Experience Report

Prepared for the 2nd Workshop on Performance and Reliability (WOPR)

Background

In 2003, I was a senior test engineer at LeftHand Networks, in Boulder, Colorado. LeftHand makes an IP-based SAN (Storage Area Network) product - instead of connecting disk drives directly to a server, the drives live in a dedicated storage appliance, and device drivers on the server access the disk drives over an IP network. LeftHand's product includes various enhancements related to storage virtualization (logical separate from physical), snapshots (instant backups), and management.

When I joined the performance improvement effort, my goal was to design and build a testbed and suite of tests that could be used to measure end-to-end throughput repeatably. Those measurements could then be used to steer the development efforts to improve system performance.

Experiences

The testbed consisted of 4 PCs running Windows 2003 Server, 6 LHN storage appliances (Linux-based, so I could ssh to them and run all the usual monitoring tools), and a gigabit ethernet switch. I also used another PC to access the testbed components over the control network.

I chose IOMeter, which is an I/O performance measurement tool developed at Intel then released as open source <http://www.iometer.org/>, as the main measurement tool for this testbed. It could use both Windows and Linux as data sources and sinks, and it was the tool used by many of the publications that review LHN's type of product.

Both to learn IOMeter and to measure a basic system characteristic, I measured max network throughput (1-to-1, 1-to-many, many-to-1, and many-to-many) using both the windows PCs and the LHN appliances as endpoints. I also used IPerf (NLANR tool, <http://dast.nlanr.net/Projects/Iperf/>) to check IOMeter's numbers.

The next step was to use IOMeter to recreate the numbers produced by in-house performance measurements tools. The measurement points were on the appliance at the disk device level, on the appliance at the stripe driver level, and on the Windows PC at the raw disk level. I could not get IOMeter to access the devices on the appliance, but I still needed to calibrate IOMeter against the in-house tools. So I took a single drive out of the appliance, hooked it up to a windows box, and compared IOMeter results from that config to the in-house tool running against the Linux disk device.

At this point in time, I was assigned two short-term tasks. We discovered that the RAID/stripe driver code that handled disk failure and replacement was not MP-safe. I was to provide a rough measurement of the effect of turning off hyper-threading on the system, so a decision could be made on how to proceed (fix code now, or take the performance hit of turning off MP now, then fix it later).

I was also asked to provide marketing with a comparison of the performance of our existing appliance with the performance of our new appliance. The product manager needed some numbers to give to the sales force.

The Model Breaks Down

Before I started this performance testing effort, I was part of the system test group. I had built up a mental model of the system, and used that in my initial testbed and test suite design. I did not realize the ways in which my model was incomplete and incorrect.

My manager's mental model of the system was also incomplete and incorrect, but in different ways. He joined the company a few months after I did, and he had many years of experience in the storage industry. His model was influenced heavily by the architecture of other products he had worked on during his career.

I relied on my manager for guidance in designing meaningful tests (valid points in the performance curve, and useful for marketing). Most of the time, his suggestions worked out well. Sometimes I didn't think his suggestions were on the right track, but after some investigation, I discovered my model was wrong and his was right, and the suggestions were valid.

Problems arose when he made suggestions based on inaccuracies in his model. For example, he asked me to measure transfer rates for 100% cache hit workloads and 100% cache miss workloads. I was fairly sure the product did not implement a cache (read or write), so I told him I didn't know how to do that given the fact that there was no cache. He did not update his model until after a few iterations of such requests.

The areas of incompleteness of our models were also sources of complications. Initial test results contained a lot of anomalous data. As I investigated, I realized the system I was testing (including the test tools) was quite complex, and measuring performance in a meaningful and repeatable manner was a hard problem.

Some of the complexity was due to the instability of the new NVRAM card. Since the new card increased performance, we decided to run the tests with it in the system. I did lag the functional testing a bit, but I tripped over many problems that were not caught by those earlier tests.

