage transfer protocols. The earliest investigations included SCSI switch fabric, PCIe switch fabric and RDMA[5]. RDMA is superior in terms of scalability, robustness, ease of use, and fault tolerance. With the large scale application of low latency 100GbE RDMA networks, hardware overhead on the data path is low, while software overhead becomes a system performance bottleneck. First is the latency overhead: a system context switch will double the overall IO latency; second is the computational overhead: the CPU consumption is extremely high under the one million IOPS of high throughput. The NVMe over RDMA performance test performed well in terms of latency, throughput, CPU utilization, and acceleration technology. Therefore, we chose the NVMe-oF protocol as the data transfer protocol. Since NVMe-oF is a standard protocol, we have an additional advantage that we can use industry's optimization and acceleration achievements. In Baidu Chitu storage, we adopted SPDK's NVMe-oF implementation, which has obtained good returns from both IO latency and CPU overhead. The SPDK user space implementation is more conducive to improve engineering development efficiency than the kernel implementation.

2. Overview of SPDK NVMe-oF Target

Since the release of the first Patch[6] related to SPDK NVMe-oF in June 2016, SPDK has continued to develop, optimize and enhance NVMe-oF for more than three years. As a user space implementation, SPDK NVMe-oF has always:

- ✓ Follow the NVMe over Fabrics specifications
- ✓ Provided both user space NVMe-oF Host and Target as a total solution
- ✓ Been compatible with kernel mode NVMe-oF Host and Target
- ✓ Followed application framework based on SPDK optimization[7]
- ✓ Followed high performance, linear expansion, ease of development, ease of maintenance, user space, etc.

Following is the development history of SPDK NVMe-oF Target and Host[8]:

Development History of SPDK NVMe-oF Target:

Development History of SPDK NVMe-oF Host:

In this part we mainly focus on SPDK NVMe-oF Target[9]. NVMe-oF Target is an extension of the NVMe protocol over different transports. The NVMe protocol has established a specification for high speed access to PCIe SSDs. Compared with SATA, SAS, AHCI and other protocols, the NVMe protocol has great advantages in terms of bandwidth, latency, IOPS, etc. Understandably, the price is relatively higher than normal mechanical disk. However, it is undeniable that servers configured with PCIe SSDs have showed up in various application scenarios and have become an industry trend. In order to expose the advantages of local high speed access to remote applications, the NVMe-oF protocol was created. The transport in the NVMe-oF protocol can be varied, such as Ethernet, Fibre Channel, Infiniband, etc. The current popular transport implementation is based on RDMA's Ethernet transport, which supports both Linux kernel and SPDK's NVMe-oF Target. In addition, for FC transport, Netapp has developed and contributed to SPDK NVMe-oF target for the FC based transport[10].

The SPDK's NVMe-oF Target implementation is earlier than the official release of the Linux kernel NVMe-oF Target. Of course, when the new Linux distribution comes with NVMe-oF Target, everyone will have a question, why should we use the SPDK's NVMe-oF Target. In general, the SPDK's NVMe-oF Target has an absolute advantage from the single CPU core performance(Performance/Per CPU core). Detailed performance report is linked to the document[11] in the last section. This advantage comes from the following aspects:

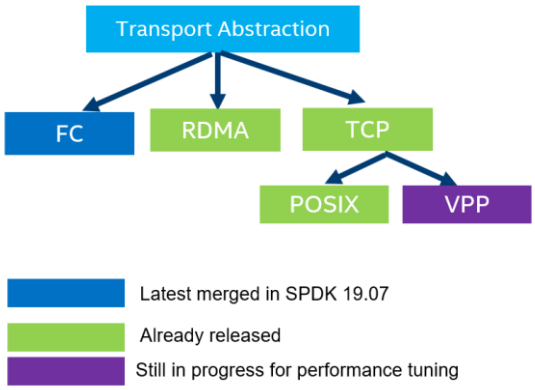
- **SPDK NVMe-oF Target can directly use the SPDK NVMe user-mode driver to package the Bdev.** It is more advantageous than the kernel NVMe driver used by the kernel.
- **Use of the SPDK application programming framework.** The SPDK NVMe-oF Target fully uses the programming framework[12] provided by the SPDK, and uses a lock-free mechanism on all I/O paths, which greatly improves performance.

