



Awide Data Processor · Storage Acceleration

Same SSDs. Up to 6x faster writes.

Inline compression, parity RAID and write shaping in one low-profile PCIe card - measured against Linux software RAID on the same four NVMe drives.



BLOCK MODE



KV MODE



INLINE COMPRESSION

CPU-centric storage can no longer keep up with NVMe flash

This is an on-prem problem. In the cloud you pay for it; on-prem you own it.



Massive host CPU overhead

Up to 80% of CPU cycles go to storage I/O, compression, garbage collection and compaction instead of application logic.



SSD wear and tail latency

Write amplification from random writes and compactions burns flash endurance and causes p99.9 latency spikes.



The RAID rebuild crisis

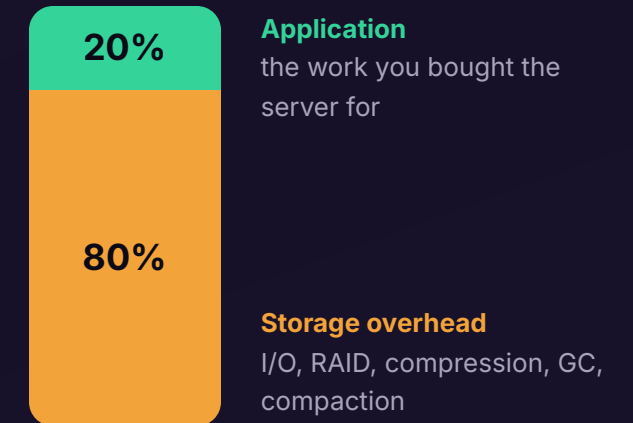
Rebuilding 16-128 TB drives with software RAID or erasure coding takes days or weeks, degrading performance and risking data.



Skyrocketing TCO

Fast drive degradation and poor space utilization force constant over-provisioning.

Where host CPU cycles go



Scaling general-purpose CPUs to solve storage bottlenecks is no longer viable. Modern workloads need dedicated hardware data acceleration.

Three simple principles, done in hardware



Write less

Compress user data before it reaches the SSD

Fewer bytes written: less wear, more usable capacity



Write big and sequential

Write to SSD in 1-2 GB chunks

Less SSD-internal write amplification, stable latency



Write sorted

Sort data by object identifier

A small mapping table, so metadata stays fast

APPLICATION WRITES



small random 8 KB writes, in any order



ADP

compress · aggregate · sort



one large, compressed, sequential write to the SSDs

One low-profile card, 48 drives, block or KV mode



Block mode

A drop-in block device. Standard filesystem, standard database, no code changes. Installed with the block bundle; every FIO and PostgreSQL result in this deck uses it.



KV mode

A native key-value interface that bypasses the filesystem and block layer. Installed with the KV bundle; used by KV databases, PostgreSQL on KV, pgvector and LLM KV-cache offloading.

Working modes	Block and KV
Random read / write	15M IOPS / 15M IOPS
Sequential read / write	56 GB/s / 56 GB/s
Capacity	Up to 2 PB raw
Supported drives	Up to 48
Host interface	PCIe Gen 5 ×16
SSD interface	NVMe (Gen 3/4/5), NVMe-oF, SAS, SATA
SSD types	TLC, QLC, ZNS
Form factor	Low-profile HHHL, single slot

One key-value engine in hardware, many workloads

The KV bundle exposes ADP's key-value store directly - no filesystem and no block layer in between.



Key-value databases

Redis-class caches and stores, MongoDB and other KV engines: keys go straight to the card.



RocksDB

XDP-Rocks: RocksDB on ADP KV, without host-side compaction storms.



MongoDB

XDP-WiredTiger: the MongoDB storage engine backed by ADP KV.



PostgreSQL on KV

PostgreSQL storage moved from POSIX files to a key-value backend.



pgvector

HNSW index builds and vector search when data outgrows RAM.



LLM KV-cache offloading

An NVMe KV-cache tier below GPU memory for LLM inference.

Block and KV are separate installation bundles of the same card. LLM KV-cache offloading has its own deck at awide.io/ai-kv-caching.

