• **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 ≠ 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 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: $IOPSDisk$
- Our workload profile tells us: $IOPSWorkload$
- Compute $\frac{IOPSWorkload}{IOPSDisk}$: 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, $IOPSDisk$ 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:

From http://www.storagereview.com/samsung_850_evo_ssd_2tb_review
What are these graphs missing?

**Error bars!**

What is the variance in these tests? Are the results significant??

From http://www.storagereview.com/samsung_850_evo_ssd_2tb_review
Interpolation-based sizing

- For large/complex storage deployments with mixed workloads, simple IOPS math may break down
- Alternative: **measurement with interpolation**
  - Beforehand:
    - foreach (synthetic benchmark configuration with access pattern $a$)
      - foreach (storage system configuration $s$)
        - set up storage $s$, generate IO pattern $a$, record metrics as $M[a,s]$
    - Later, given real workload with access pattern $a_{\text{given}}$ and performance requirements $M_{\text{required}}$
    - Find points $a,s$ in table where $a$ is near $a_{\text{given}}$ and performance $M[a,s] > M_{\text{required}}$
    - Deploy a storage system based on the constellation of corresponding $s$ values.
      - Can interpolate storage configurations $s$ (with risk)
      - Pessimistic model: Can just pick from systems where $a$ was clearly “tougher” and performance $M$ was still “sufficient”
- Why do all this? Because $s$ can include ALL storage config parameters (storage efficiency, cache, config choices, etc.)
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 RAID will get similar performance running together one 6-disk RAID

  • 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

- "OLTP" = "Online Transaction Processing" (normal user-activity-driven database)
- "DSS" = "Decision Support System" (long-running report on a database)

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

<table>
<thead>
<tr>
<th>Workload</th>
<th>LUN configuration</th>
<th>Throughput</th>
<th>95% tile latency</th>
<th>Application Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>DVDStore</td>
<td>2+1</td>
<td>130 IOPS</td>
<td>100</td>
<td>6132 TPM</td>
</tr>
<tr>
<td>OLTP</td>
<td>2+1</td>
<td>141 IOPS</td>
<td>30</td>
<td>5723 TPM</td>
</tr>
<tr>
<td>DVDStore (Shared)</td>
<td>5+1</td>
<td>144 IOPS</td>
<td>30</td>
<td>7630 TPM</td>
</tr>
<tr>
<td>OLTP (Shared)</td>
<td>5+1</td>
<td>135 IOPS</td>
<td>30</td>
<td>5718 TPM</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Workload</th>
<th>LUN configuration</th>
<th>Throughput</th>
<th>95% tile latency</th>
<th>Application Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>DVDStore</td>
<td>2+1</td>
<td>130 IOPS</td>
<td>100</td>
<td>6132 TPM</td>
</tr>
<tr>
<td>DSS</td>
<td>2+1</td>
<td>44 MB/s</td>
<td>30</td>
<td>6 completed transactions</td>
</tr>
<tr>
<td>DVDStore (Shared)</td>
<td>5+1</td>
<td>164 IOPS</td>
<td>15</td>
<td>7630 TPM</td>
</tr>
<tr>
<td>DSS (Shared)</td>
<td>5+1</td>
<td>31 MB/s</td>
<td>1</td>
<td>3 completed transactions</td>
</tr>
</tbody>
</table>

- 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.
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