- **Efficient use of RDMA Ethernet transport.** SPDK's current implementation of RDMA transport uses standard RDMA programming libraries (such as libibverbs), but incorporates the SPDK programming framework. Currently, each reactor running on the CPU core assigned to the SPDK operates a group poller. This poller can handle all the connections handled by that CPU core. These connections share one same RDMA completion queue. Therefore, in the case of multiple concurrent connections, the latency of I/O processing can be greatly reduced.

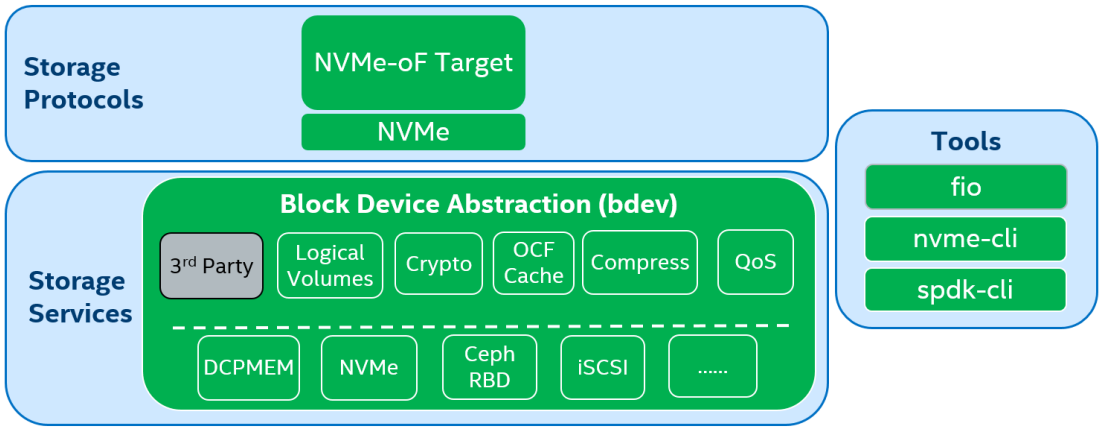
In the SPDK 19.01 release[13], we implemented the NVMe over TCP transport implementation in the user space for the SPDK NVMe-oF Target at the earlies stage when the specification is published. In general, the entire design of the SPDK NVMe over TCP transport follows the SPDK's concept of lock-free, polling and asynchronous I/O. It is a good support for existing non RDMA networks. The core design concept of SPDK NVMe over TCP transport is as follows:

Ingredients	Methodology
Design framework	Follow the general SPDK NVMe-oF framework (e.g., polling group)
TCP connection optimization	Use the SPDK encapsulated Socket API (preparing for integrating other stack, e.g., VPP)
NVMe/TCP PDU handling	Use state machine to track
NVMe/TCP request life time cycle	Use state machine to track (Purpose: Easy to debug and good for further performance improvement)

The following picture is a general overview of SPDK NVMe-oF transports. SPDK NVMe-oF software solution, combined with Intel hardware platform (CPU, network card, NVMe SSD, non-volatile memory[14] and so on) through support for different transport (RDMA, TCP and FC), can be used to unlock the performance advantages of the hardware platform.

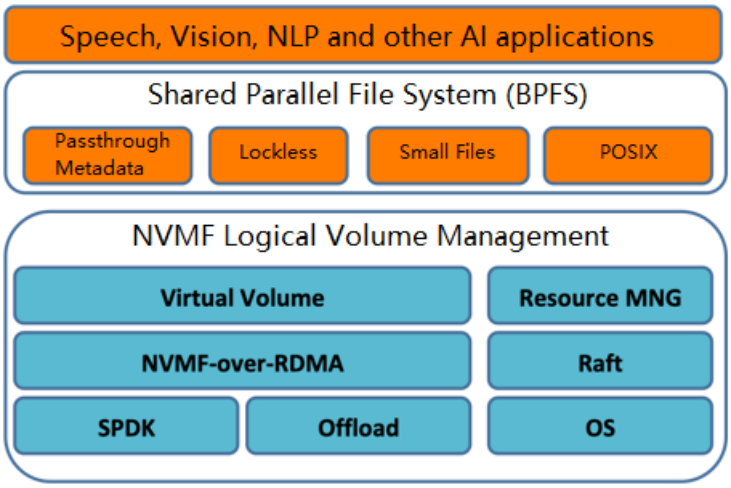


In addition to the ultimate performance benefits, the SPDK NVMe-oF software solution brings more software features such as logical volume management, end-to-end protection, compression, encryption, flow control, user space's easy development, maintain and upgrade and more, as shown in the following figure.