Same server, same four drives, two arrays

Test server

Server Dell PowerEdge R750

CPU 2× Xeon Gold 6346, 3.1 GHz

RAM 512 GB

Network 1× 100 Gb/s Mellanox

Drives 4× Samsung PM9A3 3.84 TB Gen4

OS Ubuntu 24.04

MDRAID RAID10

Linux software RAID



4 × 3.84 TB = 15.4 TB raw

7.7 TB usable

50% of raw: every block is mirrored

ADP RAID5 + COMPRESSION

Awide Data Processor



4 × 3.84 TB = 15.4 TB raw

15.4 TB exposed

parity instead of mirrors, plus inline compression

2×

the capacity of RAID10 from the same drives, with parity protection

4 of 48

drives used: the results show the limits of four Gen4 SSDs, not of the card

RAID5

is avoided in software for databases (read-modify-write); ADP writes full stripes

Raw block devices, compressible data, 800+ FIO runs

1

Wipe

Every drive is fully discarded before each array is built.

2

Build

mdraid RAID10 with mdadm; ADP installed as RAID5 with the block bundle.

3

Run FIO

Directly on the block device, no filesystem. Data is 67% compressible, like typical database pages.

4

Collect

FIO JSON and iostat are recorded per run, then parsed into one CSV per array.

```
# every drive wiped before each array is built
blkdiscard /dev/nvmeXn1

# baseline: Linux software RAID10
mdadm --create /dev/md126 --level=10 --raid-devices=4 \
  /dev/nvme0n1 /dev/nvme1n1 /dev/nvme2n1 /dev/nvme3n1 \
  --force --assume-clean

# ADP: block (FSA) bundle, RAID5 over the same drives
xdp-installer install --xdp=1 --level=5 \
  --resources="<four NVMe drives>" \
  --assume-yes --enable-set-hugepages

# one FIO job; pattern, block size, jobs, depth vary
fio --direct=1 --ioengine=libaio --group_reporting=1 \
  --buffer_compress_percentage=67 --buffer_compress_chunk=4096 \
  --refill_buffers=1 --norandommap=1 --randrepeat=0 \
  --random_generator=tausworthe64 --rw=<pattern> \
  --bs=<8k|16k|1m> --numjobs=<1-128> --iodepth=<1-32> \
  --runtime=60 --filename=<raw block device>
```

Shown in this deck: 63 matched FIO configurations per array (126 runs), selected from a suite of 800+ runs. Queue depth (QD) = jobs × iodepth.

