

ECE566
Enterprise Storage Architecture
Spring 2025

Workload profiling and sizing

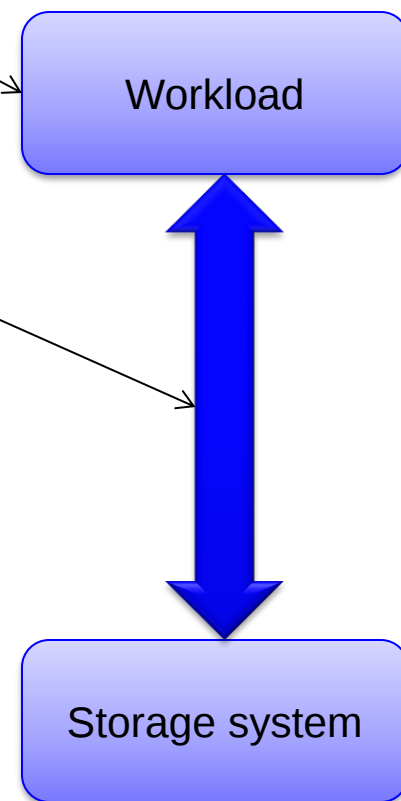
Tyler Bletsch
Duke University

The problem

- **Workload characterization:** Determining the IO pattern of an application (or suite of applications)
 - We do so by measuring it, known as **workload profiling**
- **Storage sizing:** Determining how much hardware you need to serve a given application (or suite of applications)
- The challenge of characterization and sizing
 - Storage is a complex system!
 - Danger: high penalty for underestimating needs...

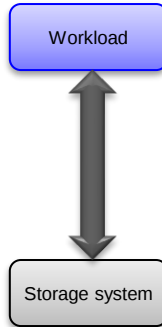
Two kinds of metrics

- Inherent **access pattern metrics**:
 - Based on the code
- Resulting **performance metrics**:
 - The performance observed when those access patterns hit the storage system
- Sometimes difficult to separate:
 - Common one that's hard to tell: *IOPS*
 - Did we see 50 IOPS because the workload only made that many requests, or because the storage system could only respond that fast?
 - Was storage system mostly idle? Then IOPS was limited by workload.



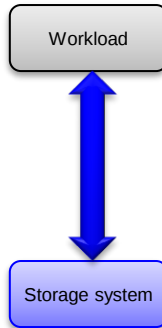
Access pattern metrics

- Random vs. sequential IO
 - Often expressed as random%
 - Alternatives: average distance, seek distance histogram, etc.
- IO size
- IOPS
 - If controller/disk utilization was low, then IOPS represent storage *demand* (the rate the app asked for)
 - Alternative metric: inter-arrival time (average, histogram, etc.)
- Reads vs. writes
 - Often expressed as read%
 - May also split all of the above by read vs. write (read access pattern often different from write pattern)
- Breaking down application: can we identify separate threads?
 - Is it 50% random, or is there one 100% random thread and one 100% sequential thread?

