Performance metrics and benchmarking of an OpenSolaris^{TM} NFS fileserver

SNE MSc Research Project

A. van Hoof (alain.vanhoof@os3.nl)

February 2010

Abstract

When benchmarking an OpenSolaris^{TM} NFS fileserver with benchmark software running on the NFS client(s) a number of layers of the Operating System on the client and server are traversed. It is useful to follow the data in the various layers, for example to find performance bottle-necks. The Dtrace tool of OpenSolaris^{TM} allows for a nonintrusive way to observe read and write to the different layers of the OpenSolaris^{TM} operating system. This document not only describes a method to benchmark a NFS fileserver using standard benchmark tools it also describes a method to monitor the behavior of the NFS fileserver.

Preface

This report was created as part of a four week research project done as part of the System- and Network-engineering master at the University of Amsterdam[1]. The research was done at the UvA-IC T.S. (Universiteit van Amsterdam Informatiserings Centrum, Technical Support) location at Science Park Amsterdam. I would like to thank the department T.S. for providing resources and time for this research project. A special thanks to Jeroen Roodhardt and Auke Folkerts whose guidance and explanations gave me the insights needed to do this project.

1 Introduction

To facilitate storage for students and employees of the UvA, a combination of NFS and CIFS is used. The SUN^{TM} storage server uses the OpenSolaris^{TM} operating system, the clients are Linux Desktops. In this setup, performance issues have been identified. To identify/observe these performance bottlenecks various tools and methods are used. Identification of the issues can be done by observing current behavior and compare it to normal behavior.

Normal behavior can be defined using load simulations in a test environment and creating a base-line of "normal system behavior". Generating a correct base-line creates the need for a representative workload for the load simulations.

Research Question

This led to the following research question used for the research in this report:

> *How can the performance bottlenecks be monitored and identified on an OpenSolaris^{TM} OS NFS fileserver. What are realistic load simulations and create a base-line.*

[1] http://www.os3.nl

2 NFS Fileserver Performance

To be able to determine performance, benchmarking is done to find performance values[4]. These values can be compared to other related performance tests, to determine if there are any performance issues. Repetition of the benchmark while changing settings on the benchmarked system can be used to improve performance. But what settings should be changed and when there is no performance change, why not? These questions most of the time can not be answered by just looking at the benchmark values. Inspection of the benchmarked system itself is needed to find bottle-necks or other constrains. It is therefore very interesting to examine/inspect the various layers in the system when the benchmarks are performed. This without influencing the outcome of the benchmark. The goal of the benchmarking tests performed in this research is to create a baseline of performance indicators for normal behavior of the benchmarked system and to get benchmark values.

But what is normal behavior/operation? When a fileserver is not serving any files or when it is serving files at its maximum rate, both are normal operation if the server is responding and performing as expected. This expected behavior is a baseline for the performance indicators. Generating various fileserver loads using micro and macro benchmarks as defined in section 3 and observing and logging the performance indicators will create a baseline. The I/O events in the NFS fileserver are linked and therefore the performance indicators will show a correlation that is unique for the baseline.

2.1 The OpenSolaris™ NFS fileserver and ZFS

The OpenSolaris™ operating system uses the ZFS[2,3] filesystem. ZFS includes, among a lot of other features, a volume manager. The NFS fileserver hardware (Appendix A) includes 48 hard disks which are managed by ZFS. ZFS includes a ZIL[4] which is used during synchronous writes operations and an ARC[5] acting as a read cache. The ZIL can be placed on a fast separate device, an SSD device for example, to improve performance. Because the NFS data writes are synchronous as described in section 2.2 NFS data writes always passes the ZIL. The ARC places its cache in memory of the server and can be extended (level2 ARC) with a fast separate device. Every read on the ZFS filesystem passes the ARC.

2.2 OpenSolaris™ NFS fileserver and Linux NFS Clients

A Linux NFS client and an NFS fileserver using the ZFS filesystem have a number of layers within the OpenSolaris™ Kernel and uses a user-space process to serve I/O on both the server and the NFS client. These layers are visualized in figure 1. A NFS client process sends data to be written to the NFS daemon running on the server via TCP/IP. The NFS daemon on the server send the data to the virtual file system (VFS) layer. The VFS layer passes the data to the ZFS layer where it is written to the physical devices.

In general the communication between the Linux clients and NFS fileserver is synchronous. This is known to lead to performance issues, but currently this can not be changed in the UvA-IC setup. The synchronous settings makes the client wait for every write operation to finish on the server before continuing. The benchmark software is running on a Linux NFS clients and a load on the NFS fileserver is generated by other the Linux NFS clients. This way the performance is measured as experienced by "real-live" users of the NFS fileserver. But when

[2]Zettabyte File System
[3]http://www.sun.com/software/solaris/zfs.jsp
[4]ZFS intent log
[5]Adaptive Read Cache

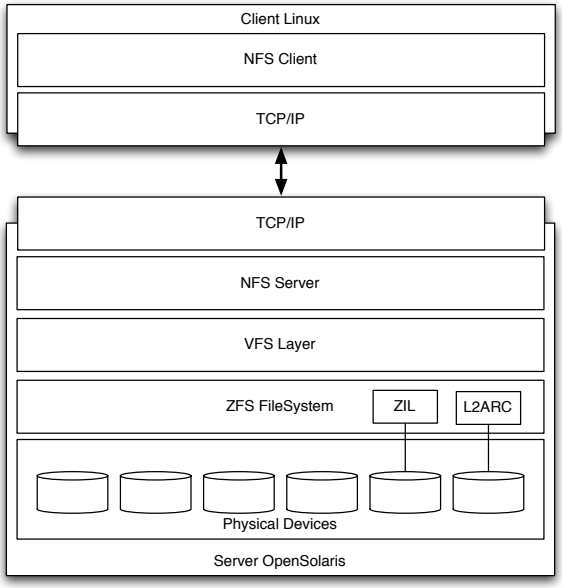

Figure 1: The stack of layers within the Server and Client used by NFS for file I/O

the performance at the clients is not as expected, the NFS fileserver needs further investigation. This investigation can be done using tools available in the OS of the server. Figure 2 relates

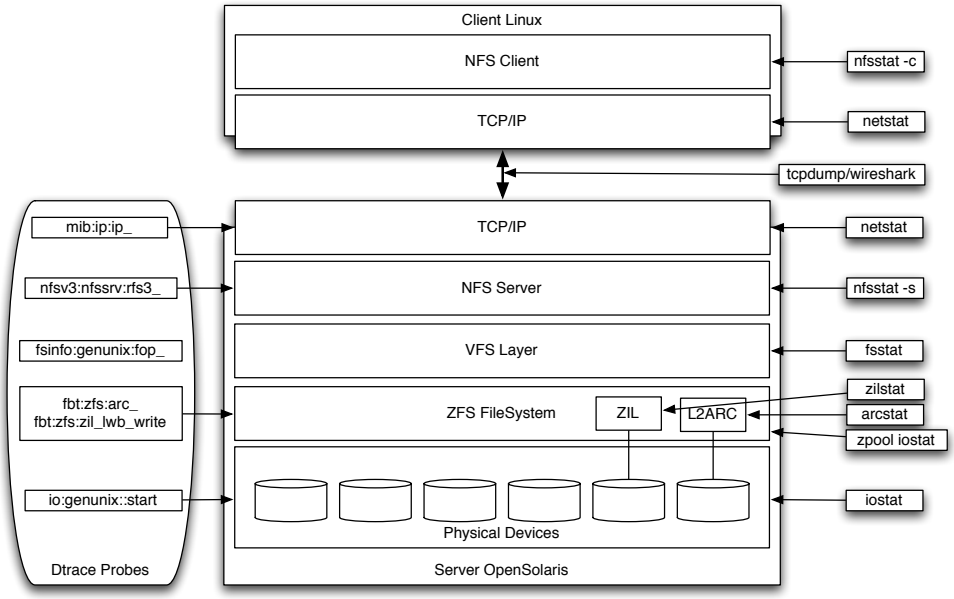

Figure 2: On the right, The software tools to inspect the layers, left the Dtrace probes

these tools to the various layers of the server and the client These tools provide current states and many parameters of the layers. Inspecting all these tools and parameters can be a tedious task. The Dtrace toolkit can be of help to observe all layers and specific parameters on the NFS fileserver.