Biggest gains where it hurts most: random writes

6.4×

random write, 8 KB
6.9 vs 1.1 GB/s

4.2×

mixed 70/30 random, 8 KB
13.2 vs 3.2 GB/s

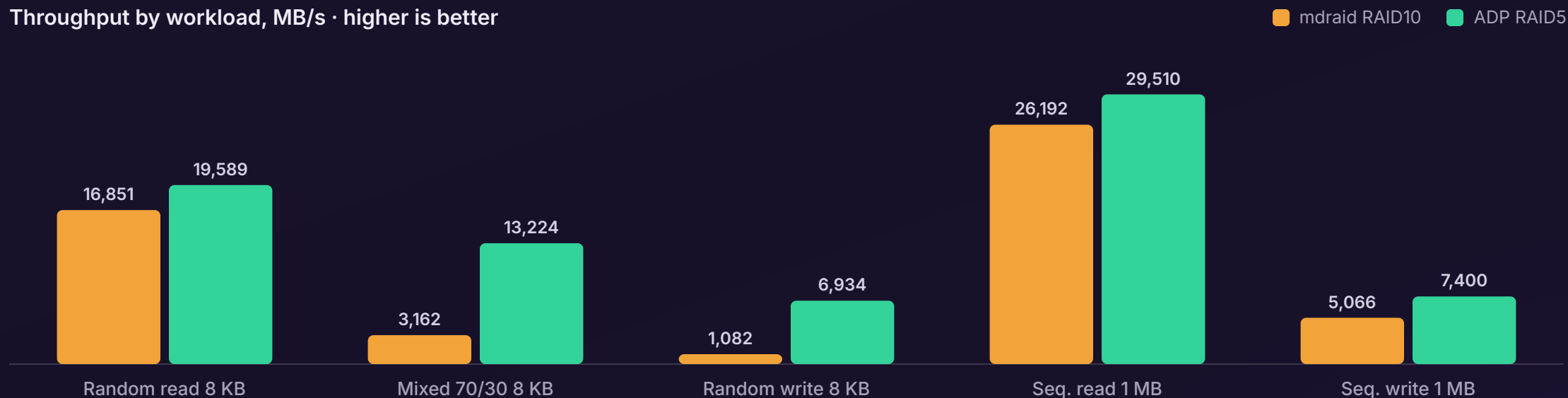
+46%

sequential write, 1 MB
7.4 vs 5.1 GB/s

+16%

random read, 8 KB
19.6 vs 16.9 GB/s

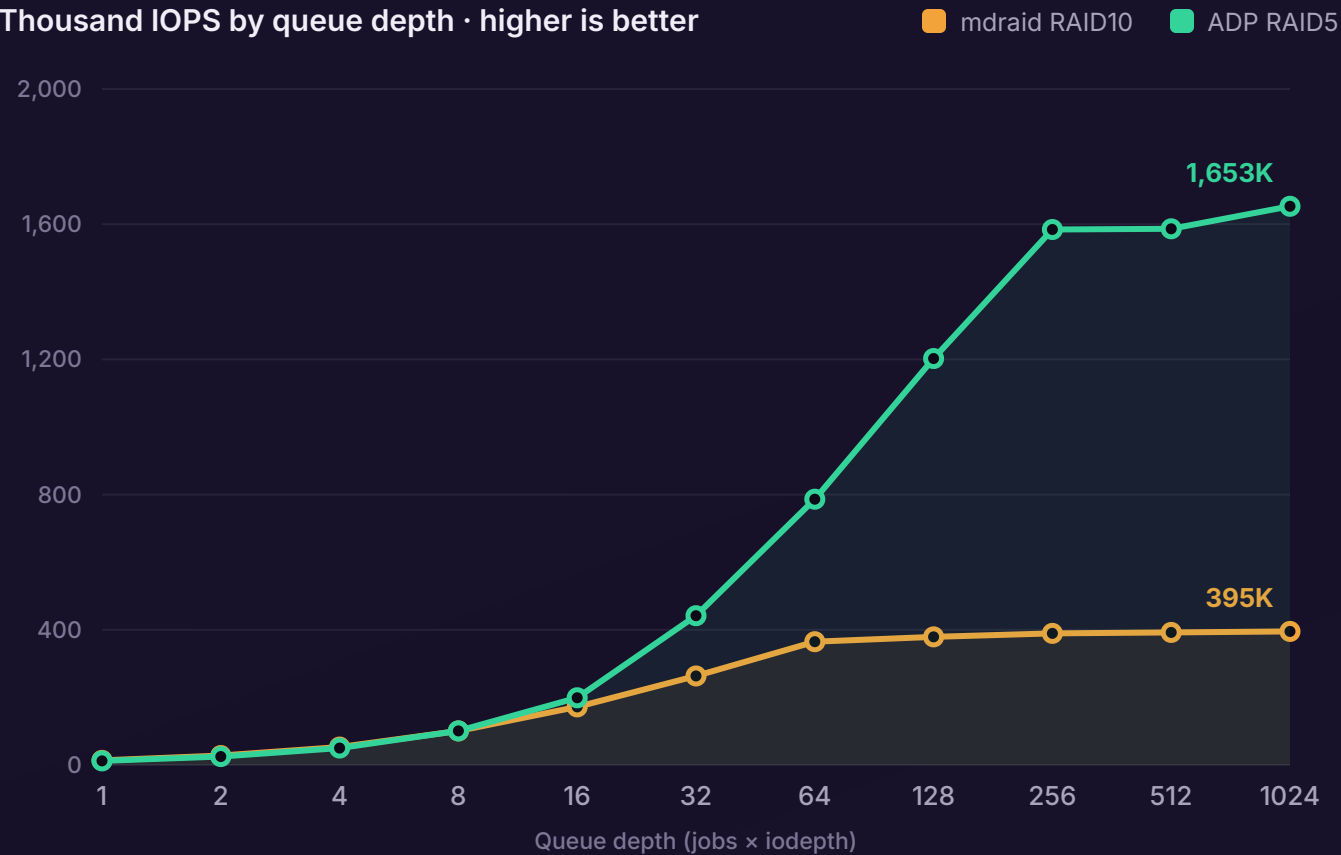
Throughput by workload, MB/s · higher is better



Same FIO job on both arrays: random read QD 1024 (32×32), mixed QD 1024 (128×8), random write QD 64 (8×8), sequential QD 32 (8×4).

Under load mdraid flattens. ADP keeps scaling.

