STORAGE DEVELOPER CONFERENCE
SDC EMEA 21
BY Developers FOR Developers
Virtual Conference
June 8, 2021

Development of software-defined storage engine for 25 million IOPS

Dmitrii Smirnov, AOSTO

Agenda

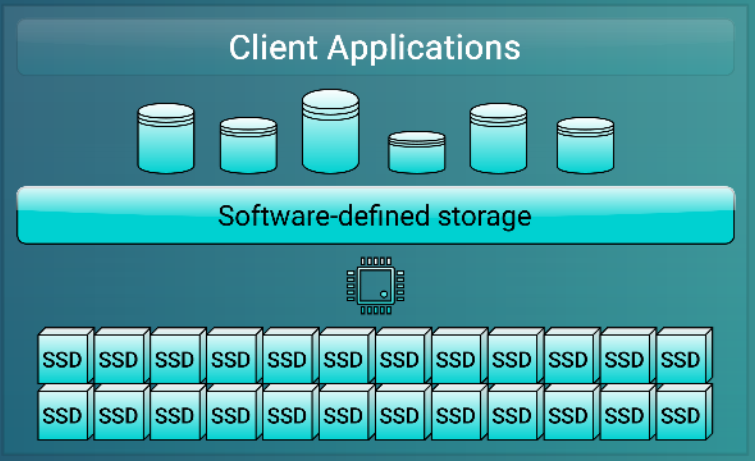
- Background
  - Problem
  - Challenge
- Software-defined storage solutions
  - Overview
  - Testing
  - Research
- Software-defined storage development
  - MVP
  - Testing
  - Research
  - Results
  - Linux kernel vs SPDK
  - Mistakes
  - Evaluation

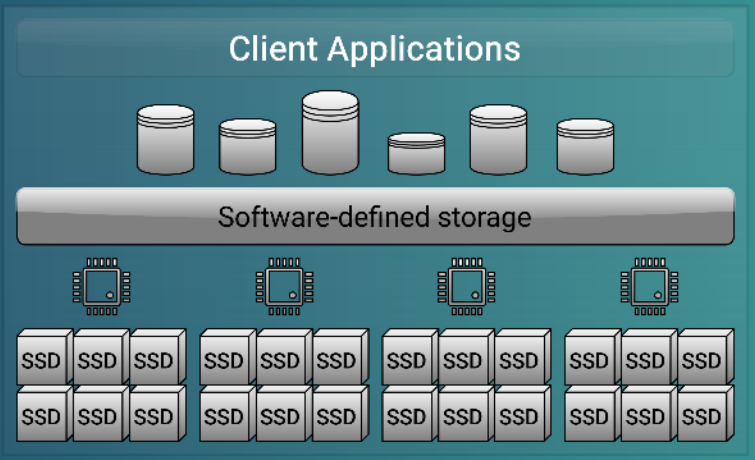
Background

- CPU
  - AMD EPYC 2nd/3rd Gen with PCI-e 4.0
  - Intel Xeon Ice Lake with PCI-e 4.0
- DPU
  - NVIDIA BlueField-2 200G and BlueField-3 400G
- NVMe
  - NVMe SSDs 1M+ IOPS and 6+ GiB/s
  - Ethernet Bunch of Flash (EBOF)
- Network
  - NVIDIA ConnectX-6 200G and ConnectX-7 400G
- Software
  - Linux kernel 5.11
  - Storage Performance Development Kit (SPDK)

Problem

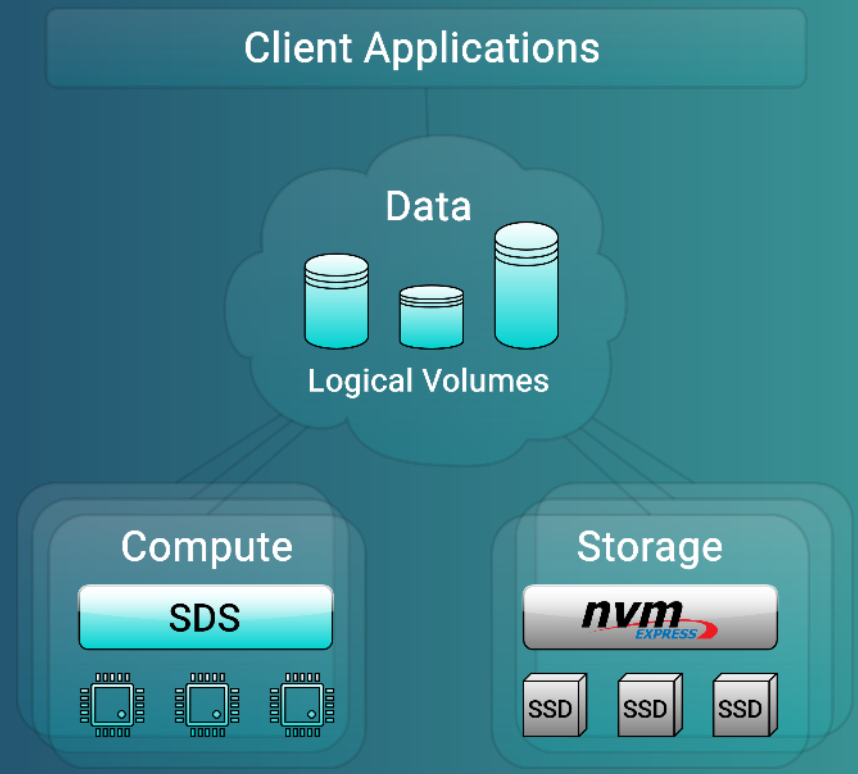
- Expectations
  - Single-node storage platform with 24 NVMe SSDs could provide up to 24M IOPS or 100 GiB/s
- Reality
  - single-node platform with 24 NVMe SSDs provides 10M IOPS
  - 2-node platform with 24 NVMe SSDs provides 20M IOPS
  - 4-node platform with 24 NVMe SSDs provides 24M IOPS
- Problems
  - Existing solutions do not use the full performance potential of modern hardware, wasting millions of IOPS
  - Dozens of inefficient storage nodes within cluster cause unnecessary infrastructure overgrowth
  - Increased data center design complexity, deployment and maintenance costs