These extra software features make it easier to reference and integrate into specific application scenarios like Baidu's system and drive project development and progress as quickly as possible. In this chapter, we have introduced the SPDK NVMe-oF solution in general. In the next chapter, we will introduce how Baidu Chitu storage integrates the SPDK NVMe-oF solution.

3. How Baidu Chitu Storage Integrate SPDK NVMe-oF



The Baidu Chitu storage is divided into 2 layers: the lower layer is the NVMe-oF (NVMF) logical volume; the upper layer is the Baidu Parallel File System Layer(BPFS). The GPU cluster can access the backend shared NVMe in parallel through BPFS, resulting in a large number of low latency, high throughput random IO reads. The following introduces the IO path and control path respectively.

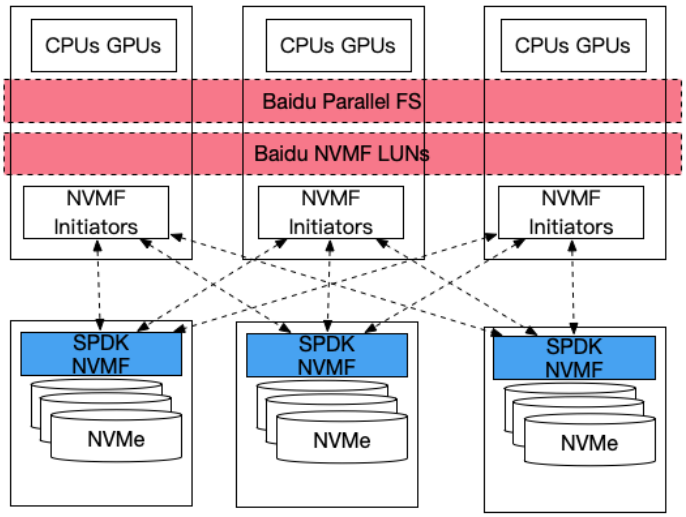
3.1 Brief Introduction to BPFS

Reducing metadata overhead is the key to improve the random access throughput of small blocks of data. Two technologies are mainly used in the industry: 1. Use more computing resource like by consistent hash; 2. Divide and conquer: metadata is distributed to multiple Masters, or Clients, or Nodes. These two technologies will be both used in the super large system at the same time. BPFS runs on the client side and is responsible for metadata processing of small files. The storage of metadata is delegated to the lower level NVMe-oF logical

volume, which is forwarded to multiple backend nodes through NVMe-oF. Compared with the optimization of small files by the architecture such as HDFS, NVMe-OF logical volume is equivalent to a very large file visible to the user, and BPFS is responsible for the data organization inside the very large file. The sharing of BPFS by multiple clients requires the handling of concurrent read and write consistency. The upper layer of BPFS is mainly based on parallel random reads. To achieve efficient concurrent read performance, BPFS uses single write and time lease to ensure write exclusivity without the use of locks.