An example was an instance where one of the read transfer rates was around 70% of the expected value. Other rates in that same test run were just fine. After significant investigation, I discovered that read performance was degraded if the test ran immediately after a write performance test. The NVRAM copy-to-disk algorithm was aggressively opportunistic when there was no write activity, so the system was draining the cache during the read test. I reordered the tests (IOMeter did not support an initial delay parameter - ramp-up time was not enough), and the value returned to the expected range.

My tests were long-running to begin with, both in setup time (8-12 hours setup was about average) and in test duration (~20 hours in some cases). The bugs in the NVRAM card invalidated a number of test runs and prevented me from making some optimizations in test scheduling (I couldn't re-use the same setup time across multiple test runs).

The pace of testing was not all that compatible with the culture of this particular high-tech startup. Although I did spend a lot of time contemplating the meaning of the results, and building my understanding of the system's behavior, I was also asked to pick up a number of functional and system testing tasks. Therefore, the time I spent tracking down the cache draining issue ate up all the time I had allocated for analyzing the performance data. Many test runs were reported raw because of time spent on this type of investigation.

Going back to the two performance tasks I needed to complete.

I managed to get some numbers comparing MP and non-MP performance, mostly due to the facts that this was on the existing product (which was slow enough to max out before the client did), and the lack of NVRAM card in that product.

The second task proved more interesting. I started comparing the two systems using performance measurement at an internal test point (local proprietary protocol, which was above the RAID/stripe driver but below the network aspect of that protocol). This was a good start, but there were anomalies even in that relatively simple measurement.

By the time those numbers were presented to marketing and the executive staff, I was asked to stop testing the old product. But my testing of the old product, and comparing those results to previous performance analyses, helped me identify gaps and bugs in my model. I'm not sure how much more I could have learned from continued testing of that system, but I felt there was value in continued study.

Presentation of the Results

At this time, I believed I understood the system well enough to publish my results within the development organization. The feedback was more intense than I had expected.

Some of it was valid, based on obscure (and not so obscure but missed by me) aspects of the system. One example was the Win2003 setting choosing between file server and application server. Our system ran faster and more consistently when the server was set to be an application server. I recall the reason having to do with buffer allocation to local disk (which we did not use) and buffer allocation to network activity (which we did a great deal).

Most of the feedback was related to non-technical issues. Results from my tests were like a lightning rod for the political squabbles in the company's culture. Everyone had an opinion on why the numbers were good or bad, and many championed their own plans for change (or lack of change) based on my results.

Lessons Learned

One aspect of my approach is to let simple tests increase my understanding of the system, only progressing on to more complex tests when I understood the previous set. I balanced this against getting results in a timely manner. In this particular case, I think the pressure to produce quickly, coupled with reduced time for analysis, left me too far towards the "get numbers quick" camp.

When using a new tool, calibrate its results against a known tool. I was glad to have the comparison data between IOMeter and the internal test tool, because "attacking the tool" was a common tactic in the discussion of my results.

If all your measurements except one take a long time, people will wonder why all the others are slow instead of wondering why that one is fast.

There is no escape from Marketing.

Whenever possible, I should let someone else point out the flaws in my boss' understanding.

Everyone has an explanation for everything. If the organization is not used to the process of discussing such things, expect to spend a lot of time in meetings.

This was a complex product, put together by a team of smart, passionate people. Performance, or "how well does this product work" rather than "does this product work", is an area in which to tread lightly.

Appendix A

Some of the basics of I/O throughput testing, as I learned them, are:

- Most drives have on-board cache memory, and will reorder I/O to optimize throughput.
- Most drives will read an entire track at a time.
- The data transfer rate at the outer diameter can be twice as fast as the transfer rate at the inner diameter.
- Reads are faster than writes, even if write-then-verify is turned off.
- The industry-standard practice for drive performance measurement is to measure throughput rates for large transfers accessed sequentially, and small transfers accessed randomly. The large sequential transfers are run across the entire disk, to balance out the difference between ID and OD rates. The small random transfers are run long enough to obtain a representative sample of seeks.
- I ran a complete set of transfer tests on a regular basis, looking for new anomalies. These tests stepped through number of threads (1-64 exponential), transfer size (512-256K, exp), read/write, and sequential/random. This took a long time.