Challenge

- Get the best performance from software-defined storage on modern hardware
  - AMD CPU, PCI-e 4.0, NVMe SSDs x24
- Find SDS engine with 24M IOPS on random read
- Provide single-node block storage for HPC, AI/ML/DL, Data analytics and Databases
- Get ready for the future hyperscale software-defined data center solutions, both hyperconverged and composable disaggregated infrastructure
  - PCI-e 5.0, EBOF, multi-node storage



Software-Defined Storage

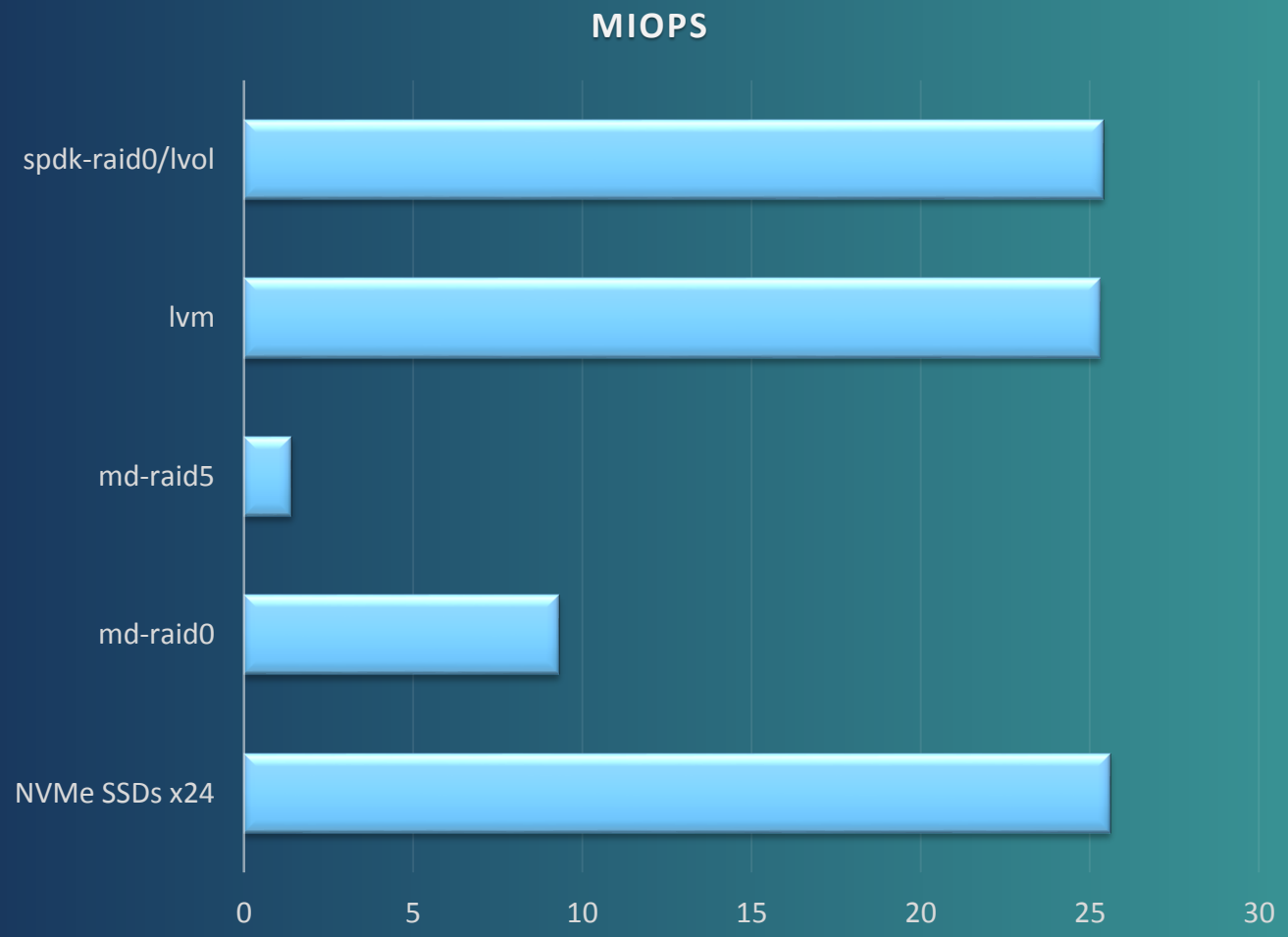
Overview

SDS solutions. Overview

- By data access
  - block, file, object
- By deployment and architecture
  - Single-node, multi-node
  - Centralized, Hyperconverged, Composable Disaggregated
  - SmartNIC/DPU-based
  - Target-side, initiator-side
- By use case
  - High-Performance Computing, Artificial Intelligence, Machine & Deep Learning, Real-time Analytics
  - Media & Entertainment, Post production, Streaming
  - Edge computing, Autonomous vehicles, 5G networks
- Open-source
  - Linux MDRAID, Volume Manager (LVM)
  - SPDK RAID, Volume Manager