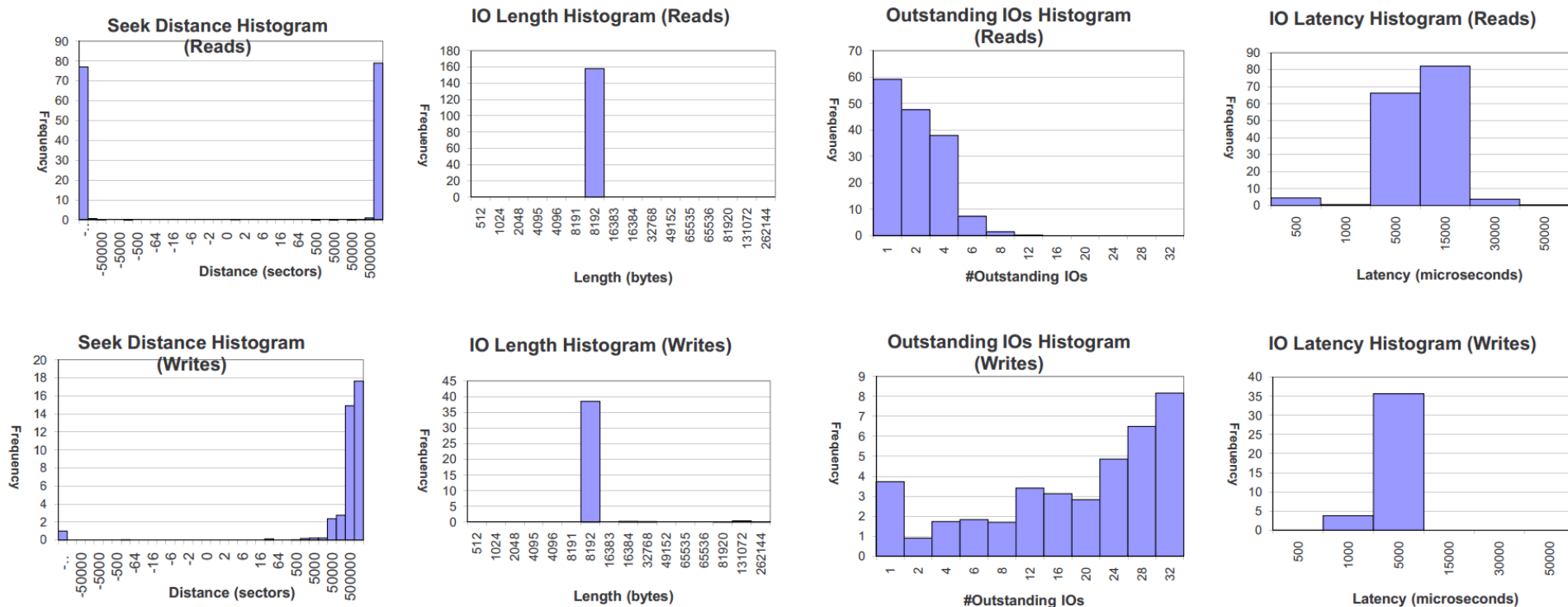


Performance metrics

- IOPS (if storage system was bottleneck)
 - Alternative metric: IO latency (average, histogram, etc.)
 - Alternative metric: throughput (for sequential workloads)
- Queue length: number of IO operations outstanding at a time
 - A measure of IO parallelism



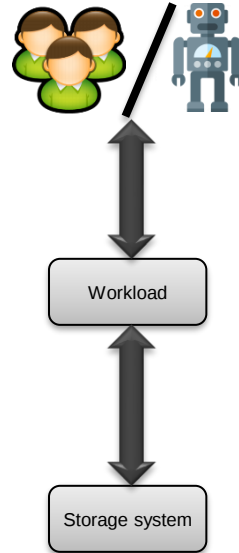
Example of metrics



- Metrics for “DVDStore”, a web store benchmark.
 - Random workload (seek distance $\neq 0$)
 - IO size = 8k
 - Short read queue, long write queue
 - Reasonable latency (within usual seek time)
 - Seek distance for writes is biased positive (likely due to asynchronous write flushing doing writes in positive order to minimize write seek distance)

How to get these metrics?

- **Profiling:** *Run* the workload and *measure*
- Two problems:
 1. How to “run”?
 - Most workloads interact with users
 - Need user behavior to get realistic access pattern!
 - Where to get users?
 - App already in production? Use actual users
 - If not, fake it: **synthetic load generation** (extra program pretends to be users)
 - What about so-called **benchmarks**?
 2. How to “measure”? We’ll see in a bit...