Thousand IOPS by queue depth · higher is better



4.2×

IOPS at QD 1024:
1.65M vs 395K

2.4×

lower p99 read latency at QD 256:
0.72 vs 1.74 ms

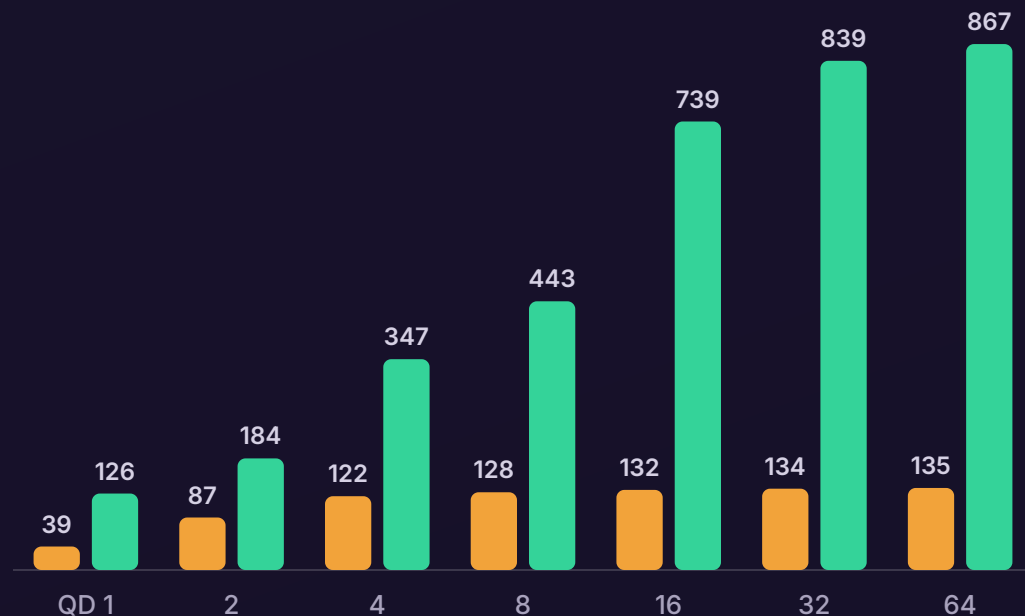
29×

lower p99 write latency at QD 256:
0.12 vs 3.52 ms

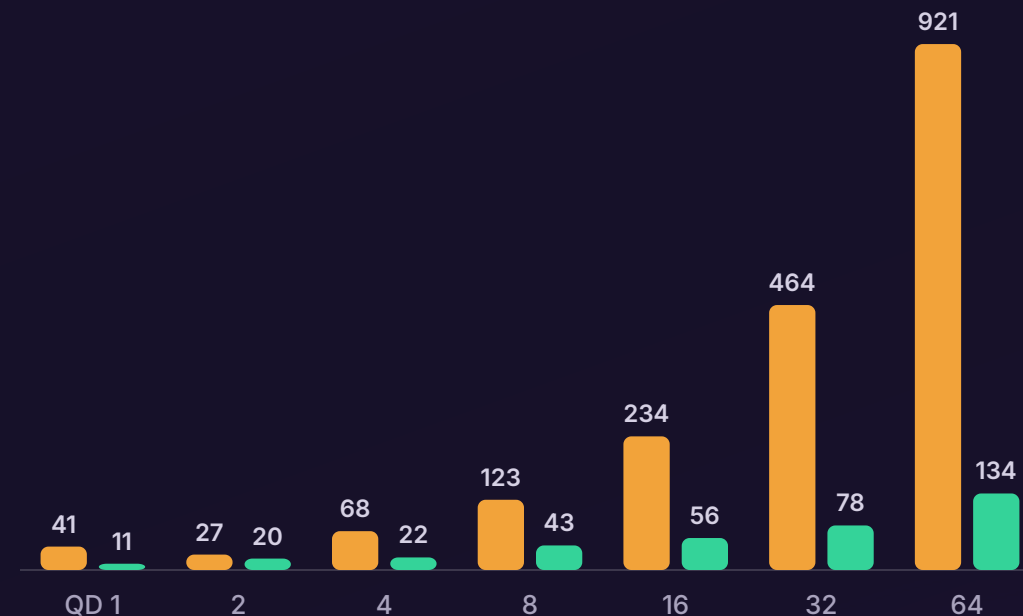
Up to QD 8 the arrays are at parity; the gap opens with concurrency, which is where production databases live. At 16 KB: 18.6 vs 3.3 GB/s (5.6×).

