SunOS 4.1 Performance Tuning

Hal L. Stern

Sun Microsystems
Northeast Area Consulting Group
55 Old Beford Road
Lincoln, MA 01773
(617) 259 2181

halstern@sun.com

Performance Tuning Myths

Unix (tm) can't be tuned

- Doesn't have exposed system parameters
- Hard to identify bottlenecks

Tuning requires source code

- Equating performance tuning and kernel hacking
- Kernel algorithms under scrutiny

Poor return on time investment

System vs. Application Tuning

Applications: Look for evidence of poor programming practices

- Eliminate inefficient constructs for major improvements
- Hand-crafted assembly code loops may only yield another 5%

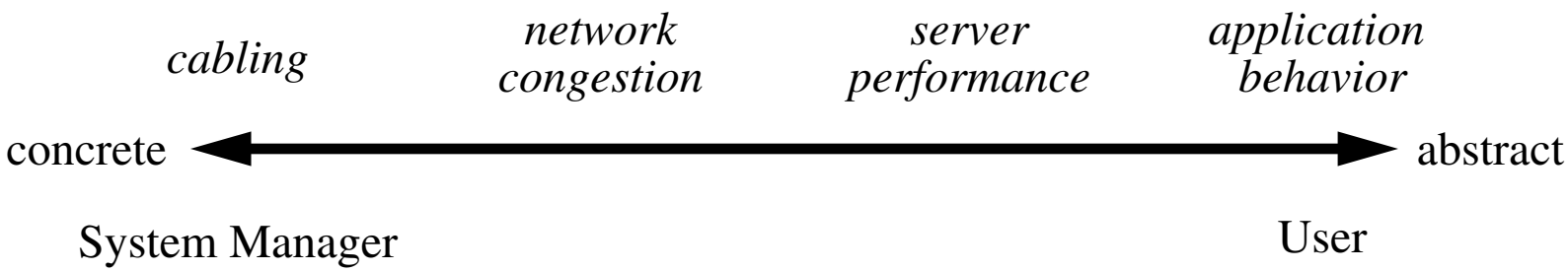
Computing Systems: Look for similar “bad habits”

- Focus on the structure of computing system, including desktops, network, and servers
- Defining “good habits” means making user and administration policies

Tuning in a Distributed Environment

Views of the computing environment

- System managers take global view
- Users have local view
- Are they the same?



Make It Close Enough For Jazz *

Law of Diminishing Returns

- Tuning to “perfection” doesn’t permit change or growth
- Performance analysis and tuning must be simple enough to be done regularly

Benchmarking

- Ensure that the quantity under test is the tightest constraint
- Avoid assumptions about linear scaling of problem sizes

Some Goals

- Adopt pro-active approach to computing systems management
- Reduce pain of typical peak usage periods
- Identify bottlenecks to guide incremental hardware purchases

* Nicholas Santoro, 1979

Tuning Suns and SunOS 4.1

- STREAMS system
- Network interfaces
- IP routing
- NFS strategies
- Application memory management
- Cache effects
- File I/O Optimization

New Features in SunOS 4.1

- Dynamic STREAMS data structure allocation
- NFS dynamic retransmission
- *tmpfs* swap-based filesystem
- *madvise*() virtual memory management hinting
- Asynchronous I/O
- Exposed configuration parameters

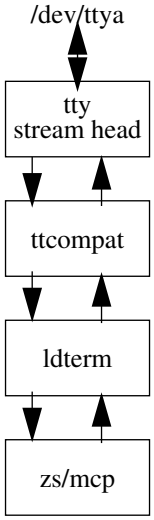
STREAMS System

tty and pseudo-tty drivers are STREAMS based

tty driver has several STREAMS modules

- ldterm for canonical input processing
- ttcompat for Berkeley style tty control

STREAMS messages contain message header and data blocks



STREAMS Input

Serial line devices interrupt at highest hardware priority

- Small input buffer requires rapid response
- Overrun of inputs masks lower priority inputs
    - Missed hardclock interrupts slows time-of-day clock
    - Ethernet packet drops

Significant input processing

- Newline handling, erase and kill editing
- Echo and case mapping
- Signal generation