2.3 The Dtrace toolkit

OpenSolaris^{TM} and Solaris^{TM} kernels and core processes can be probed non intrusively using the Dtrace Toolkit [3] [2]. There are over 84000 probes available (84545 in OpenSolaris^{TM} SVN_129), finding the right one can be difficult. With the use of "Solaris(TM) Performance and Tools: DTrace and MDB Techniques for Solaris 10 and OpenSolaris" by R. McDougall, J. Mauro and B. Gregg [1] and input of the Netherlands OpenSolaris User Group (NLOSUG[6]) probes of interest where identified. For the monitoring of the NFS fileserver 12 probes where chosen each either probing read and/or write actions within the layers. Figure 2 shows the Dtrace probes of interest. The read and write actions within the NFS fileserver and the chosen Dtrace probes for the layers are shown in figure 2.3.

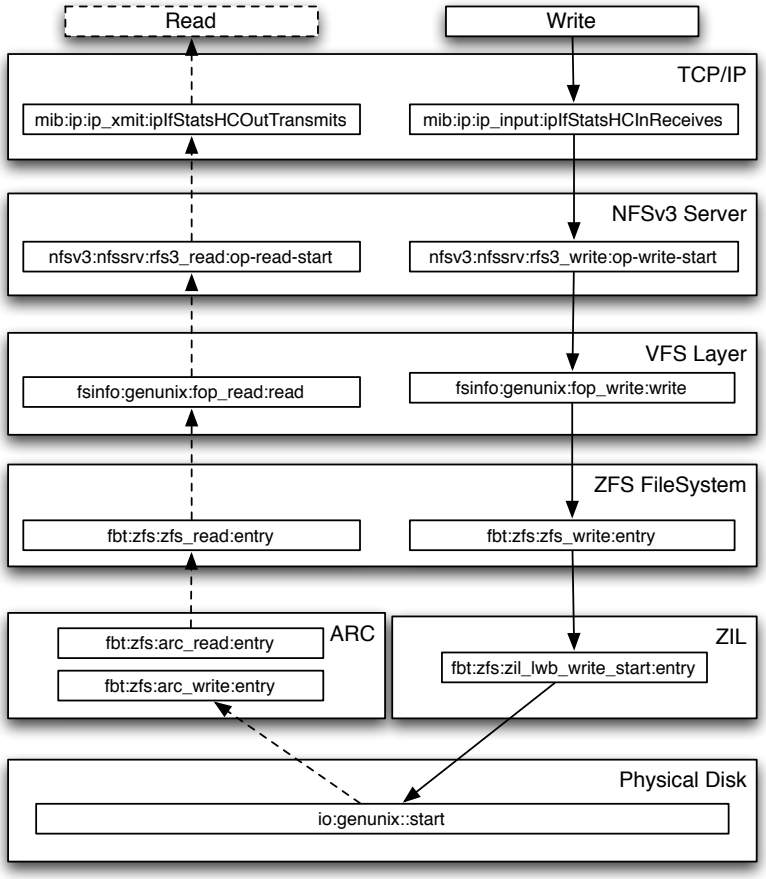

Figure 3: Dtrace probes used in the layers of the OpenSolaris^{TM} NFS file server

[6]http://hub.opensolaris.org/bin/view/User+Group+nlosug/

3 NFS Fileserver Benchmarking and load generation

To measure the performance of the NFS fileserver, benchmark tools are used. Three types of benchmarks tools are summarized by Traeger e.a. [5] in the paper "A nine year study of file system and storage benchmarking":

- *Macro-benchmarks*: The performance is tested against a particular workload that is meant to represent some real-world workload.
- *Trace-based benchmarks*: A program replays operations which were recorded in a real scenario, with the hope that it is representative of real-world workloads.
- *Micro-benchmarks*: A few (typically one or two) operations are tested to isolate their specific overheads within the system.

Because the original issues leading to the research question are in the macro level, the "user experience", emphasis in this paper will be on the macro-benchmark tool. No recordings of real-live scenarios are available so trace-based benchmarks can not be done. A micro-benchmark will be used to benchmark the system specifically for random read and random write performance. This benchmark is of interest for the comparison with other setups and previous benchmarks done by T.S. of UvA-IC.

These benchmark tools run on a NFS client, in order to benchmark the "user experience". Benchmark tools generate load on the system where the benchmark is running. If the benchmark is running on a NFS share mounted on the client, it will also generate a load on the NFS fileserver. Therefore the macro-benchmark tool is not only used to benchmark the NFS client, it is also used to generate a predefined load on the NFS fileserver.

The NFS fileserver will have a cache in normal daily operation and will not reboot very often. The NFS fileserver tested was never rebooted during the experiments. The automation tool made it possible to mount and unmount the NFS filesystem(s) between every run of the benchmarks. Each bench mark was run at least 5 times and at most 30 times,

3.1 Auto-pilot

Auto-pilot [6] is a tool for the automation of tests as advised by the First Annual Storage and File Systems Benchmarking Workshop [4] and "A nine year study of file system and storage benchmarking" [5]. Auto-pilot prevents errors due to command line typo's when doing tests. It can automatically calculate a confidence interval of the executed tests and decide how many tests to run to reach a trustworthy value.

3.2 Filebench

A macro benchmark and micro benchmark made popular by SUN^{TM}and used in many cases, which makes comparison possible. Howerver, one has to be careful when comparing benchmarks, even when the benchmark tool is the same. Filebench provides various workloads to run as the benchmark test.

3.3 IOzone

A micro benchmark and like filebench widely used. UvA-IC T.S. already did performance test using IOzone and is familiar with this benchmark. IOzone can be configured to do only specific tests with specified file and record sizes.

4 Performance testing setup and method

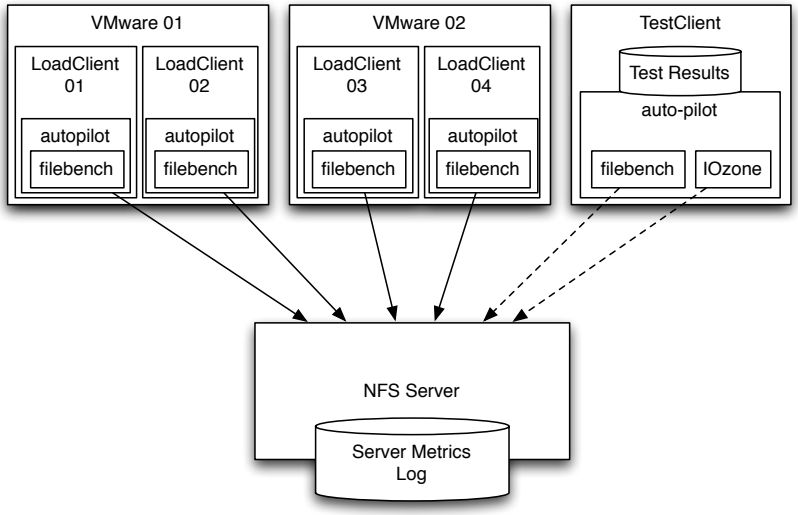

Figure 4: Test Setup

For the benchmarking and generation of the performance indicators the setup of Figure 4 is used. A more detailed description of the hardware and operating systems of the clients and servers is available in Appendix A. The configuration and parameters used of auto-pilot, filebench, IOzone and DTrace are described below.

4.1 Auto-pilot Setup

On each client (Load and Test) the source file auto-pilot-2.4.tar.gz was used to compile and install auto-pilot 2.4 on the local disk of the client, using:

```
./configure prefix=/usr; make; make install
```

Three auto-pilot scripts were used to control the running of filebench and IOzone. One for the load generation on the LoadClients using filebench, one for the performance test on the TestClient using filebench and one for the performance test on the TestClient using IOzone. Auto-pilot has a facility for running of more than one thread of the a test, this is used for running more than one filebench load on the LoadClients.

