Measured Response Times for Page-Sized Fetches on a Network

Jonathan M. Smith†

Computer and Information Science, University of Pennsylvania, Philadelphia, PA 19104-6389

Gerald Q. Maguire, Jr.

Computer Science Department, Columbia University, New York, NY 10027

ABSTRACT

Previous work has examined the response times associated with disk accesses and the response time associated with various memory access operations. Here, we discuss the response time behavior of access requests for page-sized objects which are made using a network connection. We feel that future architectures must include networked resources as a component of the storage hierarchy; such performance measurements will help architects place these resources intelligently.

1. Introduction

As the architectures of *computing systems* have evolved, the components comprising the system have evolved as well. In particular, early computers were standalone and had small, slow, secondary storage, if any. Architectural advances coupled with deeper understanding of program characteristics brought us to the typical architecture of today. That is, a *computer system* as seen by a program is still a large address space and some processing facilities, but in reality this address space is *virtual.* The contents of the address space are made available with some address resolution method; the typical scheme today uses fixed-size *pages. Demand-paging* from secondary storage allows the computing system to preserve the illusion that it has a very large address space with much less physical memory.

The availability of relatively high-speed network interconnections such as the 10Mbit Ethernet™ has seen another chain of evolution. The chain started with network file transfer, and

———————

† Some of this work was done while the author was at Columbia University, and was supported in part by equipment grants from the Hewlett-Packard Corporation and AT&T, and NSF grant CDR-84-21402. This work was also supported, in part, by Bell Communications Research (Morristown, NJ) under Project DAWN.

progressed to the various network file systems [Smith1986a] which are available today, of which the most widely used is probably Sun Microsystems' NFS [Sandberg1985a, Sandberg1985b] Since programs are often demand-loaded from files which are in a remote namespace (consider, e.g., the popular configuration of a file server with several diskless workstations), it is crucial to examine the performance parameters of such an ''architecture''. Of particular interest is the response time seen by a program as page requests are serviced remotely. This will become more important as further architectural innovations lead to one-level stores (as in Multics [Organick1972a]) being implemented with networked components [Delp1988a].

2. Measured Network Response Time

Our interest in response time is that it seems most relevant performance measure to the user of a computing system. In addition, it is readily observable. We have done previous work under the assumption that response time is the right attribute to measure, in particular disk response times [Johnson1987a] and response times for the memory copying involved in a copy-on-write *fork()* scheme [Smith1988a].

2.1. Homogeneous machines

To gather information on the response time for network page requests, the program (''netrand.c'') was written. Since it is short, a source listing is provided as an appendix. The experiment was to operate the program on two files, one on local disk, and the other on an NFS-mounted file system. The system employed for testing was a Hewlett-Packard HP9000/350 running HP-UX™ 6.2. The system was equipped with 8 megabytes of main memory and a HP7945 70 megabyte hard disk.

The UNIX buffer cache mechanism was frustrated by copying a large (2 megabyte) file previous to running the tests. Three executions of the test were run on each file; for the purposes of benchmarking we ran the program with an argument of 400 blocks. Since the randomization frustrates the buffer cache mechanism to some degree, the buffer cache has a significant effect by the third execution. The results of running the program on the local file were 10.68, 10.24, and 9.74 seconds of elapsed time for the three runs. For the remote file (the server is a machine of the same type, accessed over a lightly-loaded 10Mbit Ethernet) the results were 15.42, 15.02, and 13.32 seconds of elapsed time for the three runs. The improvement in performance seen as the runs progressed is due to the success of the UNIX buffer cache in retaining recently-read blocks. The network access seems to indicate a penalty of about a factor of 1.5 for network access of pages, which is encouraging. The time per page (using the worst case, where the cache is flushed) gives us a per-page time of 15.42/400 seconds, or about 39 milliseconds per page. This works out to about 26 pages per second for 4K pages. This cost compares unfavorably with the cost of a local page copy, which is about 1 millisecond on this machine [Smith1989a].

