sun microsystems, inc.
internal memorandum

To: File

From: Bill Joy, x333

Subject: NFS preliminary benchmarking

Date: September 6, 1984

Dan Walsh and I spent about an hour benchmarking a system where he had just got caching to work, on an idle network between two servers. We attempted a rough breakdown of the protocol overheads involved. Here is a report:

Time of trivial remote call: fstat

We first measured the time it took to do a loop over a 1000 fstat system calls on a file descriptor which was an open file on the file server. This took 29 seconds, giving us a time of 29 milliseconds per call. When run locally, this system call takes about 700 microseconds.

A guess at time breakdown of a trivial call

From the fact that the server was about 60% CPU busy while serving the fstat operations we estimate the the server is spending about 17 milliseconds per request. We believe that the Ethernet transit time for the request and the response is almost 0. We beliefve that the server machine will do a context switch from an idle state to the server process when the request arrives and that the client machine will context switch back to the client when the response arrives. Guessing that a context switch takes about 1 millisecond, this accounts for 20 milliseconds, and we guess that the remaining 9 milliseconds is in the protocol time on the client, giving:

client syscall	1
nfs code on client	9
client context switch	1
server context switch	1
server protocol	17

Read timing

We next wrote a program which did a read of a file randomly seeking so as to avoid all the caches. The program did 4kb requests. During the operation the server was 46% busy and we did 10 disk i/o operations per second, for a server service time of 46 milliseconds per request, and a total service time of 100 milliseconds per request. This gives us a guessed breakdown, incremental from that above

client syscall	1	
nfs code on client	9	
client context switch	1	
client data processing	5	(rough est)
ether transit time	5	

server context switch	1	
server basic protocol	17	
server disk latency	33	(100 – rest!?!)
disk protocol additional	28	

With readahead the server utilization goes up to nearly 80%, but the rest is idle because the client apparently cannot consume the data. During the experiment with read–ahead the client CPU is __% busy.

Additional measurements

One distressing measurement was that of a ‘‘stat(".")’’, which took 96 milliseconds, apparently 3 remote operations. This means that a stat call is taking nearly 100* longer over the net than locally. We believe that this could be done in one remote operation, or 30 milliseconds, without much trouble. A ‘‘stat("./.")’’ takes only 132 millisecodns, or 4 remote operations, showing the minimal increment in an additional pathname component.

Things to do

1. We need to reduce the time of the trivial operations as much as possible. In particular, commands like ‘‘make’’ do tons of stats; with then running at 1/10 sec per, the performance will be miserable.

2. CPU time is the bottleneck at both the server and the client machines. Improvements which can be anticipated include reducing the RPC protocol overhead (inline XDR expansion, and other wizardry should help her), and especially reducing the number of messages sent whenever possible (e.g. in stat).

3. Gprofing both the server and client sides to find bottlenecks during the read and write experiment should help a lot.

Conclusions

The CPU time is more of a bottleneck than even was anticipated in my previous memo about server performance. We have a lot of work to do to get from the current 46 milliseconds per 4k request to the 20 milliseconds which the V kernel people believed was possible on Sun–1 like hardware.