Auto-pilot assumes that the results are normally distributed, and uses this fact to calculate the appropriate number of runs. The benchmarks first run 5 times. If there is a probability of 95% that the captured mean of the test is the true mean, the test is terminated. They continue up to a maximum number of 30 runs. This number is chosen due to the time restrictions on the tests.

During the running of the tests, auto-pilot assumes that the results are normally distributed, and uses them to calculate the appropriate number of runs. The benchmarks first run 5 times. If is a probability of 95% that the captured mean of the test is the true mean, the performance test is terminated. The tests continue, up to a maximum number of 30 runs. These numbers of runs are chosen due to the time restrictions on the tests.

The following output example shows the values auto-pilot calculates:

```
NAME    COUNT MEAN       MEDIAN  LOW       HIGH      MIN      MAX     SDEV% HW%
opss    5     1224.160   1223    1171.252  1277.068  1156.700 1266    3.481 4.322
mbs     5     29.160     29.200  27.926    30.394    27.600   30.200  3.409 4.232
```

By extending the auto-pilot scripts to add variables for filebench and IOzone, auto-pilot can calculate the statistics using these variables. By default auto-pilot reports the count, mean, median, minimum, maximum, and Student-t confidence interval error bar values (shown as low and high), and the standard deviation and half-width (HW%) of the confidence interval as a percentage of the mean.

4.2 Filebench Setup

On each client (Load and Test) the source file filebench-1.4.8.tar.gz was used to compile and install filebench 1.4.8 on the local disk of the client. A needed library libtecla-1.6.1[7] was installed from source prior to compiling and installing filebench. Patching the source[8] of filebench 1.4.8 was needed for successful install and compilation. After patching filebench was compiled and installed using:

```
aclocal;autoconf;autoheader;automake --add-missing --copy
./configure --prefix=/usr --exec_prefix=/usr
make
sed -ie "s/\/opt\/filebench\/bin/\/usr\/bin/" ./bin/filebench
sed -ie "s/\/opt\/filebench/\/usr\/share\/filebench/" ./bin/filebench
make install
```

Filebench has various default tests that contain workload personalities. For performance tests and load generation the "fileserver" workload is used. This is a file system workload, similar to SPECsfs[9]. This workload performs a sequence of creates, deletes, appends, reads, writes and attribute operations on the file system. Of the adjustable parameters, only the statistics output directory and the directory where the tests are executed are modified. The other values are default values. The fileserver personality has a run time of 600 seconds for the performance tests. For the auto-pilot statistics the values MB/s and OPS/s of the filebench output were used. These values are for the entire fileserver personality test and include read and write actions.

4.3 IOzone Setup

IOzone performs a broad filesystem analysis. The benchmark tests file I/O performance for the many types of Read and Write operations. UvA-IC T.S. main interest is in the Random Write operation. File IO is limited to files contained in NFS mounted directories. Commit time for NFS V3 is included in the measurements by including file closure times ("-c"). File sizes run from 64Kbytes to 4096Kbytes, using record sizes from 4Kbytes up to 4096Kbytes ("-ag4096"). Only the Random Read and Random Writes are tested ("-i0 -i2"). Random read/write – The records, at random offsets will be written and read. For the auto-pilot statistics the values of random reads and writes per second of the file size of 1024KB of and a record size of 128KB of the IOzone output were used. Note: unlike the filebench test, IOzone measures read and write MB/s as separate values.

[7] http://www.astro.caltech.edu/~mcs/tecla/

[8] http://mail.opensolaris.org/pipermail/perf-discuss/attachments/20090722/be13dac5/attachment.obj

[9] http://www.spec.org/benchmarks.html#nfs

4.4 Dtrace Setup

A Dtrace scripts was created to display the values of the probes (figure 5) on a 10 second basis with the probe values being divided by 10 to create a per second output value. Appendix B shows this script in two versions, one with headers and one without. The noheader version in Appendix B.2 allows for easy logging and processing of the output.

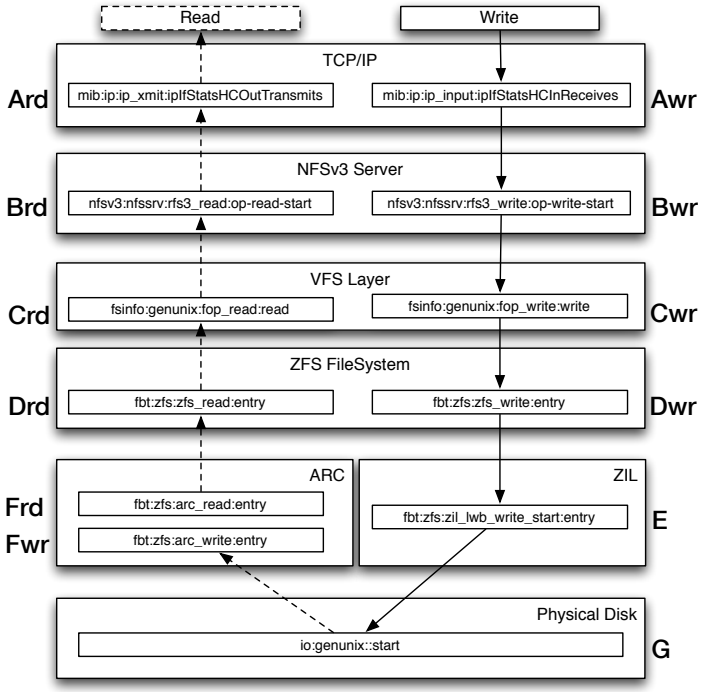

Figure 5: Dtrace Probes and the names used in de Dtrace script output.

4.5 Testing method

1. Log Dtrace probes while nothing is happening on the server
2. Log Dtrace probes while running IOzone and filebench (using autopilot) from TestClient while "nothing" is no load on the NFS server.
3. Log Dtrace probes while filebench is running on 1 LoadClient using the server
4. Log Dtrace probes while running IOzone and filebench from TestClient
5. Repeat steps 3 and 4 for 2,3,4,8,16 hosts, where 8 is 2 concurrent runs on each LoadClient and 16 is 4 concurrent runs.

5 Results

5.1 Preliminary test results

A preliminary test run of the benchmark setup was done to watch the behavior of the Dtrace probes using the Dtrace script (section 4.4). This to get familiar with the Dtrace tool and its output. The first 3 lines show output when a filebench benchmark is running and there is no load. The second 3 lines shows output when a filebench benchmark is running with the load of 1 LoadClient running filebench as load generator. The third 3 lines with the load of 2 LoadClients running filebench as load generator.

Time	Ard	Awr	Brd	Bwr	Crd	Cwr	Drd	Dwr	E	Frd	Fwr	G
1264371753	5515	13250	0	233	1	227	0	233	79	0	142	3090
1264371763	5441	13096	0	237	1	231	0	237	82	0	144	3156
1264371773	6004	11816	0	213	2	210	0	213	74	0	799	5387

Time	Ard	Awr	Brd	Bwr	Crd	Cwr	Drd	Dwr	E	Frd	Fwr	G
1264374103	6638	11779	0	210	1	205	0	210	74	0	819	5447
1264374113	7212	12849	0	223	2	218	0	223	78	0	145	3118
1264374123	7170	12698	0	233	1	228	0	233	80	0	144	3127

Time	Ard	Awr	Brd	Bwr	Crd	Cwr	Drd	Dwr	E	Frd	Fwr	G
1264375343	7278	11612	0	204	5	205	0	204	68	0	895	5968
1264375353	7836	12600	0	226	1	223	0	226	74	0	145	3057
1264375363	7548	12087	0	220	0	215	0	220	72	0	743	5491

Looking at this data, it seams like all the layers are already a their maximum troughput. It was soon determined, using the "prstat" tool on the server, that the number of NFS daemons running on the server was low compared to the current NFS fileserver in the production environment. The bottle-neck was removed by changing the server settings to allow more NFS-daemons and change the settings to that of the production server. The results where visible on the server as the following output shows:

Time	Ard	Awr	Brd	Bwr	Crd	Cwr	Drd	Dwr	E	Frd	Fwr	G
1264417143	18038	28526	0	471	12	468	0	472	61	0	343	5505
1264417153	15924	24902	0	413	11	408	0	413	57	0	1308	9811
1264417163	15604	24441	0	402	11	397	0	402	58	0	1241	8452

The graph (Figure 6 of the complete Dtrace output with the old NFS setting and the the new NFS setting clearly show the difference in of the settings and its influence on the other performance indicators of the server.