SDS solutions. Testing NVMe

- Configuration
  - AMD EPYC 7742, 64 cores, 128 threads
  - NVMe SSDs x24, PCI-e 4.0
  - Fedora Server 34, Linux Kernel 5.11
  - SPDK v21.04
  - fio-3.26
  - I/O workload 4 KiB random read, QD=128 per drive
- Results
  - Kernel IOPS and latency are not OK
  - SPDK is OK
  - Research required



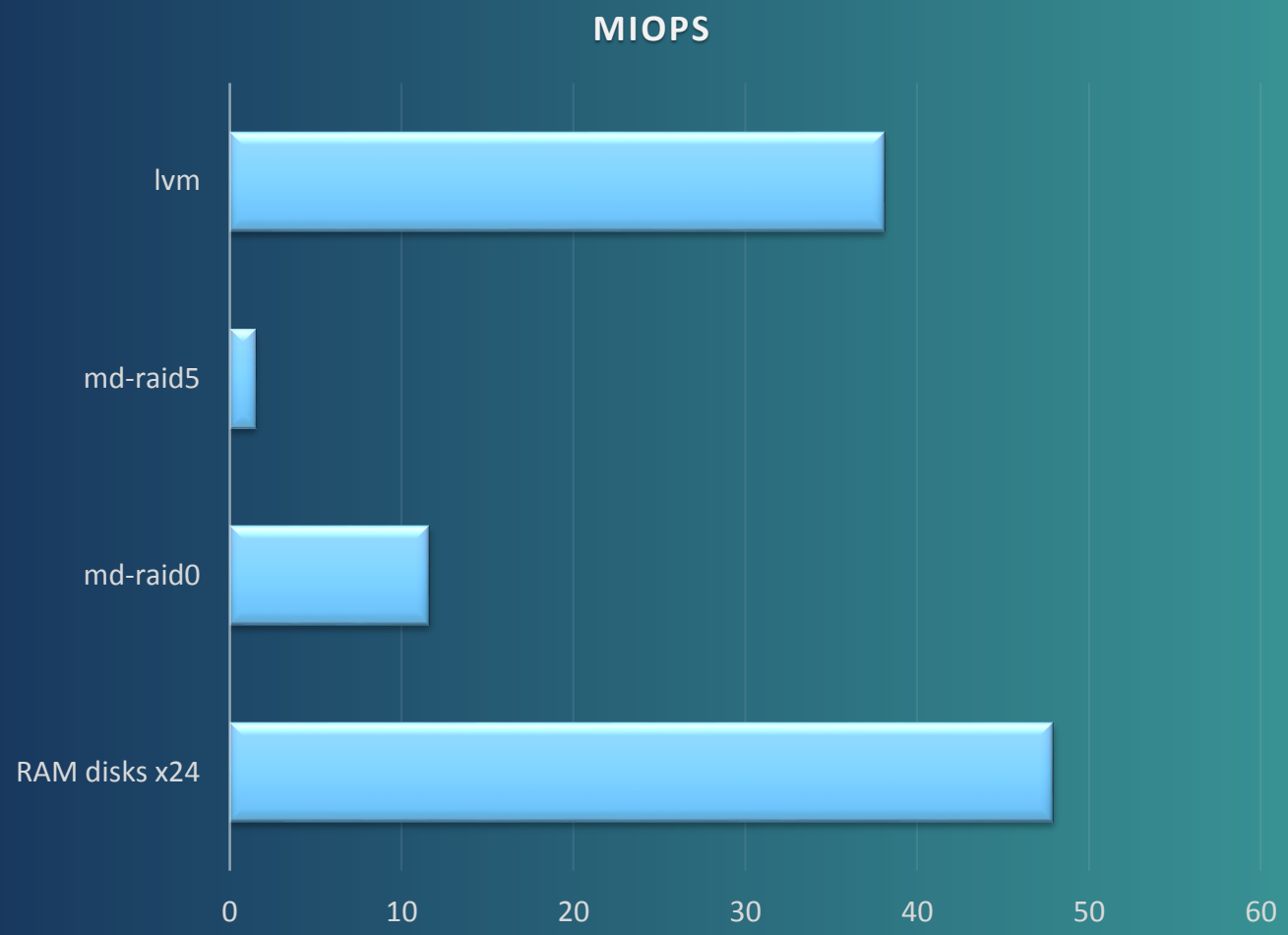
	MIOPS	Avg latency, usec	p99 latency, usec
NVMe SSDs x24	25,6	120	154
md-raid0	9,3	330	440
md-raid5	1,4	2200	2600
lvm	25,3	121	159
spdk-raid0/lvol	25,4	120	157