Six times the write IOPS at a fraction of the latency

Thousand IOPS by queue depth · higher is better



p99 write latency, μ s · lower is better



mdraid RAID10 ADP RAID5

At QD 64: 867K vs 135K IOPS (6.4 $\times$), mean latency 71 vs 458 μ s. mdraid stops scaling at QD 8: two mirrored pairs absorb about 135K random 8 KB writes. At 16 KB: 7.1 vs 1.1 GB/s (6.6 $\times$).

More capacity and fewer bytes on flash, from the same drives

Capacity from 4 × 3.84 TB NVMe

mdraid RAID10 mirroring



ADP RAID5 parity, before compression



ADP RAID5 + compression capacity exposed



2× the usable capacity of RAID10, with parity protection



Fewer bytes reach flash

The test data is 67% compressible, close to real database pages. Inline compression writes roughly a third of the bytes the host sends.



Faster than the drives write randomly

867K random 8 KB write IOPS from four drives, where mdraid gets 135K: small random writes leave ADP as large compressed sequential stripes.



Parity, not mirrors, and less wear

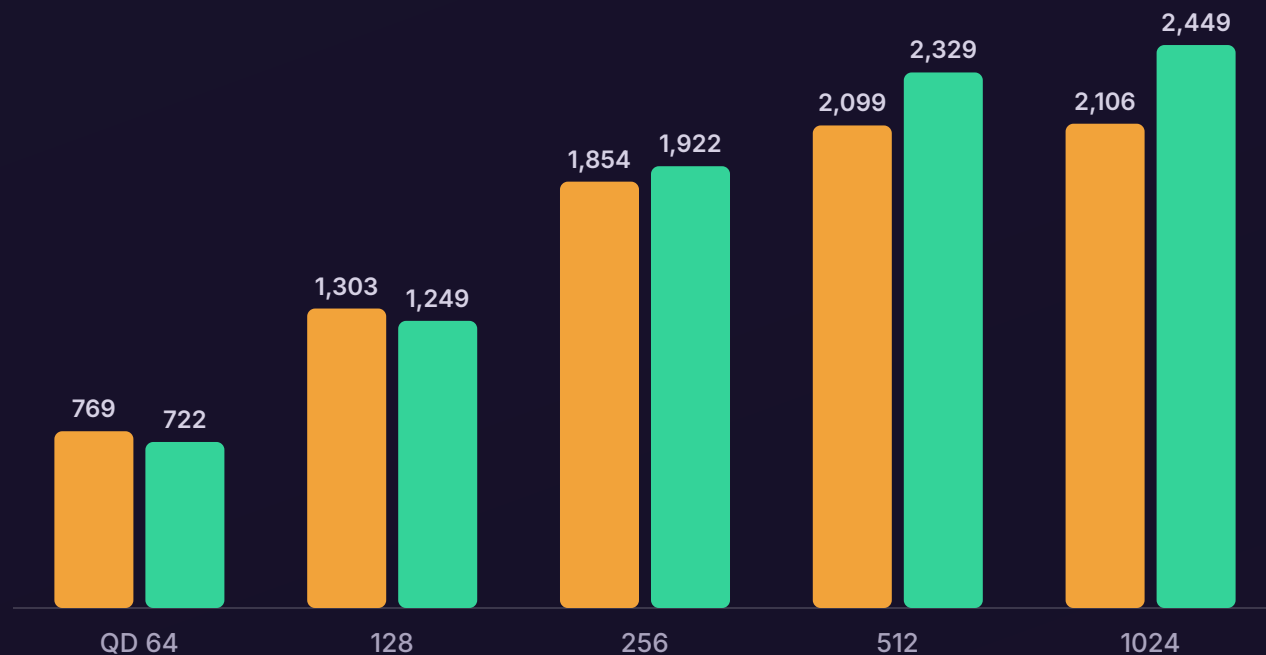
RAID5 without the read-modify-write penalty. Fewer, larger writes cut SSD-internal write amplification and extend drive life.

The capacity exposed with compression depends on how well the data compresses. FIO: buffer_compress_percentage=67, buffer_compress_chunk=4096, refill_buffers=1.