Input Overruns

Data blocks are dynamically allocated

- Increased (but not infinite) headroom
- Capacity to handle bursts of input may mask problems
- High system CPU utilization shown by *vmstat* & friends

```
% vmstat 10
.... us sy id
..... 0 21 79
..... 1 18 81
```

Unused terminal lines and modem problems

- Shield cables or turn off *getty* processes on unused lines
- Error-correcting and parity protocol negotiation
- High speed modems

Input Notification

Applications typically do not handle asynchronous inputs well

- Multiplexing input and display updates is non-trivial
- Exposure to huge number of system calls and byte-sized I/O requests

Notification requirements

- Block on any character: Normal *read*() behavior
- Non-blocking read with idle loop processing: Asynchronous I/O
- Read at least N characters: Non-canonical input processing via **termio**

Asynchronous I/O

Replace polling loops built with *select*()

```
tfd = open("/dev/ttyh0", O_RDONLY);
while (...) {
     FD_SET(tfd, &readmask);
     select(fdwidth, &readmask, NULL, NULL, NULL);
     if (FD_ISSET(tfd, &readmask)) {
          ..handle character input...
          }
     ...other processing...
}
```

Asynchronous I/O completes operation in a thread

```
int got_chars();
tfd = open("/dev/ttyh0", O_RDONLY);
signal(SIGIO, got_chars);
aioread(tfd, charbuf, sizeof(charbuf), 0, 0, &aio_res);
while (...) {
     ...other processing...
}

int got_chars() {
     ...handle character input...
}
```

Ensure smooth interactive performance

Non-Canonical Input Processing

Avoid `FIONREAD` *ioctl*() because it walks message lists

Turn off special character handling for raw data sources

- Reset `ICANON` bit in `c_lflag` structure

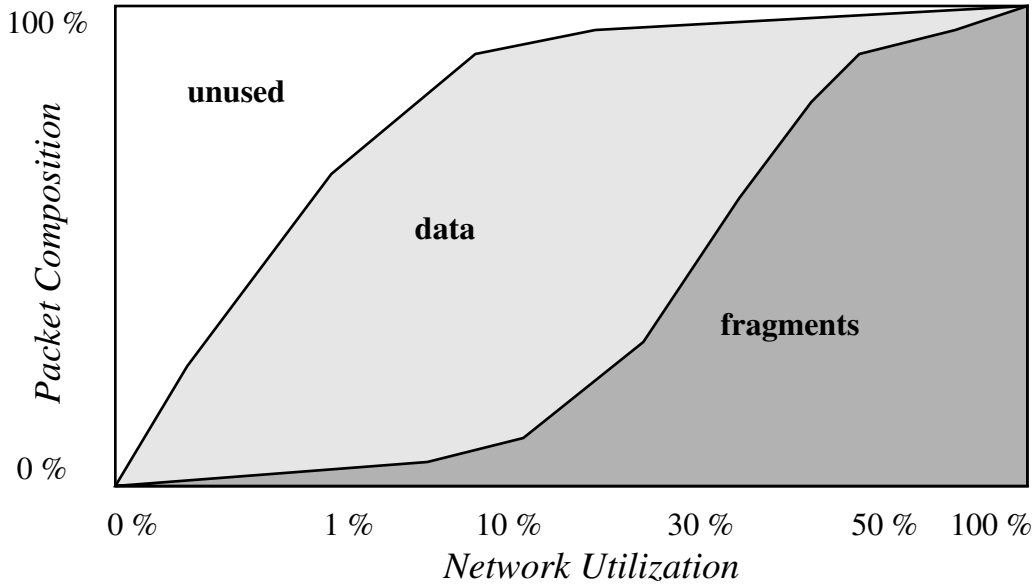
`MIN` and `TIME` **termio** parameters

- MIN > 0, TIME > 0: TIME is an inter-character timer
- MIN > 0, TIME = 0: Block on MIN characters in stream head
- MIN = 0, TIME > 0: TIME is input timer
- MIN = 0, TIME = 0: Non blocking read

Couple asynchronous I/O with non-canonical input processing for efficient burst mode handling

Network Capacity

Network utilization in burst-filled environment



Non-uniform collision rates

$$\frac{\sum_{i=0}^{N} collis_i}{\sum_{i=0}^{N} opkts_i} \neq \sum_{i=0}^{N} \frac{rate_i}{N}$$

Where Are Packets Lost?