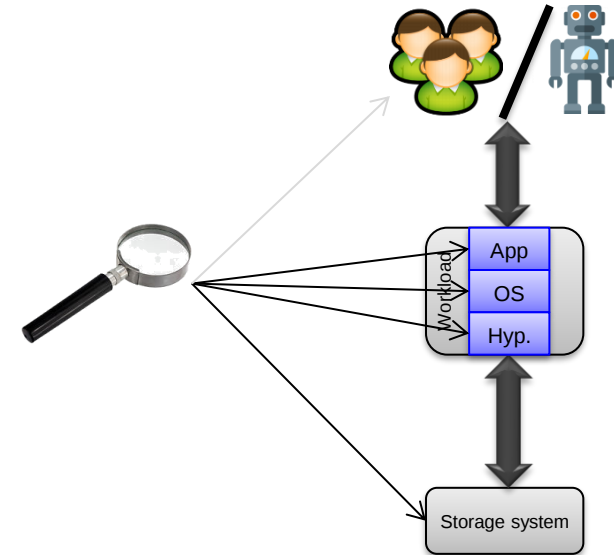


Benchmarks

- **Benchmark:** a program used to generate load in order to measure resulting performance. Various types:
 - **The application itself:** You literally run the real app with a synthetic load generator.
 - Example: Microsoft Exchange plus LoadGen
 - **Application-equivalent:** Implements a realistic task from scratch, often with synthetic load generation built in.
 - Example: DVDStore, an Oracle benchmark that literally implements a web-based DVD store.
 - **Task simulator:** Generate an access pattern commonly associated with a certain *type* of workload
 - Example: Swingbench DSS, which generates database requests consistent with computing long-running reports
 - **Synthetic benchmark:** Generate a mix of load with a specific pattern
 - Example: IOZone, which runs a block device at a given random%, read%, IO size, etc.

Methods of profiling

- App instrumentation
 - Requires code changes
- Kernel instrumentation
 - Can use kernel performance counters (e.g. iostat)
 - Can hook at system call level (e.g. strace) or block IO level (e.g. blktrace).
 - Can also do arbitrary kernel instrumentation, hook anything (e.g., systemtap)
- Hypervisor instrumentation
 - Hypervisor sees all I/O by definition
 - Example: vscsiStats in VMware ESX
- Storage controller instrumentation
 - Use built-in performance counters
 - Basically this is kernel instrumentation on the storage controller kernel
- User-level metrics (e.g. latency to load an email)
 - These don't directly help understand storage performance, but they *are* the metrics that users actually care about



Sizing

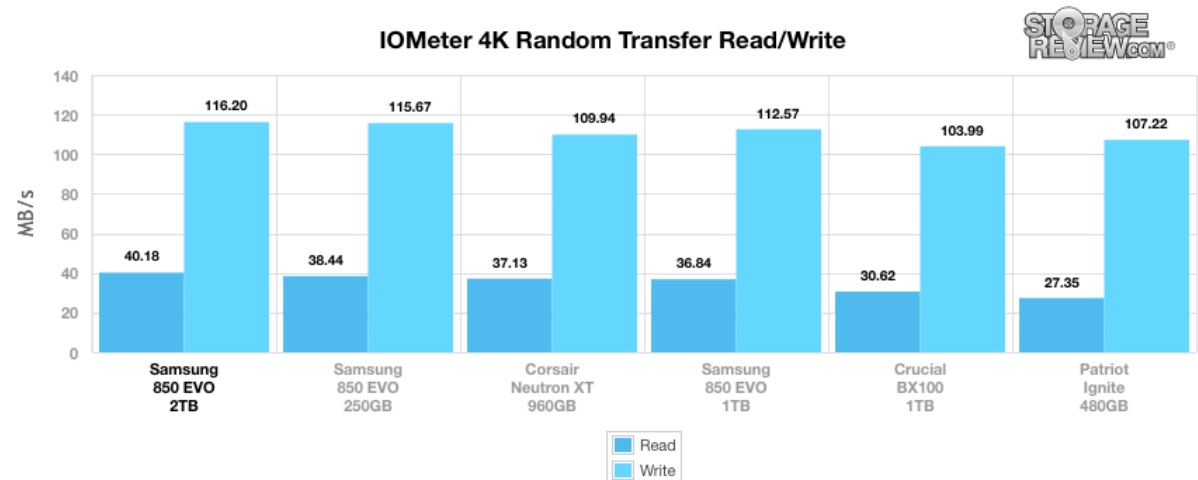
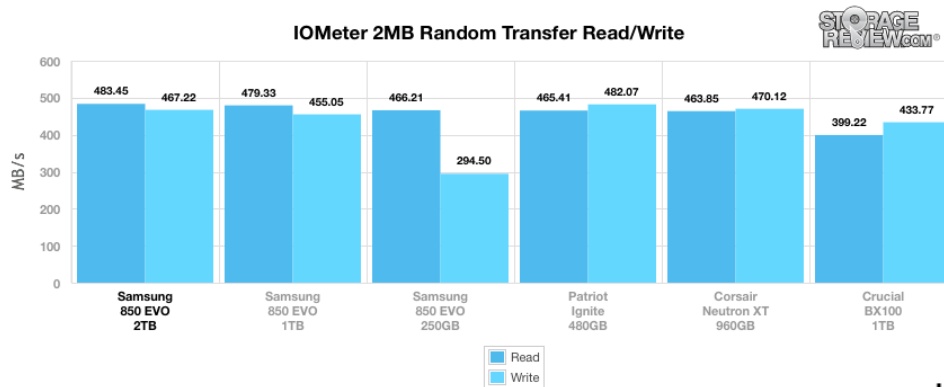
- Now we know how workload acts; need to decide how much storage gear we need to buy
- Will present basic rules, but there are complicating factors:
 - Effects of storage efficiency features?
 - Effects of various caches?
 - CPU needs of the storage controller?
 - Result when multiple workloads are combined on one system?
- Real-world sizing of enterprise workloads:
 - For commercial apps, ask the vendor – companies with big, expensive, scalable apps have sizing teams that write sizing guides, tools, etc.
 - On the storage system side, ask the system vendor – companies with big, expensive, scalable storage systems have sizing teams too.