Reads at parity, with up to half the host CPU

Thousand IOPS by queue depth · higher is better

mdraid RAID10 ADP RAID5



2×

less system CPU for the same reads:
1.25M IOPS at 15% vs 1.30M at 30% (QD 128)

+28%

16 KB random reads at QD 256: 1.57M vs 1.23M IOPS, at 20% vs 33% system CPU

+16%

8 KB at QD 1024: 2.45M vs 2.11M IOPS, once mdraid saturates

Where mdraid wins: a single outstanding read (70 vs 111 μ s at QD 1) and large sequential reads at low queue depth (QD 8: 23.4 vs 11.5 GB/s). At QD 32 ADP leads, 29.5 vs 26.2 GB/s. CPU is FIO-reported system time.

The fuller the database, the bigger the advantage

WAL sync · pg_test_fsync

2.27x

faster synchronous WAL writes

mdraid · open_datasync

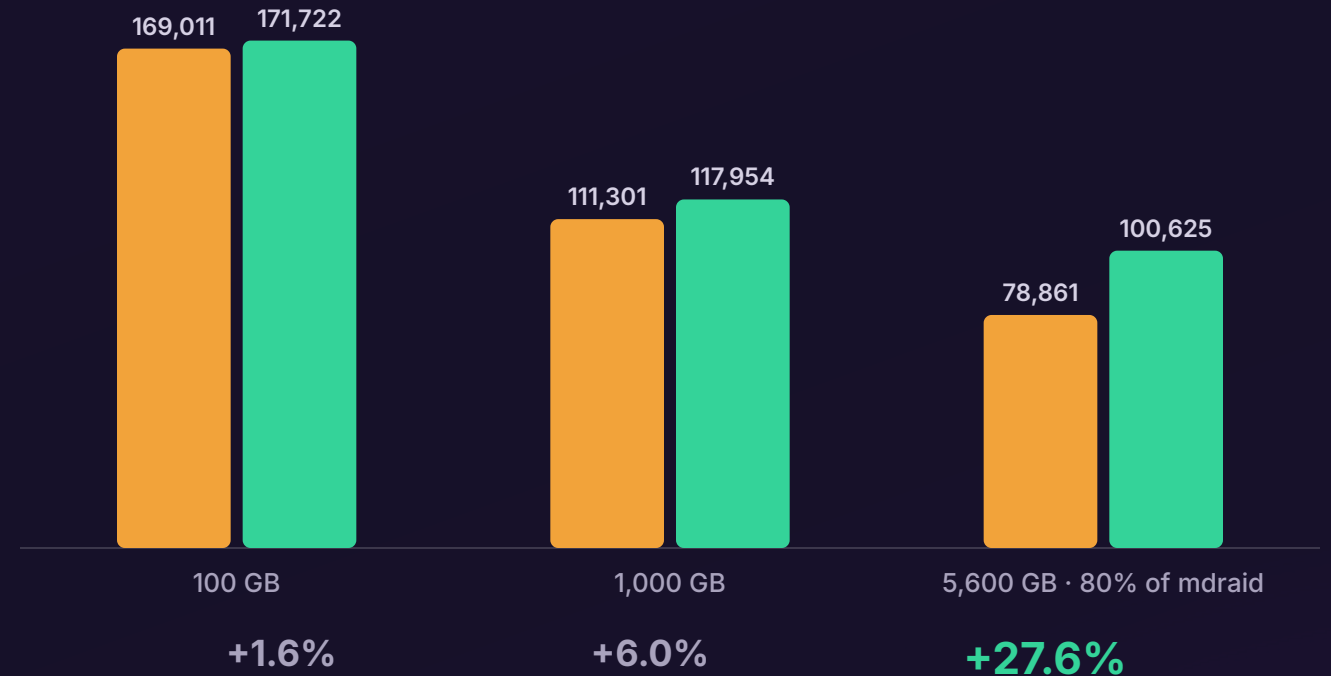
51,984 ops/s

ADP · open_datasync / open_sync

118,000 ops/s

pgbench (TPC-B), TPS by dataset size

mdraid RAID10 ADP RAID5



Small datasets fit in RAM, so storage barely matters. Once data outgrows memory and the drives fill up, mdraid degrades and ADP holds up: the real production scenario.