SDS solutions. Research

- > Why only read-intensive workload for IOPS and latency metrics?
  - Read-intensive workload shows potential performance and software bottlenecks, as opposed to write-intensive, which depends on drive health, garbage collector, journal, thin allocation, read-modify-write
- > Could we use high-performance SPDK RAID and Volume Manager for SDS?
  - No RAID metadata
  - Datapath (RAID, logical volumes) is not intended for cluster SDS
  - Data services (thin provisioning, snapshots) are not intended for cluster SDS
  - No local block device (optional requirement)
- > Why is kernel software performance so much worse than the actual NVMe SSDs performance?
  - Existing kernel software is still heavy and is not CPU-friendly
  - CPU - main resource for IOPS and latency performance
  - «Is Parallel Programming Hard, And, If So, What Can You Do About It?», Paul E. McKenney
- > How validate the real performance potential of SDS?
  - IOPS and latency testing on NULL or RAM devices

SDS solutions. Testing RAM

- Configuration
  - AMD EPYC 7742, 64 cores, 128 threads
  - RAM devices x24
  - Fedora Server 34, Linux Kernel 5.11
  - SPDK v21.04
  - fio-3.26
  - I/O workload 4 KiB random read, QD=128 per drive
- Results
  - Kernel IOPS and latency are not OK
  - SDS development required



	MIOPS	Avg latency, usec	p99 latency, usec
RAM disks x24	47,9	64	87
md-raid0	11,6	264	375
md-raid5	1,5	2050	2500
lvm	38,1	80	110