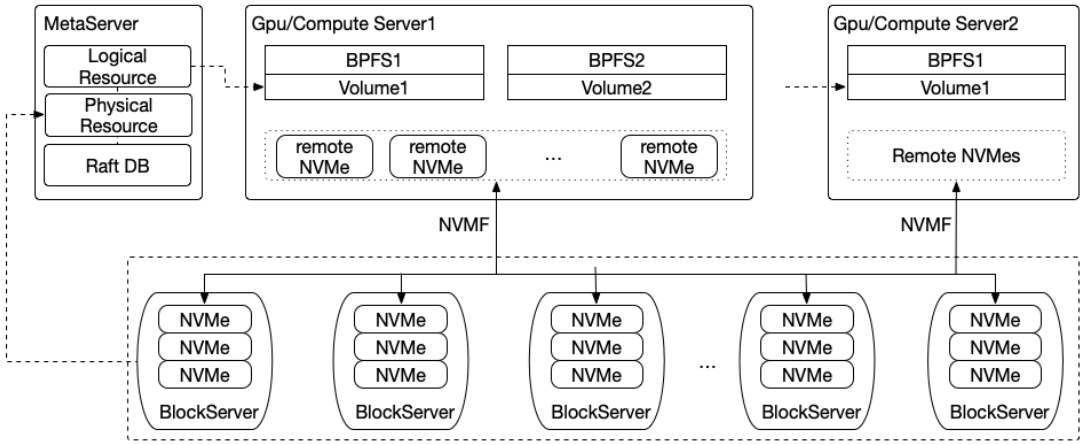
3.2 IO Path

The IO path of Baidu Chitu storage is built on NVMe-oF. Unlike HDFS, which concentrates metadata on the Master server's all flash storage, BPFS runs on the client server. The FS data and metadata are stored in the same NVMe-oF logical volume, and are loaded into the local cache as needed. NVMe-oF logical volume also runs on the client server, including routing information for the IO, information of the copies and striping IO. The read request randomly requests a copy from the client server to the storage server via NVMe-oF. Write requests are forwarded to the write node at the backend, then distributed from that write node to all copies, and the copy consistency is maintained. Any set of copies has only one unique write node. BPFS uses a single write point, so these write nodes for copies are on the BPFS write node. The data stream passes from the NVMe-oF Initiator through the RDMA network to the single node storage engine on the storage side. As the core module of the storage side, the SPDK processes the read and write requests of the NVMe-oF protocol and then forwards them to the local NVMe.



3.3 Control Path

The control path of Baidu Chitu storage uses the common GRPC and works on the logical volume. It manages the allocation and collection of the Chunk resources, the online and offline of the nodes, the online and offline of the NVMe disk, the creation, deletion, capacity expansion, data recovery, data migration, copy change write node of NVMe-oF logical volume and so on. The operation of the control path is extremely low frequency compared to the IO path. As mentioned above, the NVMe-oF logical volume is similar to the large file on the HDFS. The logical volume layer uses the micro service architecture. The control information is stored in multiple masters by using RAFT. The storage node provides the NVMe-oF Target externally.



The stability of the storage service is mainly determined by the stability of the SPDK NVMe-oF. The single node engine module manages the SPDK through JSON-RPC[15]. Outside the SPDK, local NVMe-oF Target periodically receives a sample check IO to check the NVMe-oF path, detect hardware and software failures in time, and report the inspection results to the Master service through heartbeat. Specific check method: Use the SPDK NVMe-oF user space Initiator to send a small amount of 4K random read to each NVMe-oF Target. When a NVMe fails to read or times out, the management module first tries to reset the SPDK from top to bottom, Target, Bdev, Driver, until the entire SPDK service is reset. Logical volumes provide availability with multiple copies, and can tolerate that a copy of the minute level is temporarily unavailable, and instead to use other surviving copies. Thanks to the user space implementation of the SPDK NVMe-oF, service resets can be performed without worrying about downtime, minimizing service unavailability and invalid data recovery caused by software bugs.

4. Baidu Chitu Storage Performance Comparison

4.1 SPDK NVMe-oF Target and kernel NVMe-oF Target Comparison

The main hardware configuration is as following:

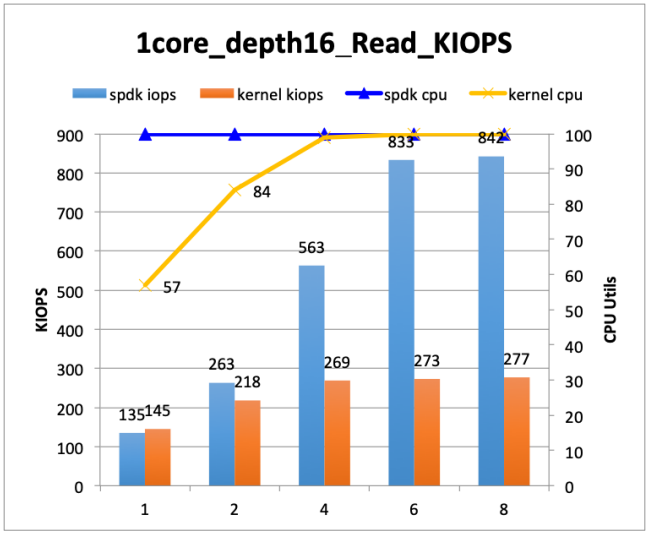
CPU	Intel(R) Xeon(R) Silver 4110[16] CPU * 2
DRAM	128GB
NVMe SSD	4TB Intel(R) P4500[17] NVMe * 8

***Equipped with Intel Enterprise P4500 NVMe SSD, combined with SPDK NVMe-oF Target software solution on Intel CPU platform, to maximize the performance advantages.**