Up to 72% more transactions on a full database

+26%

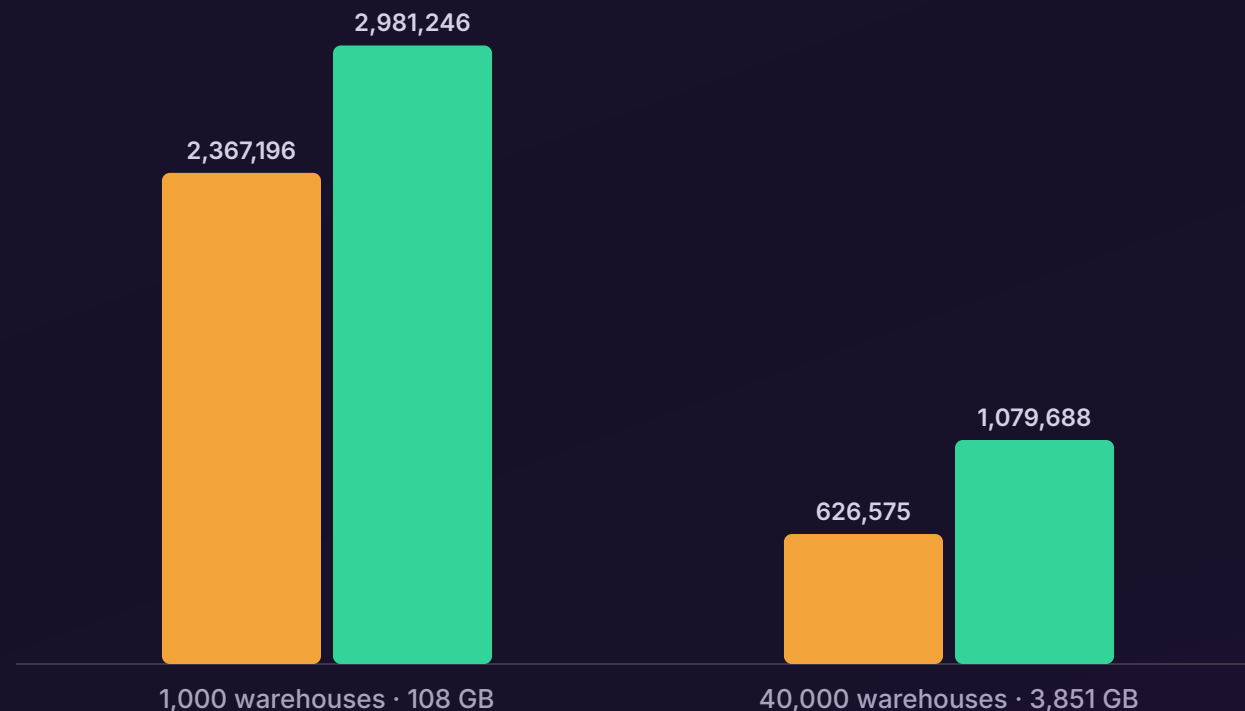
1,000 warehouses · 108 GB
2.98M vs 2.37M TPM

+72%

40,000 warehouses · 3,851 GB
(70% of mdraid capacity)
1.08M vs 0.63M TPM

Transactions per minute (TPM) · higher is better

mdraid RAID10 ADP RAID5

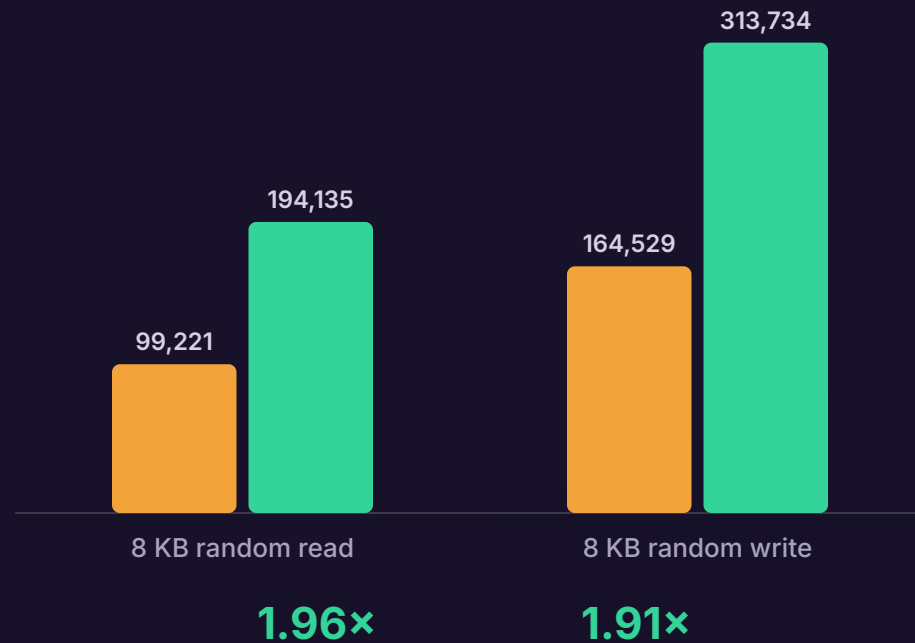


Same server, same PostgreSQL configuration, same four drives.