Software-Defined Storage

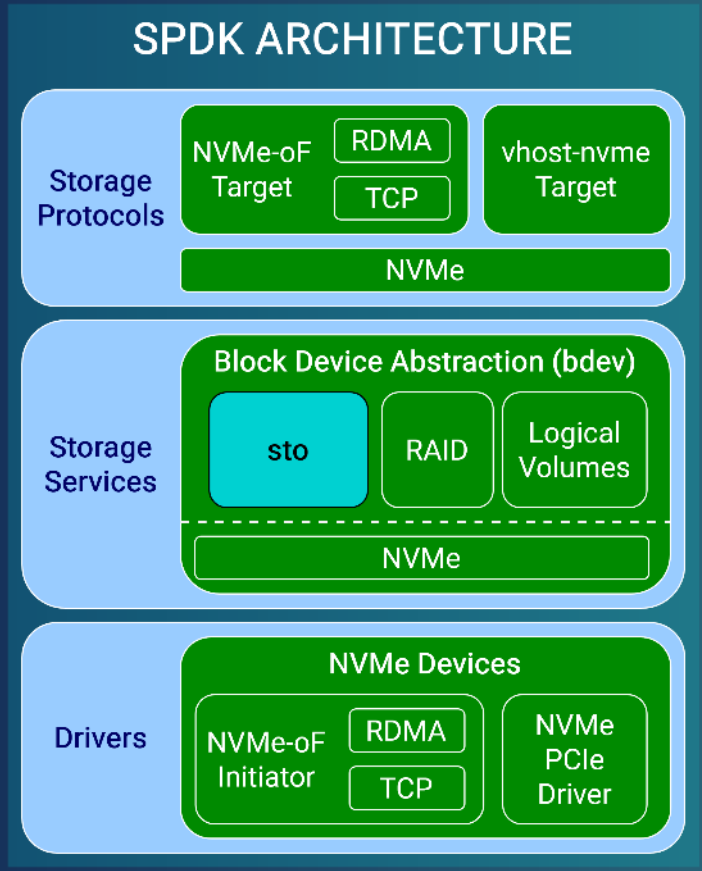
Development

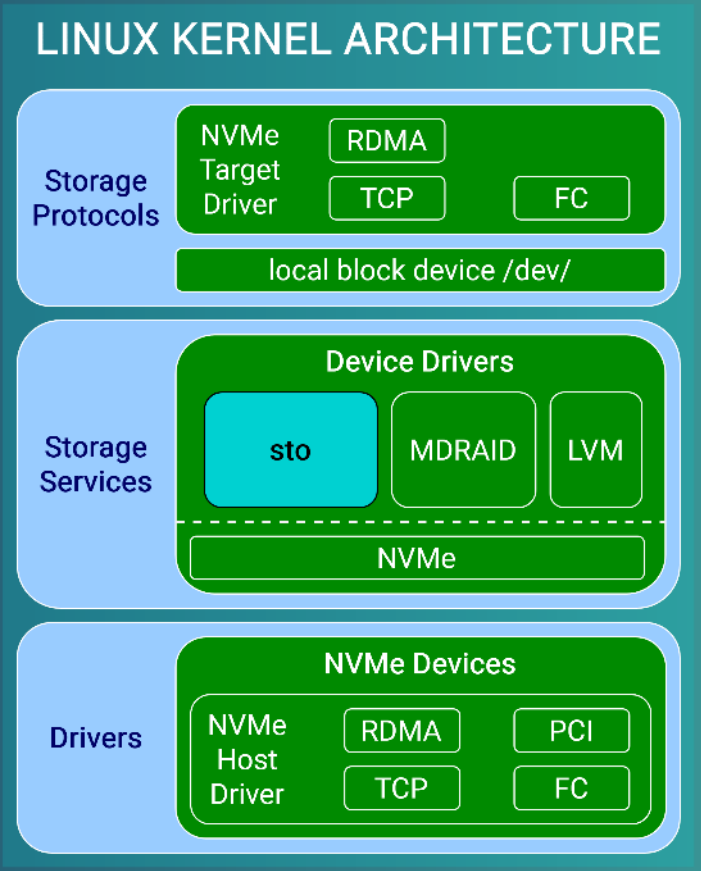
SDS development. MVP

- Software-defined storage engine
  - Datapath, RAID, volume manager, pool management, erasure coding, thin provisioning, rebuild, metadata, QoS
  - ~~Reconfiguration (level migration and scaling), journal, spare disk, spare (empty) block, snapshots, clones, compression, deduplication, ZNS support~~
- C programming
- Linux Kernel and SPDK support
  - Without differences in the core and cluster logic, the same code and algorithms
  - Kernel block device
  - SPDK virtual bdev
- Maximum performance
  - Read-intensive workload, IOPS and latency metrics

SDS development. Kernel vs SPDK

- Linux kernel
  - Use cases
    - User app needs local block device
    - Filesystem in kernel
  - Features
    - Provides local block device
    - No specific architecture requirements
- SPDK
  - Use cases
    - User app in userspace
    - Filesystem in userspace
    - Virtualization, SPDK vhost
    - NVMe/TCP, NVMe/RDMA
  - Features
    - Can run in containers
    - Requires lockless architecture for datapath and data services
- SDS module - sto

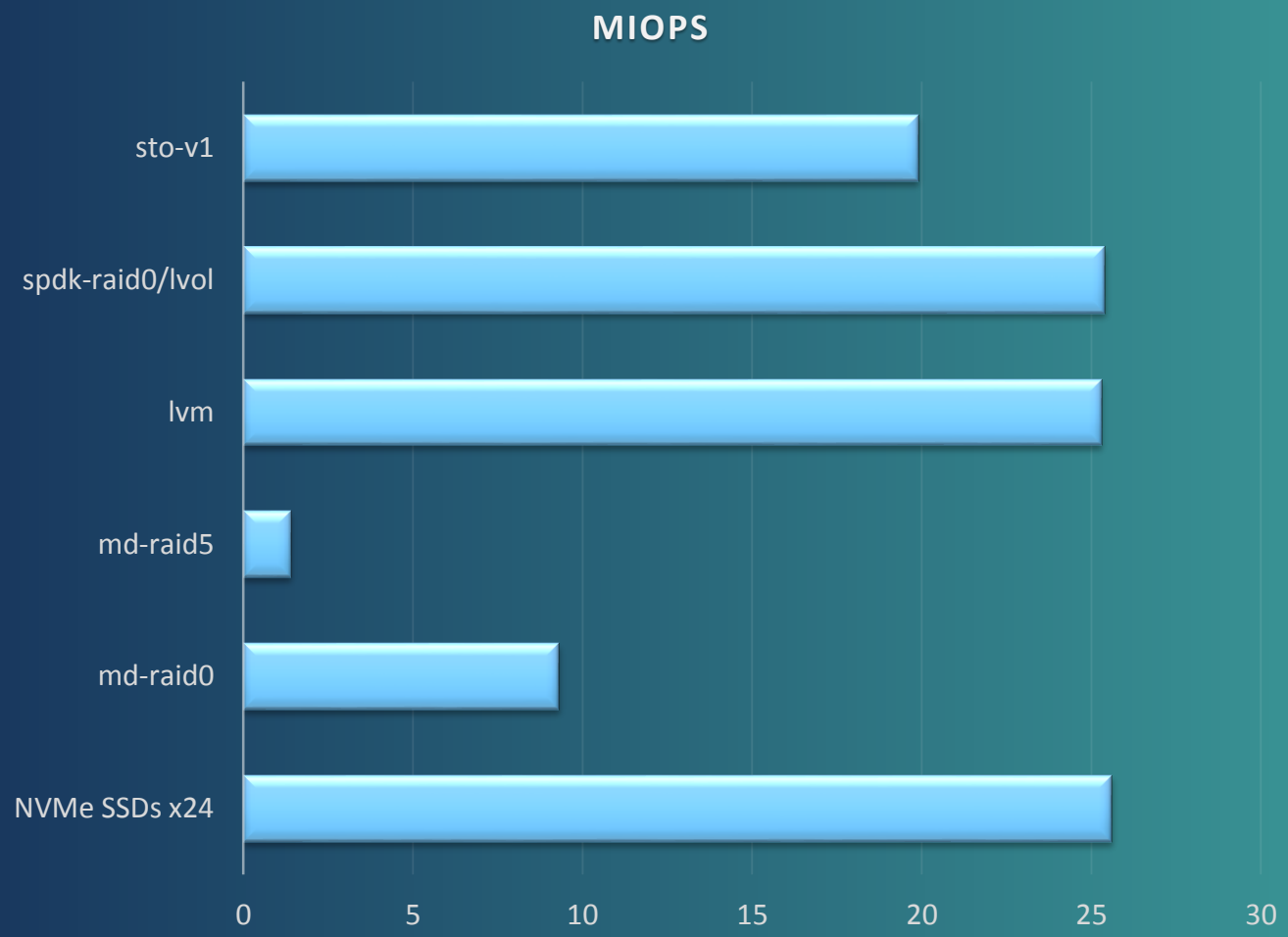




SDS development. Testing NVMe

- Configuration
  - AMD EPYC 7742, 64 cores, 128 threads
  - NVMe SSDs x24, PCI-e 4.0
  - Fedora Server 34, Linux Kernel 5.11
  - SPDK v21.04
  - fio-3.26
  - I/O workload 4 KiB random read, QD=128 per drive
- Results
  - Kernel sto-v1 IOPS are not OK
  - Kernel sto-v1 latency is OK
  - Research required