Disk array sizing

- Recall: In a RAID array, performance is proportional to number of disks; this includes IOPS
- Each disk “provides” some IOPS: $IOPS_{disk}$
- Our workload profile tells us: $IOPS_{workload}$
- Compute $\frac{IOPS_{workload}}{IOPS_{disk}}$: get number of disks needed
- Add overhead: RAID parity disks, hot spares, etc.
- Add safety margin: 20% minimum, >50% if active/active
- Note: this works for SSDs too, $IOPS_{disk}$ is just way bigger

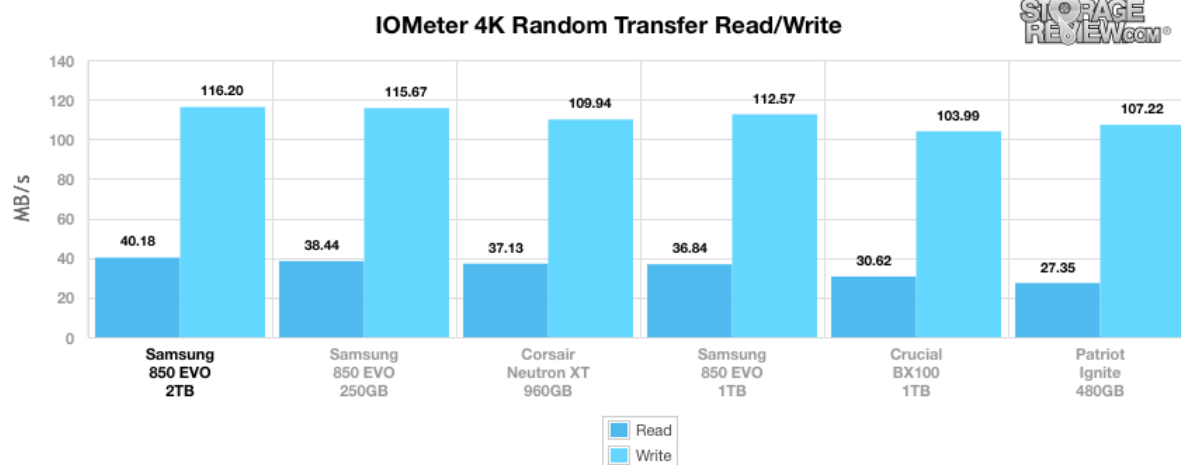
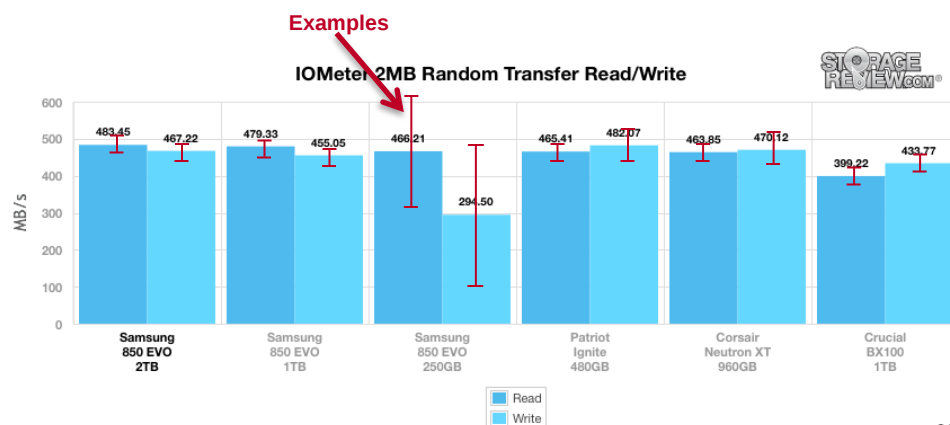
Characterizing disks