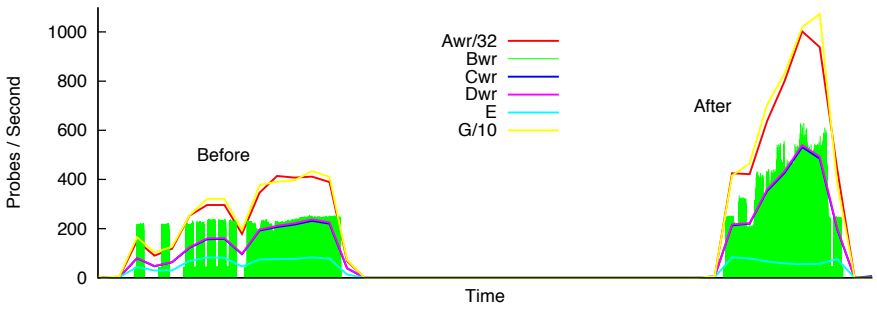

Figure 6: Probes/Second during preliminary test before (right) and after (left) changing number of NFS-daemon processes

The number of write probes per second for the TCP/IP (Awr) Layer and the number of probes per second for the physical layer (G) are a number of magnitudes bigger then the other inspected probes. By dividing the TCP/IP write (Awr) by 32 and the physical layer (G) by 10, both probes were better visualized in the graph (figure 6) without suppressing the trend.

5.2 Filebench and IOzone test results

Using the test setup and method described in the previous section (section 4) the following means (table 1) were calculated by auto-pilot when running filebench and IOzone benchmarks.

Load	0	1	2	3	4	8	16
Filebench OPs/s	1224	862	764	649	533	607	583
Filebench MB/s	29.6	20.5	17.7	15.4	13.8	14.4	13.8
IOzone Read MB/s	1001.8	1003.7	1005.5	997.6	1001.9	1006.3	1001.3
IOzone Write MB/s	30.168	20.7	20.4	22.2	20.8	16.4	19.2

Table 1: Filebench and IOzone means of the tests per load

Auto-pilot will run the tests at least 5 times and as long as it can not calculate a confidence interval of 5% (HW%) for the mean it will continue up till 30 tests (section 4.1). The numbers of test and the HW% is in table 2. The HW% values in table 2 for all filebench benchmark tests done, are less than 5%. The IOzone test with a load of 0 is the only IOzone test where HW% $< 5\%$ all other tests did run 30 times. The HW% of these benchmark tests is between 12% and 20%. The IOzone tests with load of the LoadClients are unreliable.

Load	0	1	2	3	4	8	16
Number of filebench Tests	5	10	7	6	5	5	5
HW% of MB/s	4.232	4.715	4.962	4.912	3.509	0.720	3.726
Number of IOzone Tests	7	30	30	30	30	30	30
HW% of Write MB/s	4.125	12.128	16.408	13.692	14.082	19.202	15.615

Table 2: Filebench and IOzone number of tests per load and the Half With as a percentage of the mean (HW%)

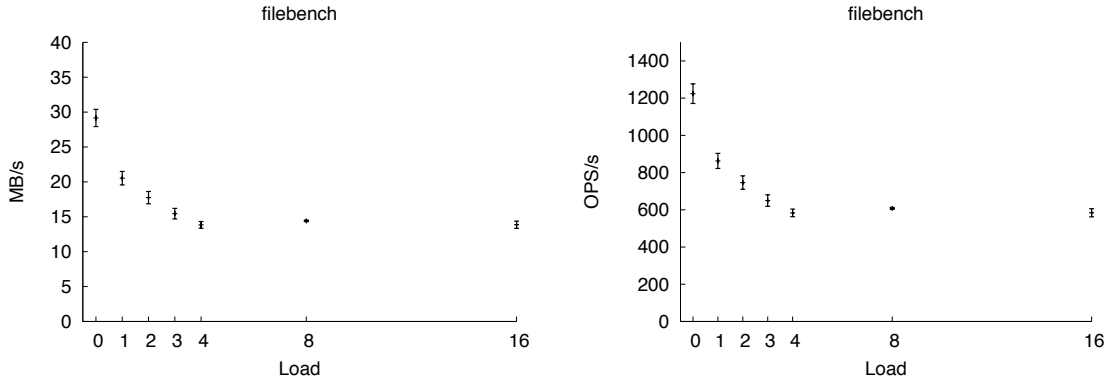

Figure 7: Filebench performance tests, MB/s and OPS/s

The graphs in figure 7 show that both the MB/s and the OPS/s have the same trend. This is as expected: every operation writes/reads an amount of data that translated in MB/s. The MB/s (and OPS/s) decline from almost 30 MB/s (load of 0) to around 14 MB/s (load of 4) and keep at this level with loads of 8 and 16. Knowing that the 8 and 16 load are generated by four LoadClients, running 2 or 4 load generating processes, this trend is due to the limitations of the LoadClients.

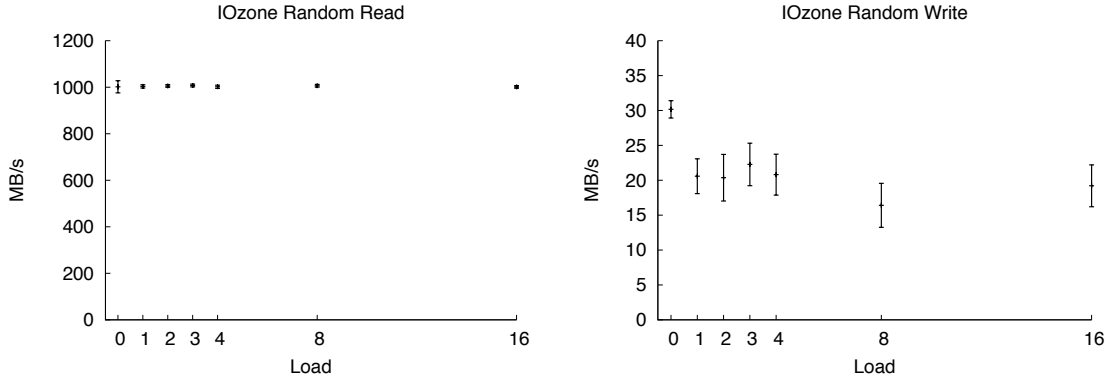

Figure 8: IOzone performance tests, random read MB/s and random write MB/s

The IOzone tests are divided in random read and random write (figure 8). The random read on an NFS Client passes a lot of layers with caches. The 1GB/s random read is therefore no indication of the performance of the NFS fileserver. The write characteristics have the same trend as the filebench values but the trend already levels at a load of 1. As stated before, the IOzone tests with the load of the LoadClients are very unreliable.

5.3 NFS fileserver performance indicators (Dtrace probes)

During the filebench load generation, the filebench tests and IOzone tests where running on the Clients, the noheaders Dtrace (appendix B.2) script was running on the NFS fileserver. Its output was saved to a file for further inspection. The focus of this inspection was on the write probes. The IOzone benchmark showed that reading is limited on the NFS client not on the NFS fileserver. The Dtrace output confirmed this, the VFS layer and NFS layer read probes where close to zero. For the the physical layer (disks) there is no distinction made between read and write actions. The trend and baseline of the write probes during filebench load generation of 0 to 4 LoadClients and benchmarking by the TestClient are shown in figure 9. All write probes except the "write ZIL" (E) probe have the same trend. The "write ZIL" levels at about 80 probes per second.