	MIOPS	Avg latency, usec	p99 latency, usec
NVMe SSDs x24	25,6	120	154
md-raid0	9,3	330	440
md-raid5	1,4	2200	2600
lvm	25,3	121	159
spdk-raid0/lvol	25,4	120	157
sto-v1	19,9	154	205



SDS development. Testing RAM

- Configuration
  - AMD EPYC 7742, 64 cores, 128 threads
  - RAM devices x24
  - Fedora Server 34, Linux Kernel 5.11
  - SPDK v21.04
  - fio-3.26
  - I/O workload 4 KiB random read, QD=128 per drive
- Results
  - Kernel sto-v1 IOPS are not OK
  - Kernel sto-v1 latency is OK
  - Research required

	MIOPS	Avg latency, usec	p99 latency, usec
RAM disks x24	47,9	64	87
md-raid0	11,6	264	375
md-raid5	1,5	2050	2500
lvm	38,1	80	110
sto-v1	23,1	132	184

SDS development. Research

- Bottleneck search by disabling SDS components
  - Datapath (fast path) disassembly
  - Inline and background data services
    - Logical allocation tables, parallel data access, stripe ownership, QoS and request counting
  - Linux kernel block layer, thread and memory management
- Bottleneck types
  - Code problems
    - Heaviness, using a lot of CPU resources per 1M IOPS
  - Logic problems
    - CPU or NVMe resources are not fully utilized, idling, scheduling

SDS development. Research #2

- General suggestions
  - Do not use blk-mq or device-mapper for IOPS target, if it is not necessary
  - Do not use block schedulers for IOPS target
  - Avoid moving from current thread context even within the same NUMA-node and CPU
  - Avoid abusing kmem_cache for a lot of wrappers
  - Avoid kernel API with spinlocks and atomics inside
    - kthread_queue_work, queue_work, wake_up_process, etc.
- Not recommended, but better than nothing
  - Fine-grained spinlocks and atomics, read/write locks and semaphores
  - Read-Copy-Update (RCU) is not a panacea for write-mostly and consistent data
  - Polling-based thread management

SDS development. Research #3

- Solutions
  - Use a bio-based queue and develop your own bio merge algorithms
  - Stay in the thread context
  - Per-CPU and thread-aware architecture
    - Develop your own lockless queue_work using kthreads API, if it is necessary
  - Use non-blocking synchronization techniques
    - Wait-free, Lock-free, Obstruction-free, Clash-free
  - Parallel-Fastpath Design Patterns
    - Fine-grained atomic CAS-based algorithms
    - Fine-grained bitmaps and lockless lists for data access