PostgreSQL on KV: ~2× I/O. pgvector: 9.5× faster builds.

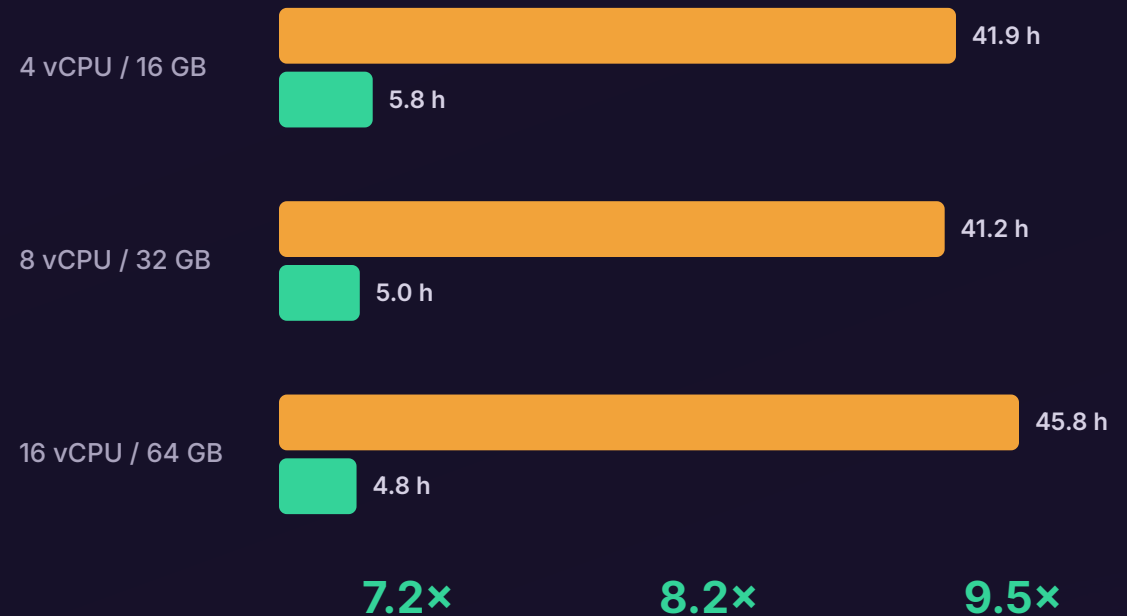
PostgreSQL storage I/O, ops/s

POSIX files ADP KV



HNSW index build, hours · lower is better

pgvector 0.8.0 ADP pgvector 0.4.0



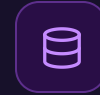
pgvector: dataset equals RAM in each configuration; Cohere 768-d, HNSW m=16, ef_construction=200, ef_search=120, same recall. Up to 3.6× higher query throughput once data exceeds RAM, no penalty when it fits.

Each result maps to a mechanism



6.4× random writes ← Write big and sequential

Random 8 KB writes are aggregated into large stripes, so four drives do sequential work instead of random work.



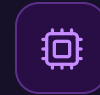
2× capacity ← Compression and parity instead of mirrors

RAID5 keeps 75% of raw capacity; inline compression stretches it further.



Lower tail latency ← Full-stripe writes, no read-modify-write

Parity is computed on whole stripes in hardware, so writes never wait on reads.



Less host CPU ← RAID and compression offloaded

The card does the work the kernel would do in md, leaving the cores to the database.

Headroom: these runs used four PCIe Gen4 drives on a card rated for 48 drives and 56 GB/s. The ceilings on these slides are the drives', not the card's.

Faster storage, not more servers.



Write-heavy workloads, unblocked

Up to 6.4× random-write throughput and up to 72% more database transactions on the same drives.



Twice the capacity of RAID10

Parity protection and inline compression instead of mirroring every block.



Drop-in deployment

A block device for the software you run today, or KV mode for engines built on it.