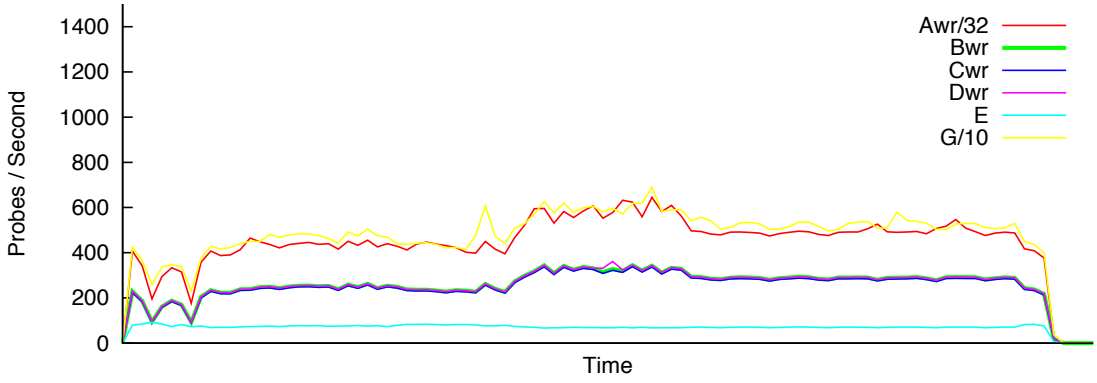

Figure 9: Probes/Second during the filebench tests with 0 to 4 LoadClients active

5.4 Filebench and IOzone test results with ZIL on SSD

Like the preliminary test in section 5.1 the performance indicators in the previous section (5.3) are indicating an other bottle-neck: the ZIL. It is known that the ZIL has a major influence on the performance of synchronous writes[10,11]. By using fast storage for the ZIL, performance can improve. To test this hypothesis the test setup and method described in section 4 was again used. The ZFS configuration was changed and the ZIL was placed on 2 SSD disks in striped mode. Original setup in appendix A.5 new setup in appendix A.6. The following means were calculated by auto-pilot when running filebench (table 3) and IOzone (table 4) benchmarks.

Load	0	1	2	3	4	8	16
Filebench MB/s	50.0	29.2	21.7	17.5	15.3	14.8	15.4
Number of filebench Tests	5	5	5	5	5	5	6
HW% of MB/s	2.772	1.014	1.100	0.593	0.724	2.934	3.831

Table 3: Filebench test results - ZIL on SSD

Load	0	1	2	3	4	8	16
IOzone MB/s	45.1	36.9	23.4	23.4	17.5	19.1	18.7
Number of IOzone Tests	30	30	30	30	30	30	30
HW% of MB/s	8.118	14.318	21.181	13.495	20.001	15.008	15.762

Table 4: IOzone random write test results - ZIL on SSD

[10] http://blogs.sun.com/brendan/entry/slog_screenshots
[11] http://blogs.sun.com/ahl/entry/fishworks_launch

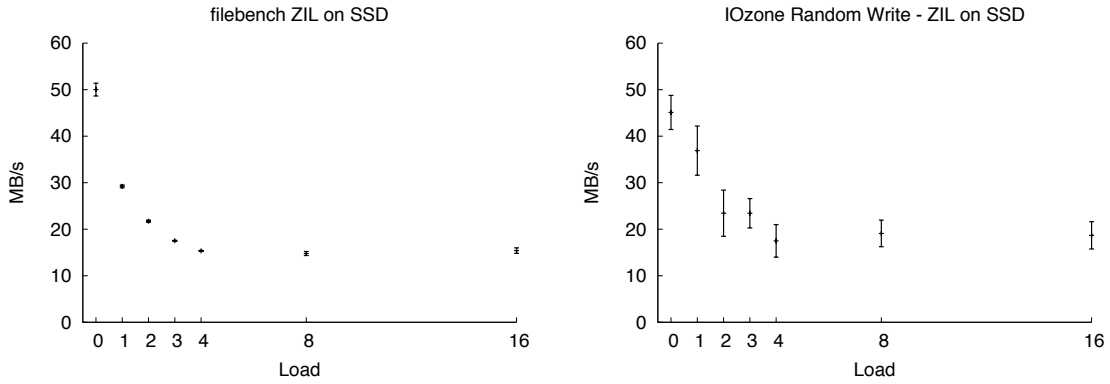

Figure 10: Filebench ad IOzone performance test with ZIL on SSD

The MB/s of both filebench and IOzone show an improvement when there is no load on the server. The filebench test is much more influenced by the load of 3 and 4 LoadClients, while IOzone levels at 2. Again the IOzone test are unreliable especially with load from the LoadClients. In figure 10 are the graphs of the measured values.

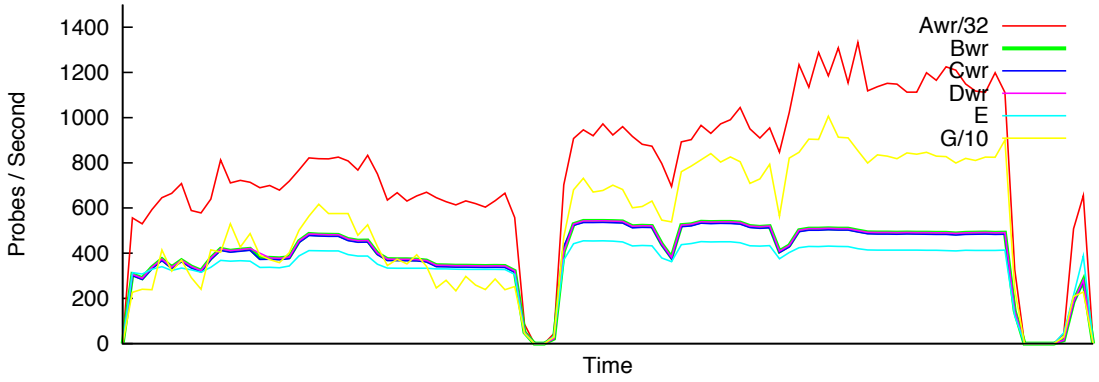

Figure 11: Probes/Second during the filebench tests with 0 to 4 LoadClients active ZIL on SSD

The Dtrace write probes for the ZIL (E) in figure 11 now show a maximum of write probes per second of about 380. The "write ZIL" now follows the trend of the other probes. The VFS layer and NFS layer now seam to flatten when the load is getting higher.

6 Conclusion and Future work

6.1 Conclusion

Referring back to the research question:

> *How can the performance bottlenecks be monitored and identified on an OpenSolaris^{TM} OS NFS fileserver. What are realistic load simulations and create a base-line.*

The following conclusions are made.

The Dtrace write probes defined, can be a performance indicator of the NFS fileserver. The output of the Dtrace probes can be used to identify performance bottle-necks.

Filebench and the fileserver personality are a way to show the performance impact of the load on a NFS fileserver (as experienced by the NFS Client, figure 12).

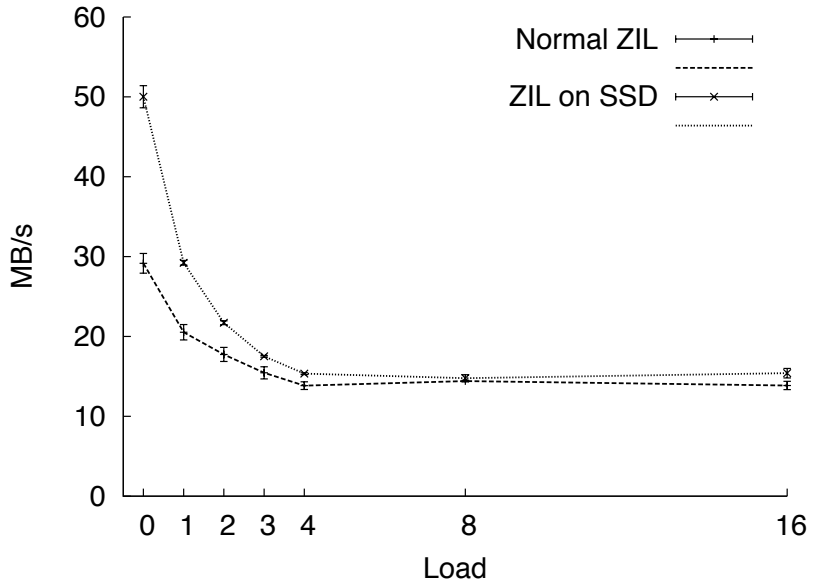

Figure 12: filebench performance test with and without ZIL on SSD

IOzone test are unreliable in the given environment and give an impression of non declining client performance when the load on the NFS increases.