2.2. Variability

Tests on a single file are rather unconvincing. To reinforce the measurements, we ran the following script in the root directory of the remotely-mounted file system:

```
$ find . -size +800 -print >/tmp/bigfiles
$ for i in ‘cat /tmp/bigfiles‘
> do
>  ~/netrand 100 ${i}
> done
```

(*find* uses a blocksize of 512 bytes, and we need 100 4K pages for the *netrand* algorithm to work) The results for the 24 files found were:

```
elapsed time, 2.26 seconds.
elapsed time, 2.92 seconds.
elapsed time, 2.74 seconds.
elapsed time, 3.2 seconds.
elapsed time, 2.98 seconds.
elapsed time, 3.1 seconds.
elapsed time, 3.02 seconds.
elapsed time, 2.44 seconds.
elapsed time, 2.94 seconds.
elapsed time, 2.46 seconds.
elapsed time, 2.74 seconds.
elapsed time, 3.62 seconds.
elapsed time, 4.22 seconds.
elapsed time, 3 seconds.
elapsed time, 3.34 seconds.
elapsed time, 2.94 seconds.
elapsed time, 3.02 seconds.
elapsed time, 2.86 seconds.
elapsed time, 2.96 seconds.
elapsed time, 2.68 seconds.
elapsed time, 3.2 seconds.
elapsed time, 2.56 seconds.
elapsed time, 3.1 seconds.
elapsed time, 2.78 seconds.
```

Thus, the average time was about 3 seconds, which for the 400 page-access case should imply a time of about 12 seconds. This indicates that *on average* the penalty for network page access is remarkably small. However, this may be a function of the server.

2.3. Heterogeneous Servers

A final question which occurs is the variability due to the server characteristics, in particular its CPU, Memory, Disk, and Network Interface speeds. Fortunately, there were several types of machines which were NFS servers on our network. Two of the more interesting points were a Sun-4 running SunOS™ 4.0.1 with 32 megabytes of main memory, and a DEC™ VAX 11/750 running 4.3BSD with 4 megabytes of main memory. A script of the form of the one given above was applied on the Sun-4, 89 files were found, and the results were:

```
elapsed time, 2.14 seconds.
elapsed time, 1.88 seconds.
elapsed time, 1.88 seconds.
elapsed time, 1.76 seconds.
elapsed time, 1.76 seconds.
```

```
elapsed time, 1.88 seconds.
elapsed time, 1.84 seconds.
elapsed time, 1.88 seconds.
elapsed time, 1.94 seconds.
elapsed time, 1.8 seconds.
elapsed time, 2.06 seconds.
elapsed time, 1.88 seconds.
elapsed time, 2.02 seconds.
elapsed time, 2.08 seconds.
elapsed time, 1.78 seconds.
elapsed time, 2.28 seconds.
elapsed time, 1.88 seconds.
elapsed time, 1.82 seconds.
elapsed time, 1.76 seconds.
      .
      (70 more lines)
```

The average response time was 2.11 seconds for the 100 page case, showing an average response time of 8.44 seconds under the assumption that we can scale up. This is rather interesting, since the page accesses from this server require less time than from the local disk (although the UNIX buffer cache mechanism is likely to be much more effective for typical access patterns on the local disk). This implies that a fast server is to be preferred to a slower local disk, when response time is considered.

The same script run on the VAX found 17 files, and produced:

```
elapsed time, 4.36 seconds.
elapsed time, 4.26 seconds.
elapsed time, 4.78 seconds.
elapsed time, 4.38 seconds.
elapsed time, 4.28 seconds.
elapsed time, 4.34 seconds.
elapsed time, 4.12 seconds.
elapsed time, 4.38 seconds.
elapsed time, 4.32 seconds.
elapsed time, 4.64 seconds.
elapsed time, 3.98 seconds.
elapsed time, 4.42 seconds.
elapsed time, 4.44 seconds.
elapsed time, 3.5 seconds.
elapsed time, 4.46 seconds.
elapsed time, 4.56 seconds.
elapsed time, 4.36 seconds.
```

This gives an average time of about 4.3 seconds for the 100 page case, which we can use to compute a 17.3 second average for the 400 page case. Thus, this older technology machine is slower than the local disk, and slower than a server of the same type as the workstation. Accesses to pages on this machine will take about twice as long as references to the local disk when demand paging is necessary.

The conclusion we can draw immediately from these measurements is: **Put your money into the server**. The fast server (SUN) provided performance improvements relative to the both the slower server (VAX) and the equivalent server (HP). More importantly, the fast server improved performance over a local disk. This is not to recommend any particular machine over any other; these machines span at least three generations of architectural development and the comparison between them is patently unfair. The important issue is their relative performance

and the effect it has on the measured response time of the attached computing systems.