- Use synthetic benchmark to find performance in the extremes (100% read, 100% write, 100% seq, 100% random, etc.)
- You did this on HW1...results for Samsung 850 Evo 2TB SSD:



Statistical side-note

- What are these graphs missing?
 - **Error bars!**
 - What is the variance in these tests? Are the results significant??



Combining workloads

- Rare to have one storage system handle just ONE workload; shared storage on the rise
- Can we simply add workload demands together?
 - Sometimes...it's complicated.
 - Example that **works**: two random workloads run on separate 3-disk RAID5s will get similar performance running together one 6-disk RAID5
 - Example that **doesn't**: a random workload plus a sequential workload wrecks performance of the sequential workload
 - Random IOs will "interrupt" big sequential reads that would otherwise be combined by OS/controller.

Workload combining

RAID5 config

Workload	LUN configuration	IOPS	IO latency	Application Metric
Random → DVDStore	2+1	130	100	6132 TPM
Random → OLTP	2+1	141	30	5723 TPM
DVDStore (Shared)	5+1	144 😐	30	7630 TPM
OLTP (Shared)	5+1	135 😐	30	5718 TPM

Table 1. Comparison of DVDStore and OLTP when run in isolation and shared mode

Workload	LUN configuration	Throughput	95% tile latency	Application Metric
Random → DVDStore	2+1	130 IOPS	100	6132 TPM
Sequential → DSS	2+1	44 MB/s	30	6 completed transactions
DVDStore (Shared)	5+1	164 IOPS 😊 +26%	15	7630 TPM
DSS (Shared)	5+1	31 MB/s 😞 -29%	1	3 completed transactions

Table 2. Comparison of DVDStore and DSS when run in isolation and shared mode

- “OLTP” = “Online Transaction Processing” (normal user-activity-driven database)
- “DSS” = “Decision Support System” (long-running report on a database)
- Table 2: DVDStore benefits from twice as many disks to help with latency, but DSS’s sequential IO gets wrecked by the random interruptions to its stream

Effects of aging on performance

- A storage system can get worse over time due to **aging effects** – changes in storage layout over time as data is added/removed.
- Can complicate testing and require much longer tests
- Can come from on-disk data structures (filesystem) and the device itself (especially for SSD)...

Sources of aging effects

- The **filesystem** makes placement decisions.

- Example: write of blocks A,B,C,D,E,F,G:



- Later, we delete B,D,F and write X,Y:



- This is classic external fragmentation
- Also occurs in metadata of filesystem (e.g. inode tables)
- An **SSD** will indirect the location of objects
 - An SSD never actually overwrites a page! Always writes elsewhere, compacts later, etc.
 - Effects most pronounced when SSD is near full; controller has fewest choices it can make to place new data!
 - Side note: "Full" here means "not TRIM'd"
 - Can mean slower writes, worse internal cache effects on reads
 - Worst case: if SSD has to block a write to do an erase cycle (~3ms!)

Dealing with aging in benchmarking

- Result: Have to subject storage system to **realistic** workload for a sufficient number of IOs to get long-term measurements
 - Tests might have to run for days/weeks/months...
- Test realism is important here:
 - Real workloads typically follow various 80/20 rules (e.g., 80% of writes to 20% of locations)
 - Therefore, naïve synthetic workloads will be either hugely optimistic (repeated full-device sequential write) or hugely pessimistic (repeated full-device random write)
 - Accuracy is achieved with a test workload that mirrors the intended use case

Conclusion

- To **characterize** a workload, we must **profile** it
 - Run it (generating user input if needed)
 - Measure IO metrics in app/kernel/hypervisor/controller
- Can use workload profile for **sizing**:
to identify storage gear needed
 - Basic rule: provision enough disks for the IOPS you need
 - Past that, look for published guidance from software/hardware vendor
 - Failing that, use successive experiments with differing gear to identify performance trends