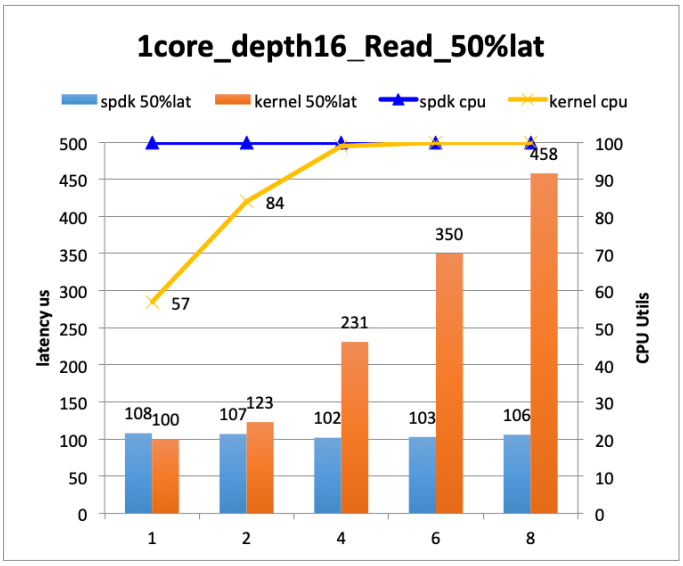
These tests are conducted by the commonly used tool FIO[18]. The main configuration is as following:

IoDepth=16; Job=1; Core=1; BlockSize=4K

*At this time, the NVMe disk does not reach the bottleneck, reflecting the CPU processing bottleneck.

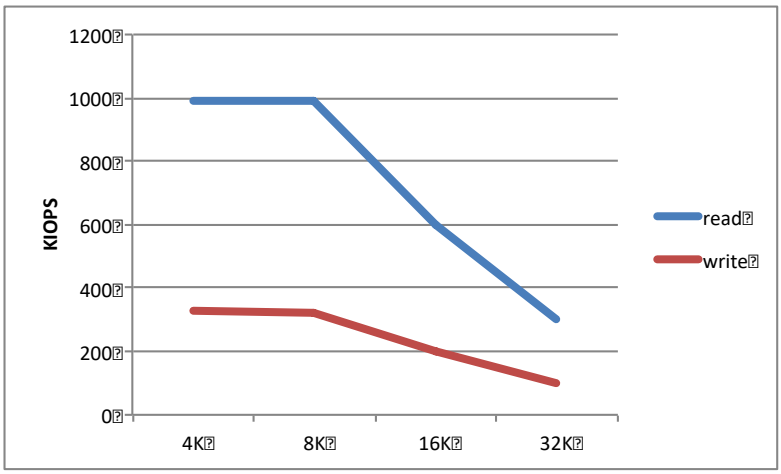


The QQS on single CPU core from SPDK is over 3 times that of Kernel.

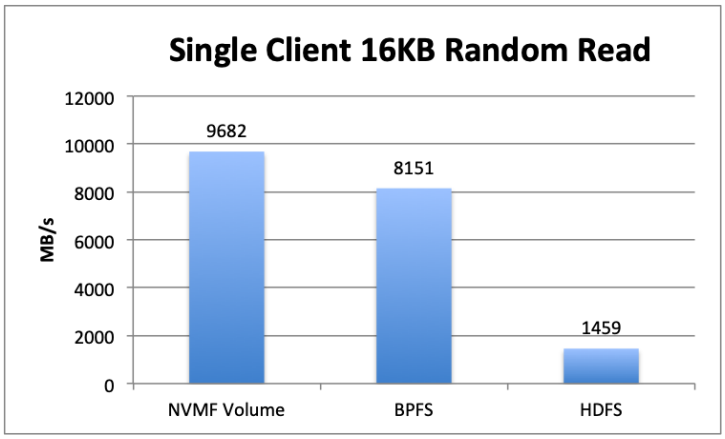


At the same time, the 50% percentile latency from SPDK is only 23% of the kernel, and the SPDK NVMe-oF latency is close to the latency of the NVMe disk itself.

4.2 Throughput of SPDK NVMe-oF Target



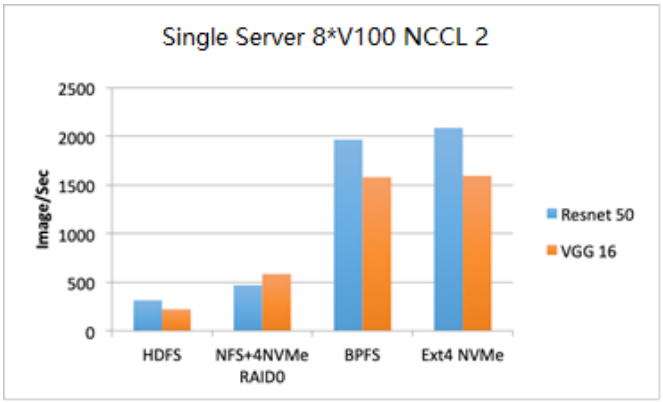
The IOPS capability of a single client NVMe-oF volume is 1,000,000, and the 16KB random IO can saturate the 100GbE network. Write throughput is relating to 3 copies' distribution. 16 threads and 16 asynchronous IO per thread.