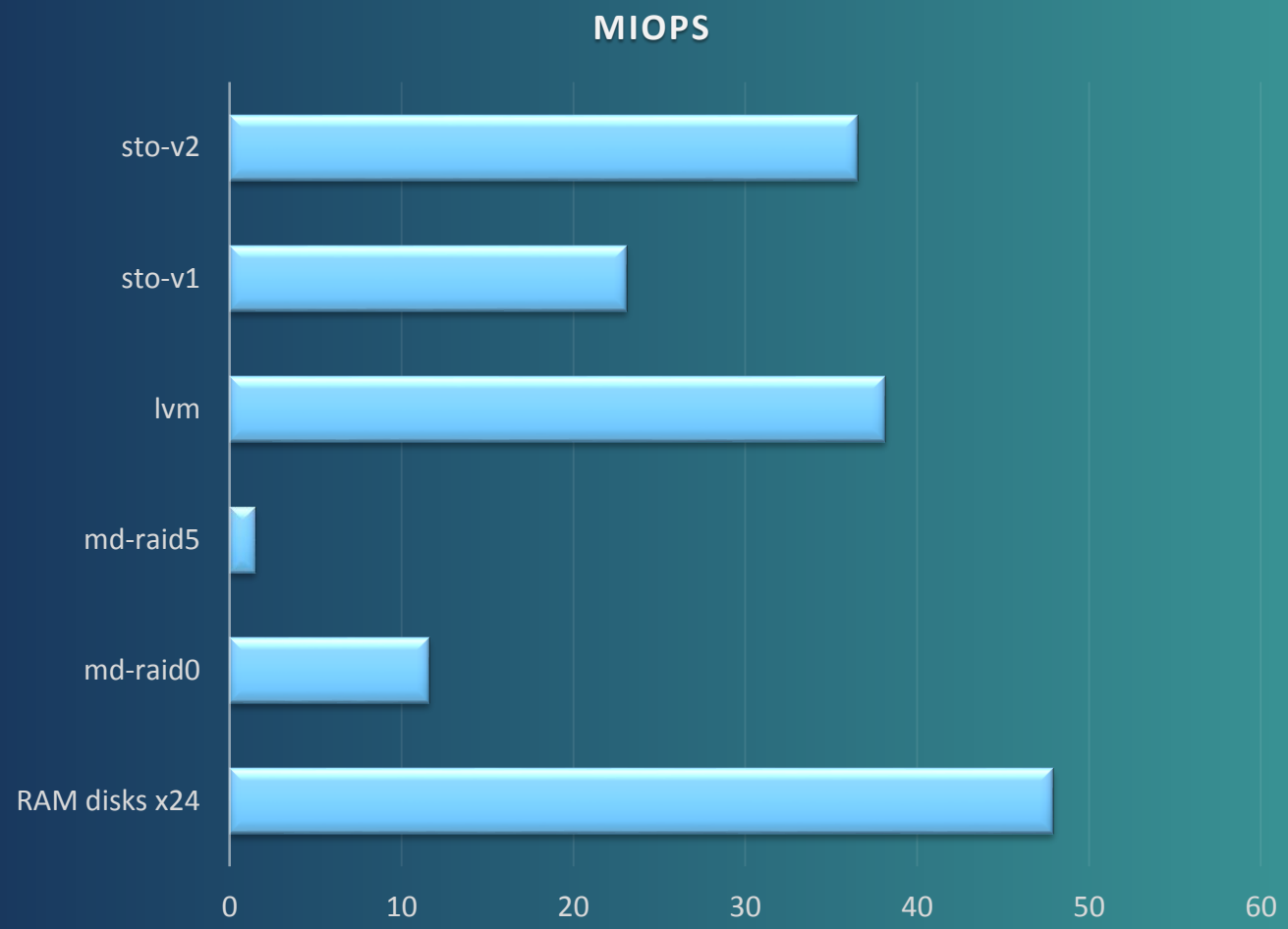
SDS development. Data services

- Multi-tenancy and noisy neighbor problem
  - QoS
    - Per-cpu/thread objects and RCU for read-mostly data
  - Mapping for logical and physical extents (allocation table)
    - Per-cpu/thread reserved data areas with offsets
  - io merge algorithms
    - Per-cpu/thread trees for sequences and ranges
  - Stripe cache
    - Per-cpu/thread pools and reserved buffers
  - Stripe access
    - Fine-grained bitmaps and CAS-based ownership
  - Journal and snapshots (copy-on-write, redirect-on-write)
    - Fine-grained bitmaps and reserved areas within consistency groups

SDS development. Results on RAM

- Configuration
  - AMD EPYC 7742, 64 cores, 128 threads
  - RAM devices x24
  - Fedora Server 34, Linux Kernel 5.11
  - SPDK v21.04
  - fio-3.26
  - I/O workload 4 KiB random read, QD=128 per drive
- Results
  - Kernel sto-v2 IOPS and latency are OK

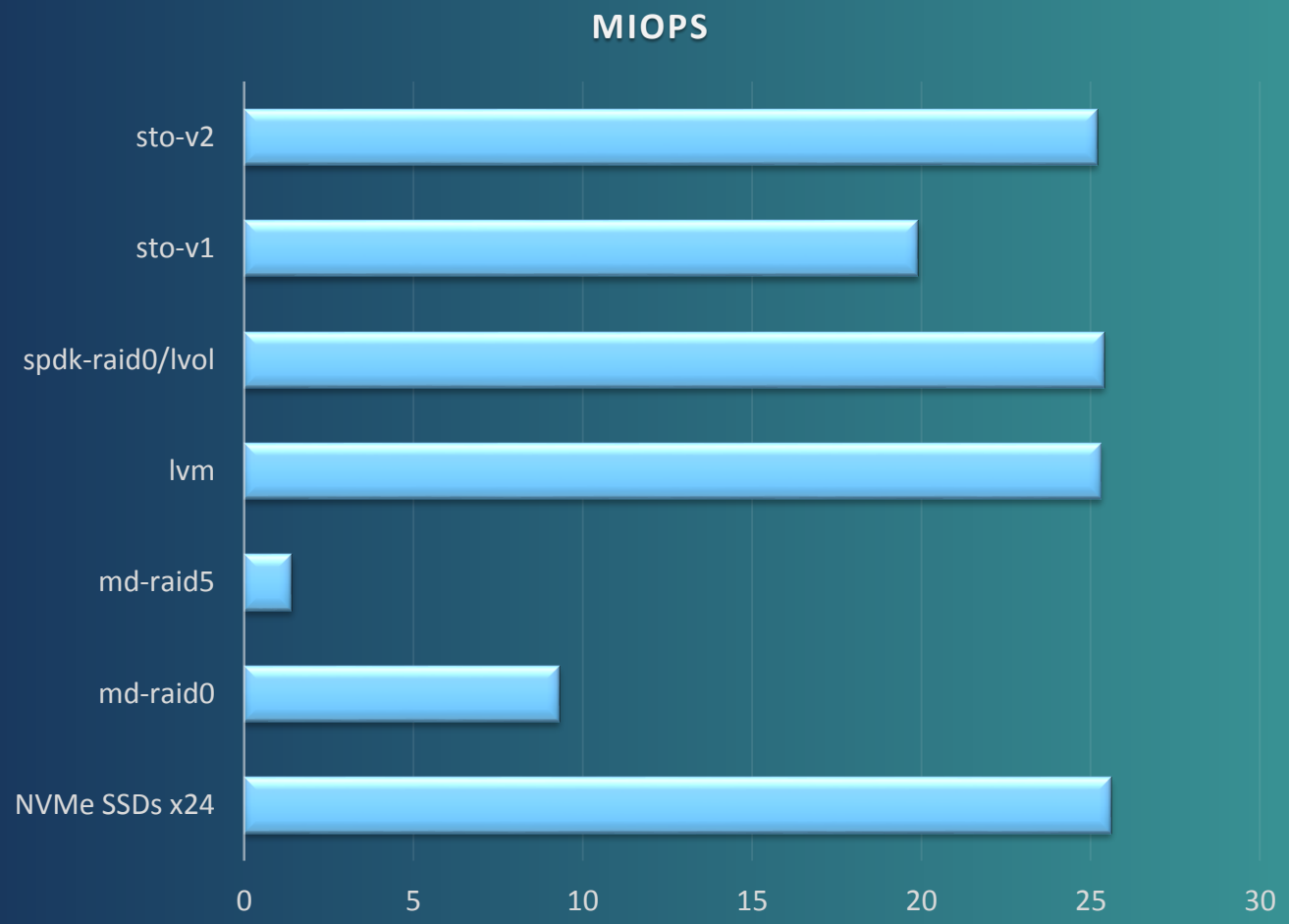
	MIOPS	Avg latency, usec	p99 latency, usec
RAM disks x24	47,9	64	87
md-raid0	11,6	264	375
md-raid5	1,5	2050	2500
lvm	38,1	80	110
sto-v1	23,1	132	184
sto-v2	36,5	84	114