The filebench tests on the TestClient can be used as a baseline for other test, for example when benchmarking other NFS fileserver Hardware. The client can be the same and has no influence on the test.

6.2 Further work

The tests performed in this research can be reproduced and a baseline was created. Using the filebench measurements as a baseline, other types of NFS fileserver hardware could be compared using the same tests. The influence of the ZIL on faster SSD devices, is an interesting subject.

Auto-pilot was used on the TestClient to get the statistics of the benchmark tests. The Dtrace probes provide baseline data of the NFS server. A subject for futher research can be: "How can auto-pilot be used on the NFS server to statistically compare current behavior with the baseline?"

References

[1] Richard McDougall, Jim Mauro, and Brendan Gregg. *Solaris(TM) Performance and Tools: DTrace and MDB Techniques for Solaris 10 and OpenSolaris (Solaris Series)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2006.

[2] SUN Microsystems. *DTrace User Guide (819–5488–10)*. SUN Microsystems, 2006.

[3] SUN Microsystems. *Solaris Dynamic Tracing Guide (817–6223–12)*. SUN Microsystems, 2008.

[4] A. Traeger, E. Zadok, E. L. Miller, and D. D. E. Long. Findings from the first annual storage and file systems benchmarking workshop. *;login: The USENIX Magazine*, 33(5):113–117, October 2008.

[5] Avishay Traeger, Erez Zadok, Nikolai Joukov, and Charles P. Wright. A nine year study of file system and storage benchmarking. Technical report, 2007.

[6] Charles P. Wright, Nikolai Joukov, Devaki Kulkarni, Yevgeniy Miretskiy, and Erez Zadok. Auto-pilot: A platform for system software benchmarking. In *USENIX Annual Technical Conference, FREENIX Track*, pages 175–188, 2005.

A Test Systems Hardware/Software specifications

A.1 NFS fileserver

SUN x4540

CPU	2 x Quad-Core AMD Opteron 2356, 2300MHz
Memory	32GB
Disk	46x500GB SATA HITACHI HUA7250S-A90A 2x30GB SSD OCZ VERTEX-TURBO Setup of ZFS in appendix A.5 and A.6
Network	4 x 1Gb (1 used)
OS	OpenSolaris SVN_129

A.2 VMWare Servers 01 and 02

SUN x4440

CPU	4 x Quad-core AMD Opteron 8384 , 2700MHz
Memory	32GB
Disk	Local 2x146GB SATA and NFS for VMware Images
Network	4 x 1Gb
OS	VMware 4 EXi

A.3 LoadClients 01, 02, 03 and 04

VMware Virtual Machine

CPU	Quad-Core AMD Opteron 8384, 2700MHz (1 core assigned)
Memory	4GB
Disk	20GB VMware-Disk
Network	2 x 1Gb (1 dedicated used)
OS	CentOS 5.4 (yum update 25/01/2010 13:00)

A.4 TestClient

SUN x2200

CPU	2 x Dual-Core AMD Opteron 2210, 1000MHz
Memory	2GB
Disk	750GB Sata
Network	4 x 1Gb (1 dedicated used)
OS	CentOS 5.4 (yum update 25/01/2010 13:00)

A.5 ZFS Setup

```
  pool: bootpool
 state: ONLINE
```

```
 scrub: none requested
config:

        NAME          STATE     READ WRITE CKSUM
        bootpool      ONLINE       0     0     0
          mirror-0    ONLINE       0     0     0
            c0t0d0s0  ONLINE       0     0     0
            c1t0d0s0  ONLINE       0     0     0

errors: No known data errors

  pool: mypool
 state: ONLINE
 scrub: none requested
config:

        NAME        STATE     READ WRITE CKSUM
        mypool      ONLINE       0     0     0
          raidz2-0  ONLINE       0     0     0
            c0t1d0  ONLINE       0     0     0
            c1t1d0  ONLINE       0     0     0
            c2t1d0  ONLINE       0     0     0
            c3t1d0  ONLINE       0     0     0
            c4t1d0  ONLINE       0     0     0
            c5t1d0  ONLINE       0     0     0
            c0t2d0  ONLINE       0     0     0
            c1t2d0  ONLINE       0     0     0
            c2t2d0  ONLINE       0     0     0
            c3t2d0  ONLINE       0     0     0
            c4t2d0  ONLINE       0     0     0
            c5t2d0  ONLINE       0     0     0
            c0t3d0  ONLINE       0     0     0
            c1t3d0  ONLINE       0     0     0
          raidz2-1  ONLINE       0     0     0
            c2t3d0  ONLINE       0     0     0
            c3t3d0  ONLINE       0     0     0
            c4t3d0  ONLINE       0     0     0
            c5t3d0  ONLINE       0     0     0
            c0t4d0  ONLINE       0     0     0
            c1t4d0  ONLINE       0     0     0
            c2t4d0  ONLINE       0     0     0
            c3t4d0  ONLINE       0     0     0
            c4t4d0  ONLINE       0     0     0
            c5t4d0  ONLINE       0     0     0
            c0t5d0  ONLINE       0     0     0
            c1t5d0  ONLINE       0     0     0
            c2t5d0  ONLINE       0     0     0
            c3t5d0  ONLINE       0     0     0
          raidz2-2  ONLINE       0     0     0
            c4t5d0  ONLINE       0     0     0
            c5t5d0  ONLINE       0     0     0
            c0t6d0  ONLINE       0     0     0
            c1t6d0  ONLINE       0     0     0
            c2t6d0  ONLINE       0     0     0
            c3t6d0  ONLINE       0     0     0
            c4t6d0  ONLINE       0     0     0
            c5t6d0  ONLINE       0     0     0
            c0t7d0  ONLINE       0     0     0
            c1t7d0  ONLINE       0     0     0
            c2t7d0  ONLINE       0     0     0
```

```
            c3t7d0  ONLINE       0     0     0
            c4t7d0  ONLINE       0     0     0
            c5t7d0  ONLINE       0     0     0
        spares
          c2t0d0    AVAIL
          c3t0d0    AVAIL
          c4t0d0    AVAIL
          c5t0d0    AVAIL

errors: No known data errors
```