In the 16KB random read scenario, BPFS single client throughput is much better than Baidu's internal HDFS system.

4.3 Baidu's Business Performance with SPDK NVMe-oF

Training model ResNet-50[19] and VGG-16[20], GPU cluster, average picture size 16KB, Torch 8 thread, single training server throughput Image/Sec.



The IO latency is the core factor. The Baidu Chitu BPFS's latency is close to the local combination of Ext4 and NVMe, and the 8-card training performance is close to the local NVMe storage.

5. Future Collaborations[21]

1. **Continuously improve the stability of the SPDK module:**

   The SPDK code adds type checking. Void * weak type causes the running logic to not match the actual type. We currently use magic code for runtime type checking, skipping mismatched running logic.

2. **SPDK support for new features in the NVMe specification:**

   SPDK adds more media and custom support. OCSSD, multiple streams NVMe, etc.

3. **Consider to introduce SPDK NVMe over TCP to support existing network resources:**

   SPDK NVMe over TCP: cross node RDMA requires additional asset and technology investment, and NVMF over TCP provides a low cost and cross node NVMF implementation.

4. **SPDK's additional Support for new features in the specification:**

   The SPDK supports customization and processing of NVMF SQE reserved areas. For example, support for Seqnum, self describing and other metadata.

6. Notes and Links

1. https://en.wikipedia.org/wiki/NVM_Express
2. https://spdk.io/doc/nvmf_tgt_pg.html
3. https://spdk.io/doc/about.html
4. https://en.wikipedia.org/wiki/Remote_procedure_call
5. https://en.wikipedia.org/wiki/Remote_direct_memory_access
6. https://github.com/spdk/spdk/commit/0f912a0eafd92b3bb56b92d7bb3ab87ea965b3b4#diff-73782554262a7f916516b66d5dbf0afe
7. https://spdk.io/doc/about.html
8. https://dqtibwqq6s6ux.cloudfront.net/download/events/2019-summit/24+SPDK+-+It's+a+Bird+It's+a+Plane+It's+NVMe+over+TCP+and+more.pdf
9. https://mp.weixin.qq.com/s?__biz=MzI3NDA4ODY4MA==&mid=2653334751&idx=1&sn=eb13c596b3e938a093ef9dccc716dd97&chksm=f0cb5b58c7bcd24ee2c9a9d0a648f1dae83ab52d5449291af41655bc485be56f382809975aa7&token=431082989&lang=zh_CN#rd
10. https://github.com/spdk/spdk/commit/ed56a3d482383f77f225e83657ff1dcec4322161#diff-d39313363332ee3abd6079a5786e1881
11. https://dqtibwqq6s6ux.cloudfront.net/download/performance-reports/SPDK_19.04_NVMeOF_RDMA_benchmark_report.pdf
12. https://mp.weixin.qq.com/s?__biz=MzI3NDA4ODY4MA==&mid=2653334751&idx=1&sn=eb13c596b3e938a093ef9dccc716dd97&chksm=f0cb5b58c7bcd24ee2c9a9d0a648f1dae83ab52d5449291af41655bc485be56f382809975aa7&token=431082989&lang=zh_CN#rd
13. https://github.com/spdk/spdk/releases/tag/v19.01
14. https://optane.intel.com/
15. https://en.wikipedia.org/wiki/JSON-RPC
16. https://ark.intel.com/content/www/us/en/ark/products/123547/intel-xeon-silver-4110-processor-11m-cache-2-10-ghz.html
17. https://www.intel.com/content/www/us/en/products/memory-storage/solid-state-drives/data-center-ssds/dc-p4500-series.html
18. https://fio.readthedocs.io/en/latest/fio_doc.html
19. https://neurohive.io/en/popular-networks/resnet/
20. https://neurohive.io/en/popular-networks/vgg16/
21. https://dqtibwqq6s6ux.cloudfront.net/download/2019-summit-prc/02_Presentation_21_Introduction_of_Baidu_Chitu_Storage_with_SPDK_NVMe-oF_Application_Baidu_Zhenyuan.pdf