SDS development. Results on NVMe

- Configuration
  - AMD EPYC 7742, 64 cores, 128 threads
  - NVMe SSDs x24, PCI-e 4.0
  - Fedora Server 34, Linux Kernel 5.11
  - SPDK v21.04
  - fio-3.26
  - I/O workload 4 KiB random read, QD=128 per drive
- Results
  - Kernel sto-v2 IOPS and latency are OK
  - Ready for SPDK

	MIOPS	Avg latency, usec	p99 latency, usec
NVMe SSDs x24	25,6	120	154
md-raid0	9,3	330	440
md-raid5	1,4	2200	2600
lvm	25,3	121	159
spdk-raid0/lvol	25,4	120	157
sto-v1	19,9	154	205
sto-v2	25,2	121	160



Summary. Kernel vs SPDK

- Linux kernel
  - Easy to develop SDS, hard to get performance
  - Lets you use whatever you want and helps you. But every memory allocation or atomic variable will hurt you and waste millions of IOPS
  - You have no right to make a mistake in a performance architecture
- SPDK
  - Hard to develop SDS, easy to get performance
  - Requires lockless SDS design, event-driven, poll-based. But does not punish you for atomics, inefficient code and heavy algorithms
  - You have the right to make a mistake
- Solution
  - Non-blocking synchronization techniques and parallel-fastpath design patterns will fix all your issues in both Linux kernel and SPDK

Software-Defined Storage

Mistakes

Summary. Mistakes

- > We removed this potential bottleneck and didn’t notice an increase in performance, probably not a problem
  - Probably a much larger bottleneck hides the current bottleneck
  - The performance optimization graph is not linear - the first bottleneck costs 15M IOPS, the second costs 10M, the third costs 5M, and so on
- > It's just an io request counter
  - Even one atomic variable for statistics/traces/QoS could waste from 1M to 15M IOPS
- > If there are no problems on the flamegraph, there is nothing to optimize
  - If you don’t have outliers - the whole code is equally slow

Summary. Mistakes #2

- > Let’s optimize datapath but the background data service will not
  - Performance optimization approach – all or nothing
  - The more efficiently a background data service uses CPU, the more resources remain for the datapath
- > Let’s assign only a part of all CPU cores to datapath to free resources for other applications
  - The less resources the datapath uses, the less performance you get
  - The solution – per CPU datapath, per CPU data services (optional)
- > Let’s balance datapath between CPU cores to get more performance
  - Moving the io request to another CPU takes time, unlike handling in the current context
  - Balancing datapath between CPUs reduces system performance and increases the io request latency
- > Let’s limit the CPU usage
  - CPU usage by SDS depends on incoming data workload
  - If you limit the CPU usage, you limit the IOPS

Summary. Mistakes #3

- > We develop in userspace and have no performance issues
  - No matter how you develop and run storage software - kernel or DPDK/SPDK, storage node or SmartNIC/DPU
- > We don’t need high-performance processor for software development
  - Don’t be sure you unleash the full hardware potential
- > Let's not reinvent the wheel, use the generic system API and hope it is optimized
  - Don't be sure other developers are testing on multi-million IOPS hardware

Summary. Evaluation

- Goal
  - Researching performance bottlenecks using read-intensive workloads
  - Unlocking the software potential on modern hardware
- Benchmarks
  - Synthetic workloads vs real-world workloads
  - Industry standards
    - SPC-1, SPC-2, SNIA SSS PTS, SPECstorage Solution 2020, STAC-M3, IO-500
- Applying
  - Single-node local storage for cloud-native applications
  - Disaggregated storage for HPC using JBOF/EBOF
- Limitations
  - PCIe 4.0 x128, DDR4, Network

What’s next

- Multi-active software-defined storage for hyperconverged and composable disaggregated infrastructure
- SmartNIC/DPU integration
- Cloud-native API integration
- Open-source

Please take a moment to rate this session.

Your feedback is important to us.