A.6 ZFS Setup ZIL on SSD

```
  pool: bootpool
 state: ONLINE
 scrub: none requested
config:

        NAME          STATE     READ WRITE CKSUM
        bootpool      ONLINE       0     0     0
          mirror-0    ONLINE       0     0     0
            c0t0d0s0  ONLINE       0     0     0
            c1t0d0s0  ONLINE       0     0     0

errors: No known data errors

  pool: mypool
 state: ONLINE
 scrub: none requested
config:

        NAME        STATE     READ WRITE CKSUM
        mypool      ONLINE       0     0     0
          raidz2-0  ONLINE       0     0     0
            c0t1d0  ONLINE       0     0     0
            c1t1d0  ONLINE       0     0     0
            c2t1d0  ONLINE       0     0     0
            c3t1d0  ONLINE       0     0     0
            c4t1d0  ONLINE       0     0     0
            c5t1d0  ONLINE       0     0     0
            c0t2d0  ONLINE       0     0     0
            c1t2d0  ONLINE       0     0     0
            c2t2d0  ONLINE       0     0     0
            c3t2d0  ONLINE       0     0     0
            c4t2d0  ONLINE       0     0     0
            c5t2d0  ONLINE       0     0     0
            c0t3d0  ONLINE       0     0     0
            c1t3d0  ONLINE       0     0     0
          raidz2-1  ONLINE       0     0     0
            c2t3d0  ONLINE       0     0     0
            c3t3d0  ONLINE       0     0     0
            c4t3d0  ONLINE       0     0     0
            c5t3d0  ONLINE       0     0     0
            c0t4d0  ONLINE       0     0     0
            c1t4d0  ONLINE       0     0     0
            c2t4d0  ONLINE       0     0     0
            c3t4d0  ONLINE       0     0     0
            c4t4d0  ONLINE       0     0     0
            c5t4d0  ONLINE       0     0     0
            c0t5d0  ONLINE       0     0     0
```

```
    c1t5d0  ONLINE       0     0     0
    c2t5d0  ONLINE       0     0     0
    c3t5d0  ONLINE       0     0     0
  raidz2-2  ONLINE       0     0     0
    c4t5d0  ONLINE       0     0     0
    c5t5d0  ONLINE       0     0     0
    c0t6d0  ONLINE       0     0     0
    c1t6d0  ONLINE       0     0     0
    c2t6d0  ONLINE       0     0     0
    c3t6d0  ONLINE       0     0     0
    c4t6d0  ONLINE       0     0     0
    c5t6d0  ONLINE       0     0     0
    c0t7d0  ONLINE       0     0     0
    c1t7d0  ONLINE       0     0     0
    c2t7d0  ONLINE       0     0     0
    c3t7d0  ONLINE       0     0     0
    c4t7d0  ONLINE       0     0     0
    c5t7d0  ONLINE       0     0     0
logs
  c2t0d0    ONLINE       0     0     0
  c3t0d0    ONLINE       0     0     0
spares
  c4t0d0    AVAIL
  c5t0d0    AVAIL
```

B Dtrace Scripts

B.1 read_write_probes.d

```
#!/usr/sbin/dtrace -s
#pragma D option quiet

dtrace:::BEGIN {
     Ard = 0;Awr = 0;Brd = 0;Bwr = 0;
     Crd = 0;Cwr = 0;Drd = 0;Dwr = 0;
     E = 0;Frd = 0;Fwr = 0;G = 0;
printf("%10s %7s %7s %7s %7s %7s %7s %7s %7s %7s %7s %7s %7s\n" \
,"Time","Ard","Awr","Brd","Bwr","Crd","Cwr","Drd","Dwr","E","Frd","Fwr","G");
}

mib:ip:ip_xmit:ipIfStatsHCOutTransmits { Ard++; }
mib:ip:ip_input:ipIfStatsHCInReceives  { Awr++; }
nfsv3:nfssrv:rfs3_read:op-read-start   { Brd++; }
nfsv3:nfssrv:rfs3_write:op-write-start { Bwr++; }
fsinfo:genunix:fop_read:read           { Crd++; }
fsinfo:genunix:fop_write:write         { Cwr++; }
fbt:zfs:zfs_read:entry                 { Drd++; }
fbt:zfs:zfs_write:entry                { Dwr++; }
fbt:zfs:zil_lwb_write_start:entry      { E++; }
fbt:zfs:arc_read:entry                 { Frd++; }
fbt:zfs:arc_write:entry                { Fwr++; }
io:genunix::start                      { G++; }

profile:::tick-10s {
        Ard = Ard / 10; Awr = Awr / 10; Brd = Brd / 10; Bwr = Bwr / 10;
        Cwr = Cwr / 10; Crd = Crd / 10; Dwr = Dwr / 10; Drd = Drd / 10;
E = E / 10; Fwr = Fwr / 10; Frd = Frd / 10; G = G / 10;
ts = walltimestamp / 1000000000;
printf("%d %7d %7d %7d %7d %7d %7d %7d %7d %7d %7d %7d %7d\n" \
,ts,Ard,Awr,Brd,Bwr,Crd,Cwr,Drd,Dwr,E,Frd,Fwr,G);
        Ard = 0;Awr = 0;Brd = 0;Bwr = 0;
        Crd = 0;Cwr = 0;Drd = 0;Dwr = 0;
        E = 0;Frd = 0;Fwr = 0;G = 0;

}
profile:::tick-100s {
printf("%10s %7s %7s %7s %7s %7s %7s %7s %7s %7s %7s %7s %7s\n" \
,"Time","Ard","Awr","Brd","Bwr","Crd","Cwr","Drd","Dwr","E","Frd","Fwr","G");
}
```

B.2 read_write_probes_noheaders.d

```
#!/usr/sbin/dtrace -s
#pragma D option quiet

dtrace:::BEGIN {
     Ard = 0;Awr = 0;Brd = 0;Bwr = 0;
     Crd = 0;Cwr = 0;Drd = 0;Dwr = 0;
     E = 0;Frd = 0;Fwr = 0;G = 0;
}

mib:ip:ip_xmit:ipIfStatsHCOutTransmits { Ard++; }
mib:ip:ip_input:ipIfStatsHCInReceives  { Awr++; }
nfsv3:nfssrv:rfs3_read:op-read-start   { Brd++; }
nfsv3:nfssrv:rfs3_write:op-write-start { Bwr++; }
```

```
fsinfo:genunix:fop_read:read           { Crd++; }
fsinfo:genunix:fop_write:write         { Cwr++; }
fbt:zfs:zfs_read:entry                 { Drd++; }
fbt:zfs:zfs_write:entry                { Dwr++; }
fbt:zfs:zil_lwb_write_start:entry      { E++; }
fbt:zfs:arc_read:entry                 { Frd++; }
fbt:zfs:arc_write:entry                { Fwr++; }
io:genunix::start                      { G++; }

profile:::tick-10s {
        Ard = Ard / 10; Awr = Awr / 10; Brd = Brd / 10; Bwr = Bwr / 10;
        Cwr = Cwr / 10; Crd = Crd / 10; Dwr = Dwr / 10; Drd = Drd / 10;
E = E / 10; Fwr = Fwr / 10; Frd = Frd / 10; G = G / 10;
ts = walltimestamp / 1000000000;
printf("%d %7d %7d %7d %7d %7d %7d %7d %7d %7d %7d %7d %7d\n" \
,ts,Ard,Awr,Brd,Bwr,Crd,Cwr,Drd,Dwr,E,Frd,Fwr,G);
        Ard = 0;Awr = 0;Brd = 0;Bwr = 0;
        Crd = 0;Cwr = 0;Drd = 0;Dwr = 0;
        E = 0;Frd = 0;Fwr = 0;G = 0;
}
```

C Auto-pilot Scripts

C.1 filebench_loadgen.ap

```
#!/usr/bin/perl /usr/bin/auto-pilot
# filebench_loadgen.ap

QUIET true

VAR BENCH=filebench_loadgen
VAR TERMINATE=1
VAR NTHREADS=1

RESULTS=$HOME$/results/%BENCH%
LOGS=$HOME$/logs/%BENCH%

ENV TESTFS=nfs
ENV FBRUNTIME=36000
ENV TESTROOT=/test_NFS

INCLUDE common.inc

THREADS=%NTHREADS%

TEST filebench_loadgen-%THREADS% %TERMINATE%
SETUP mount_nfs.sh %THREADS%
EXEC filebench_loadgen.sh
CLEANUP umount_nfs.sh %THREADS%
DONE
```

C.2 mount_nfs.sh

```
#!/bin/bash
# mount_nfs.sh

for i in `seq $1`
 do mount 10.0.0.1:/mypool/test_FS/$HOSTNAME/$i /test_NFS/$i
done
```