Appendix B

LeftHand's appliance threw in a few wrinkles:

- Logical volumes, which appeared to the servers to be physical disk drives, could span multiple drives within an appliance, and could also span multiple appliances. Volumes could also be mirrored, both within and across appliances.
- Initially, there was no write-cache in the system (the on-disk write cache was turned off). Once the queue was full, write performance fell off sharply as RTT was factored in to each write. Later models included a NVRAM card acting as a write cache.
- There was no read cache in the system.

- Logical blocks (LBAs) were mapped to physical disk blocks in the order the LBAs were written. While the distribution across physical disks and appliances was deterministic (striped round-robin, stripe size 128KB), the allocation of that stripe on a particular disk drive was random. The end result was that the testbed was never exactly the same twice, and sequential was never really sequential.

Appendix C

IOMeter idiosyncrasies

- IOMeter could be configured to run for a certain length of time, but could not be configured to run for a certain total of data transferred. I had to estimate how long it would take to traverse then entire volume based on some initial quick measurements, then plug that time in to the run and hope for the best.
- IOMeter does not report interim results. In order to see performance over time, I had to run perfmon and save the output. This was particularly important for one degenerate load which caused short burst of very high throughput mixed in with longer periods of zero throughput. The time average was within the predictions of the models.
- IOMeter can't execute scripts between runs, but it can run in a batch mode.

Appendix D

Tests I ran

Whenever possible, I stepped through number of threads (1-64 exponential) and transfer size (512byte-128KB exponential). With both reads and writes (all random - none sequential), this gave me 126 data points.

Raw Disk: I as not able to use the IOMeter worker on Linux to access the drive directly; however, I had a custom tool that could do so. I compared these numbers to IOMeter running against a directly-attached drive on a client, and to vendor performance stats.

Numbers were slightly higher than my manager expected, but slightly lower than vendor benchmarks.

Striping layer: On the storage appliance, I ran the custom tool against the stripe driver. LeftHand's storage management software ran above this layer, and I was able to measure the overhead of the striping driver against the theoretical aggregate performance of the individual drives. Overhead was small.

One interesting aspect of LeftHand's storage appliances (at the time of my testing) is that physical disk blocks are not allocated until they are needed, and the algorithm for block allocation is only semi-deterministic with respect to physical location of blocks. I/O to an unallocated block incurs significant overhead. The approach I used was to "pre-write" the logical drive using a workload specifically designed to take advantage of the allocation algorithm. This added many hours to the testbed setup time whenever the logical drive configuration could not be preserved.

Another interesting aspect (again, at the time of my testing) is that the drives in the appliance are all used both for data and for metadata, so an increase of metadata activity (i.e. logging) will negatively affect performance numbers. This limited the amount of data collection I could easily perform on-box. I ended up scripting a logged remote shell (with timestamps) running vmstat.

LeftHand storage layer: Again, run on the storage appliance using the custom tool. The tool and the storage software initially competed for CPU time. Tuning the tool code and turning off various subsystems reduced this contention.

LeftHand network protocol: I used a tool that simulated a remote client, but ran on the storage appliance. Much processing takes place at this network layer - knowing the overhead without the constraint of the network stack was helpful. I was not able to reduce the CPU contention to trust the numbers in an absolute sense, but I did want to know where these numbers fit in the overall performance scheme.

Remote client: I ran the tool on the client machine, bypassing the device driver. Performance was significantly lower, and the Win32 Network stack was blamed. I did not attempt to “fix” the code (of the tool) for this platform.

Remote device driver: I ran IOMeter on the client, and monitored CPU utilization on both client and storage appliance.