3. Conclusions

The network response figures showed us that there is very little penalty for paging from a remote machine via a network, relative to paging from a local disk. In fact, for a very fast server, network accesses are faster than accesses to the local disk. This seems somewhat surprising; intuition says that disks are slow, and that networks are slower. But this is not what we've measured; in fact the network seems to add very little delay.

The cost in page references comes from delays inherent with the devices as well as copying. Many delays can be hidden by the use of intelligent caching, and higher-speed networks will further increase the attractiveness of computing systems connected by networks. Future architectures which use the network as a component of the storage hierarchy would be well-advised to reduce copying and other software overheads as much as possible. The right approach may well be one which integrates address resolution and paging with the network fabric; such an approach has been successfully explored in MEMNET [Delp1988a].

4. References

[Delp1988a] Gary Delp, Adarshpal Sethi, and David Farber, ''An analysis of Memnet: An experiment in high-speed shared-memory local networking,'' in *Proceedings,SIGCOMM'88 Symposium*, Stanford (August, 1988).

[Johnson1987a] Thomas D. Johnson, Jonathan M. Smith, and Eric S. Wilson, ''Disk Response Time Measurements,'' in *Proceedings, Winter 1987 USENIX Technical Conference*, Washington, DC (January, 1987), pp. 147-162.

[Organick1972a] Elliott I. Organick, *The Multics System,* Massachusetts Institute of Technology Press (1972).

[Sandberg1985a] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and R. Lyon, ''The Design and Implementation of the Sun Network File System,'' in *USENIX Proceedings* (June 1985), pp. 119-130.

[Sandberg1985b] R. Sandberg, *Sun Network File System Protocol Specification,* Sun Microsystems, Inc. (1985).

[Smith1986a] Jonathan M. Smith, ''Approaches to Distributed UNIX Systems,'' Technical Report CUCS-223-86, Columbia University Computer Science Department (1986).

[Smith1988a] Jonathan M. Smith and Gerald Q. Maguire,Jr., ''Effects of copy-on-write memory management on the response time of UNIX *fork* operations,'' *Computing Systems: The Journal of the USENIX Association* **1**(3), pp. 255-278, University of California Press (1988).

[Smith1989a] Jonathan M. Smith, ''Concurrent Execution of Mutually Exclusive Alternatives,'' Ph.D. Thesis, Columbia University Computer Science Department (May, 1989). also available from UMI

5. Appendix: netrand.c

```
#include <sys/types.h>
#include <sys/param.h>
#include <sys/times.h>

long times();
struct tms tbuf;

#include <stdio.h>

main( argc, argv )
int argc;
char *argv[];
{
   long start, diff;
   int count;
   FILE *fp;

   if( argc < 2 )
   {
      fprintf( stderr, "number required.\n" );
      exit( 1 );
   }
   count = atoi( argv[1] );
   if( count <= 0 )
      exit( 1 );
   if( argc > 2 )
   {
      fp = fopen( argv[2], "r" );
      if( fp == (FILE *) NULL )
      {
         perror( argv[1] );
         exit( 1 );
      }
   }
   else
      fp = stdin;

   start = times( &tbuf );

   random_read( fp, count );

   diff = times( &tbuf ) - start;
   printf( "elapsed time, %g seconds.\n", (float) diff / (float) HZ );
   exit( 0 );

}

random_read( fp, count )
FILE *fp;
int count;
{
   int i, fd, block_no, bytes;
   char page[NBPG_M320];
#ifdef DEBUG
   FILE *lfp;

   lfp = fopen( "/tmp/RANDBLOCKS", "w" );
   if( lfp == (FILE *) NULL )
```

```
      lfp = stdout;
#endif

   fd = fileno( fp );

   for( i = 0, block_no = random( times( &tbuf ) ) % count;
      i < count;
      block_no = random( 0 ) % count, i = i + 1 )
   {
      lseek( fd, block_no * sizeof(page), 0 );
      bytes = read( fd, page, sizeof(page));
#ifdef DEBUG
      fprintf( lfp, "block_no: %d, bytes read: %d\n",
         block_no, bytes );
   }
   fclose( lfp );
#else
   }
#endif

   return;
}

int
random( start )
int start;
{
   static int last_val;

   if( start != 0 )
      last_val = start;

   last_val = (17*last_val+123) % 65521;
   while( last_val < 0 )
      last_val += 65521;

   return( last_val );
}
```