C.3 filebench_loadgen.sh

```
#!/bin/bash
# filebench_loadgen.sh

if [ -z "$TESTROOT" ] ; then
echo "TESTROOT EnvVar Empty"
exit 1
fi

if [ ! -d "$TESTROOT" ] ; then
echo "TESTROOT not a directory"
exit 1
fi


source commonsettings || exit $?

FILEBENCH="/usr/bin/filebench"
TESTNAME="fileserver"
OUTDIR="/root/stats/filebench_loadgen/${APEPOCH}-${APTHREAD}"
```

```
CONFIG="loadconf-$$"

BENCHDIR=${TESTROOT}/${APTHREAD}

cat <<END > "${CONFIG}.prof"
CONFIG $TESTNAME {
function = generic;
personality = $TESTNAME;
}

DEFAULTS {
description = "FileServer_loadgen";
runtime = ${FBRUNTIME};
stats = ${OUTDIR};
filesystem = ${TESTFS};
dir = ${BENCHDIR};
}
END
semdec $APIPCKEY
ap_measure $FILEBENCH $CONFIG
rm ${CONFIG}.prof
exit 0
```

C.4 umount_nfs.sh

```
#!/bin/bash
# umount_nfs.sh

for i in ‘seq $1‘
 do umount /test_NFS/$i
done
```

C.5 filebench_perftest.ap

```
#!/usr/bin/perl /usr/bin/auto-pilot
# filebench_perftest.ap

QUIET true

VAR BENCH=filebench_perftest
VAR NTHREADS=1

RESULTS=$HOME$/results/%BENCH%
LOGS=$HOME$/logs/%BENCH%

ENV TESTFS=nfs
ENV TESTROOT=/test_NFS
ENV NFSSERVER=10.0.0.1
ENV SERVERROOT=/mypool/test_FS/$HOSTNAME$
ENV FBRUNTIME=600

VAR TERMINATE=5 1 /usr/bin/getstats --predicate \
’("$name" ne "opss" && "$name" ne "mbs") || \
("$delta" < 0.05 * $mean) || ($count >= 30)’ --

INCLUDE common.inc

TEST %BENCH% %TERMINATE%
SETUP fs-setup.sh %TESTFS%
EXEC filebench_perftest.sh
```

```
CLEANUP fs-cleanup.sh %TESTFS%
DONE
```

C.6 filebench_perftest.sh

```
#!/bin/bash
# filebench_perftest.sh

MEASURE_FILEBENCH="on"

source commonsettings || exit $?

FILEBENCH="/usr/bin/filebench"
TESTNAME="fileserver"
OUTDIR="/root/filebench-perftest/$APEPOCH"
SAFEDIR="/root/filebench-perftest.archive/‘date ’+%Y_%m_%d_%H%M%S’‘/"
CONFIG="perfconf-$$"

cat <<END > "${CONFIG}.prof"
CONFIG $TESTNAME {
function = generic;
personality = $TESTNAME;
}

DEFAULTS {
description = "FileServer";
runtime = ${FBRUNTIME};
stats = ${OUTDIR};
filesystem = ${TESTFS};
dir = ${TESTROOT};
}
END

semdec $APIPCKEY
ap_measure $FILEBENCH $CONFIG
rm ${CONFIG}.prof
mkdir -p $SAFEDIR
cp -r $OUTDIR/* $SAFEDIR
rm -r $OUTDIR/*

exit 0
```

C.7 commonsettings.dfilebench

```
#!/bin/bash
# filebench (commonsettings.d)

ME=filebench
TYPE=measure

if [ ! -z "AP_MEASURE_HOOK" ] ; then
        eval ‘echo "AP_SAVED_"$TYPE"_HOOK_"$ME"=""$AP_MEASURE_HOOK"‘
fi
if [ ! -z "$MEASURE_FILEBENCH" ] ; then
   AP_MEASURE_HOOK=ap_measure_filebench
fi

function ap_measure_filebench {
        local ME=filebench
        local TYPE=measure
```

```
        if [ "$1" = "premeasure" ] ; then
                true
        elif [ "$1" = "start" ] ; then
                true
        elif [ "$1" = "end" ] ; then
                ap_logexec /usr/share/auto-pilot/fb2ap.sh \
  ${OUTDIR}/*/${TESTNAME}/stats.${TESTNAME}.out || return $?
        elif [ "$1" = "final" ] ; then
         true
        else
                echo "Filebench Measurement hook failed " 1>&2
                exit 1
        fi

        local HOOK=‘eval "echo $""AP_SAVED_"$TYPE"_HOOK_"$ME‘
        if [ ! -z "$HOOK" ] ; then
                "$HOOK" $*
                return $?
        fi

        return 0
}

unset ME
```

C.8 fb2ap.sh

```
#!/bin/bash
# fb2ap.sh Transform filebench output to auto-pilot

cat $1 | grep "IO" | sed -e ’s/mb\/s,//’ | awk ’{print "ops = "$3"\nopss = "$5"\nmbs = "$9 }’
```

C.9 iozone_perftest.ap

```
#!/usr/bin/perl /usr/bin/auto-pilot
# iozone_perftest.ap

QUIET true

VAR BENCH=iozone_perftest
VAR NTHREADS=1

RESULTS=$HOME$/results/%BENCH%
LOGS=$HOME$/logs/%BENCH%

ENV TESTFS=nfs
ENV TESTROOT=/test_NFS
ENV NFSSERVER=10.0.0.1
ENV SERVERROOT=/mypool/test_FS/$HOSTNAME$

VAR TERMINATE=5 1 /usr/bin/getstats --predicate \
’("$name" ne "rkbs" && "$name" ne "wkbs") || \
("$delta" < 0.05 * $mean) || ($count >= 30)’ --

INCLUDE common.inc

TEST %BENCH% %TERMINATE%
SETUP fs-setup.sh %TESTFS%
EXEC iozone_perftest.sh
```

```
CLEANUP fs-cleanup.sh %TESTFS%
DONE
```

C.10 iozone_perftest.sh

```
#!/bin/bash
# iozone_perftest.sh

MEASURE_IOZONE="on"

source commonsettings || exit $?

OUTDIR="/root/iozone-perftest/$APEPOCH"
SAFEDIR="/root/iozone-perftest.archive/‘date ’+%Y_%m_%d_%H%M%S’‘/"
mkdir -p $OUTDIR
IOZONE="/usr/bin/iozone"
IOZONEOPT="-acg4096 -f ${TESTROOT}/iozone.test -i 0 -i 2"

semdec $APIPCKEY
ap_measure ${IOZONE} ${IOZONEOPT} > ${OUTDIR}/iozone.out

mkdir -p $SAFEDIR
cp -r $OUTDIR/* $SAFEDIR
rm -r $OUTDIR/iozone.out

exit 0
```

C.11 commonsettings.diozone

```
#!/bin/bash
# iozone (commonsettings.d)

ME=iozone
TYPE=measure

if [ ! -z "AP_MEASURE_HOOK" ] ; then
        eval ‘echo "AP_SAVED_"$TYPE"_HOOK_"$ME"=""$AP_MEASURE_HOOK"‘
fi
if [ ! -z "$MEASURE_IOZONE" ] ; then
   AP_MEASURE_HOOK=ap_measure_iozone
fi

function ap_measure_iozone {
        local ME=iozone
        local TYPE=measure

        if [ "$1" = "premeasure" ] ; then
                true
        elif [ "$1" = "start" ] ; then
                true
        elif [ "$1" = "end" ] ; then
                ap_logexec /usr/share/auto-pilot/io2ap.sh ${OUTDIR}/iozone.out \
|| return $?
        elif [ "$1" = "final" ] ; then
         true
        else
                echo "Iozone Measurement hook failed " 1>&2
                exit 1
        fi
```

```
        local HOOK=‘eval "echo $""AP_SAVED_"$TYPE"_HOOK_"$ME‘
        if [ ! -z "$HOOK" ] ; then
                "$HOOK" $*
                return $?
        fi

        return 0
}

unset ME
```

C.12 io2ap.sh

```
#!/bin/bash
# io2ap.sh Transform iozone output to auto-pilot

cat $1 | grep "            1024     128" | awk ’{print "rkbs = "$5"\nwkbs = "$6 }’
```