Sun's Network File System

Annalise Tarhan

March 25, 2021

1 Using the timestamps found in the first column, determine the period of time the traces were taken from.

First timestamp: 1034787600.773984
Wednesday, October 16, 2002 5:00:00.773

Last timestamp: 1034791199.541974
Wednesday, October 16, 2002 5:59:59.541

awk '(n == 0) n=1; t0 = $1 (n == 1) t = $1 END print t - t0' trace.txt
3598.77

The trace was taken over a period of an hour, with the difference between the first timestamp and the last being exactly $3598.77/3600 = 0.9996583$ hours.

2 How many of each type of operation occur in the trace? Sort these by frequency; which operation is the most frequent? Does NFS live up to its reputation?

awk '{print $8}' trace.txt | sort | uniq -c | sort -r -n

1610395 getattr
1043752 read
619819 write
131453 lookup
27699 access
19485 setattr
11640 remove
9924 create

9135 fsstat
4290 link
2297 readdirp
1321 fsinfo
918 readdir
501 rename
439 readlink
187 pathconf
136 symlink
36 mkdir
14 rmdir
4 mknod

It absolutely appears to be a file system, yes.

3 Make a distribution of file sizes accessed within the trace; what is the average file size? Also, how many different users access files in the trace? Do a few users dominate traffic, or is it more spread out? What other interesting information is found within GETATTR replies?

getattr_sizes:

```
($8 == ``getattr" && $20 == ``size")
{printf(``%d\n", ``0x" $21)}
```

average_size:

```
{num = num + $1} {total = total + $1 * $2}
END { printf(``
        Total size: %d Total files: %d
        Average file size: %d\n",
        total, num, total/num    ) }
```

```
awk -f getattr_sizes trace.txt|sort -n|uniq -c|awk -f average_size
```

Total size: 14239080477780 Total files: 803160 Average file size: 17728821

getattr_users

```
($5 == ``C3" && $8 == ``getattr") { printf(``%s\n", $2) }
```

```
awk -f getattr_users trace.txt | sort -n | uniq -c | sort -n
```

This assumes that column two represents the user id when column five is C3. Over 800,000 getattr requests were made, but only by 136 separate users. Almost half of them came from a single user, and over three quarters came from just two. Every request was served by the same machine, identified as 30.0801.

4 Look at the details of the READ and WRITE requests to determine whether files are read or written to sequentially or randomly.

Note: This problem and the next one would have been a lot easier (and faster!) if I realized AWK can handle arrays. Oh well.

This solution uses AWK and a short shell script. The first step scans the file and extracts filehandles with corresponding offset/count data, then eliminates duplicates and those that were only accessed once. The second step scans the original file again, extracting all filehandles along with the offset and count information, and converts the hexadecimal offset/count data to decimal and saving it to tally.txt. The third step is to run the bash script, which runs an AWK command once for each of the unique filehandles, passing the filehandle as an argument. The AWK command scans tally.txt for that filehandle's operations and compares the offset to the offset that would be expected for a sequential operation. The final step is to add all the results of the bash script together. The result was 503,900 sequential accesses and 331,449 random accesses (neither including files that were only accessed once), so about 60% of the accesses were sequential.

Step 1

```
awk -f unique_filehandles.awk trace.txt
| sort -n
| uniq -c
| awk '($1>1) {print $2}' > uniq_fhs.txt
```

unique_filehandles.awk

```
($9 == ``fh" && $11 == ``off") { printf(``%s\n", $10)}
```

Step 2

```
gawk --non-decimal-data -f strip_convert.awk trace.txt
> trace_stripped_data
```

strip_convert.awk

```
($9 == ``fh" && $11 == ``off")
{printf(``%s %d %d\n", $10, ``0x" $12, ``0x" $14)}
```

Step 3

```
while read fh; do
        awk -v num=$fh -f tally.awk trace_stripped_data
        >> tally.txt
done < uniq_fhs.txt
```

tally.awk

```
($1 == num) { if ($2 == expected) seq++; else rndm++ }
{ expected = $2+$3 } END {printf(``%d %d\n", seq, rndm)}
```

Step 4

```
awk '{s=s+$1} {r=r+$2} END {printf(
``Total sequential accesses: %d\n
Total random accesses: %d\n", s, r)}'
tally.txt
```

5 Traffic comes from many machines and goes to one server. Compute a traffic matrix, which shows how many different clients there are in the trace, and how many requests/replies go to each. Do a few machines dominate, or is it more evenly balanced?

Out of the 219 users identified here, they were evenly split between those who were the source and destination an equal number of times and those who were the source more often than they were the destination. Once again, a few users dominated. The heaviest user generated a third of the traffic, and the top three together generated over 70%.

Step 1

```
awk `{ if ($2==30.0801) printf("\``%s\n", $3);
else printf(``%s\n", $2)}' trace.txt
| sort | uniq > users.txt
```

Step 2

```
while read user; do
        awk -v user=$user -f count_users.awk trace.txt
        >> user_count.txt
done < users.txt
```

count_users.awk

```
$2==user || $3==user { if ($2==user) source++; else dest++ }
END {printf(``%s %d %d\n", user, source, dest) }
```

Step 3

```
{if ($2==$3) same++}
{if ($2>$3) more_src++}
{if ($2<$3) more_dst++}
END {printf("Same: %d\nMore source: %d\nMore dest: %d\n",
        same, more_src, more_dst)}
```

```
cat user_count.txt | sort -k 2 -n
```

6 The timing information, and the per-request/reply unique ID, should allow you to compute the latency for a given request. Compute the latencies of all request/reply pairs. What is the average? Maximum? Minimum?

This isn't as simple as it should be, because the IDs aren't actually unique. Several are re-used, complicating calculations. As long as the ID was used in pairs I could correct for it, but I wasn't sure how to handle cases like 57fdf82a, where it is used for a quick back and forth, then for a standalone transaction, then again for another pair. How to avoid associating the standalone with the first transaction in the second pair? So, my data is junk. But here are the commands you'd use if it wasn't.

```
awk -f latencies.awk trace.txt | awk -f average.awk
```

```
awk -f latencies.awk trace.txt | awk -f maximum.awk
```

```
awk -f latencies.awk trace.txt | awk -f minimum.awk
```

latencies.awk

```
{if (timestamp[$6] > 0) print $1-timestamp[$6];
else timestamp[$6]=$1}
```

average.awk

```
{total = total + $1} {sum++} END {print total/sum}
```

maximum.awk

```
{ if ($1 > max) max = $1} END {print max}
```

minimum.awk

```
BEGIN { min = 100} { if ($1 < min) min = $1} END {print min}
```

7 Sometimes requests are retried, as the request or its reply could be lost or dropped. Can you find any evidence of such retrying in the trace sample?

There do seem to be some repeats, such as the following, repeated three times:

1034787601.463028 31.0320 30.0801 U C3 2c8a1970 1 getattr fh 1e2f6f0073480500200000000072570485932309648600000a0cf90002e303000 con = XXX len = XXX