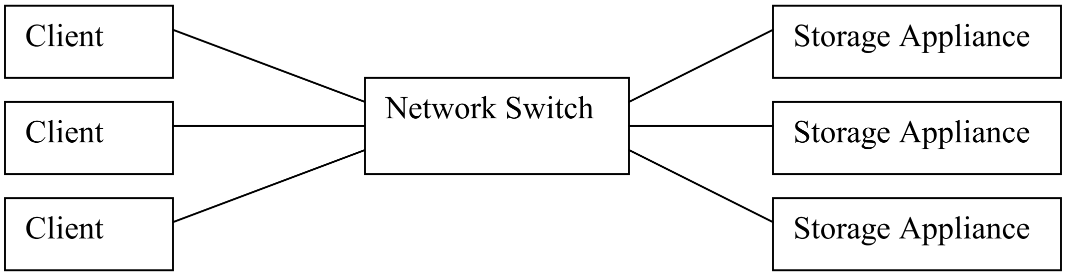
One question was how long to run each workload.

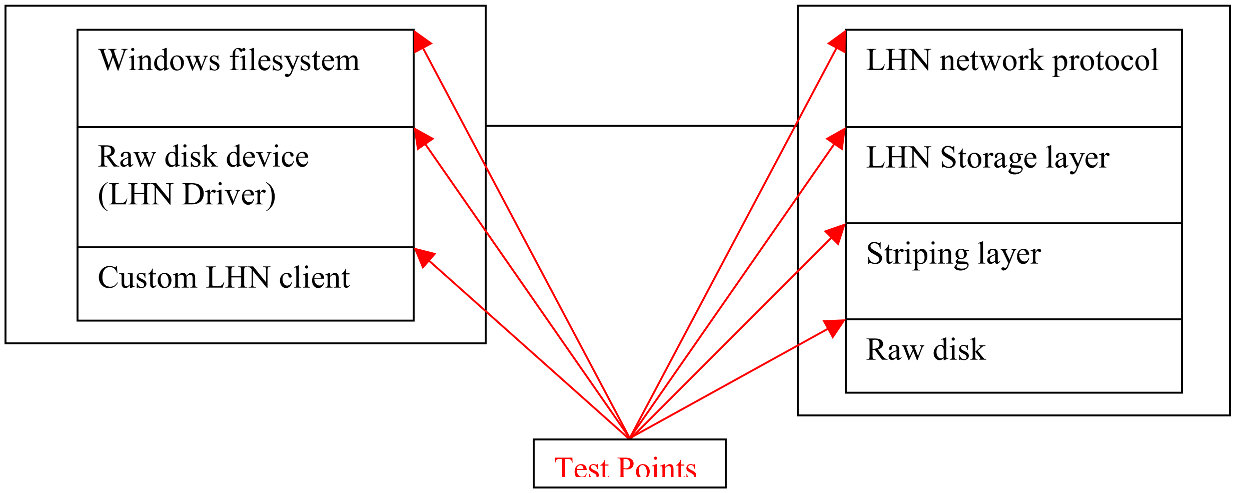
I ran the random workloads 10 times each for different durations to help determine an appropriate duration for actual test runs. We decided on 20 minute runs for random workloads.

One of IOMeter’s limitations is that it can only limit a workload run time by clock time, rather than by bytes transferred. So I could not configure a workload to read the entire disk once; instead, I had to guess how long such a run would take, then use that to configure the time limit for that workload. (Later, I realized that sequential transfer workloads were equivalent to random transfer workloads where the random location was constrained to stripe boundaries. But the numbers still looked good.)

Appendix E

Testbed and architecture diagrams:


Client
Client
Client
Network Switch
Storage Appliance
Storage Appliance
Storage Appliance


Windows filesystem
Raw disk device (LHN Driver)
Custom LHN client
LHN network protocol
LHN Storage layer
Striping layer
Raw disk
Test Points