On the wire

- Bridges or routers drop part of packet train

On the host

- Insufficient buffer space to receive packets
- CPU misses interrupts
- IP cannot queue packet for TCP or UDP socket

How Packets are Received

Ethernet chip copies packet into ethernet receive buffer

- Fixed number of buffers
- Driver cannot run out of buffers - it will drop packets first

Receive buffer wrapped in an `mbuf` and passed to IP

IP forwards packet if necessary or queues it for TCP/UDP socket

- TCP/UDP processing done by “fast” and “slow” handlers to improve efficiency
- Socket can run out of receive space

Ethernet Receive Buffers

Shortage of buffers shows up as input error rate > 0.025%

```
[sunne]% netstat -i
Name Mtu Net/Dest      Addres   Ipkts  Ierrs Opkts Oerrs Collis Queue
le0 1500 129.151.52.0 sunne    2242490 570 2814032 478   16386  0
lo0 1536 loopback   localhost 285548   0  285548 0       0  0
```

Causes

- Clients with faster ethernet interface than Sun-3 server
- Multiple network interfaces
- Electrical problems

Configuration files

- `/sys/sunif/ie_conf.c` Intel interfaces
- `/sys/sunif/le_conf.c` Lance interfaces

Buffers and Descriptors

Buffer and descriptors occupy about 1600 bytes

- `ie_rbufs` Receive buffers
- `ie_rbds` Buffer descriptors
- `ie_rfds` Frame descriptors

Constraints

- `ie_rbufs > ie_rbds > ie_rfds`
- Buffers and descriptors must fit in 125 Kbytes on 4/400 series
- High and low values

Protocol Receive Spaces

Socket overflows reported by `netstat -s`

```
[sunne]% netstat -s
udp:
     0 incomplete headers
     0 bad data length fields
     0 bad checksums
     75 socket overflows
```

Causes of overflows

- User process such as **nfsd** does not drain socket quickly enough
- **spray**

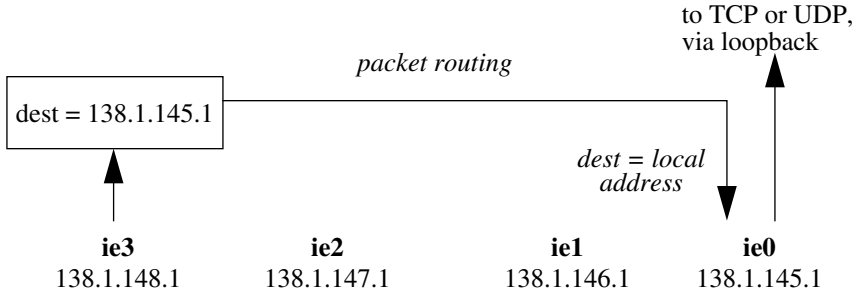
Increasing UDP receive space

- `/sys/netinet/in_proto.c` has protocol configuration parameters
- `udp_recvspace` is size of socket buffer

Multi-Homed Hosts

```
138.1.145.1       acadia-bb1 acadia   # ie0
138.1.146.1       acadia-bb2          # ie1
138.1.147.1       acadia-bb3          # ie2
138.1.148.1       acadia-bb4          # ie3
138.1.148.2       nfsclient
```

```
nfsclient# mount acadia:/home/acadia /home/acadia
```




```
nfsclient# mount acadia-bb4:/home/acadia /home/acadia
```

Routing Strategies

`in.routed` routing daemon

- Started by default out of `/etc/rc.local` on all diskfull systems
- Broadcasts route information every 30 seconds

Single-outlet routing

- With only one router on network, use default route

```
#if [ -f /usr/etc/in.routed ]; then
#    in.routed;echo -n ' routed'
#fi
route add default gatehost 1
```

Quiet mode routing

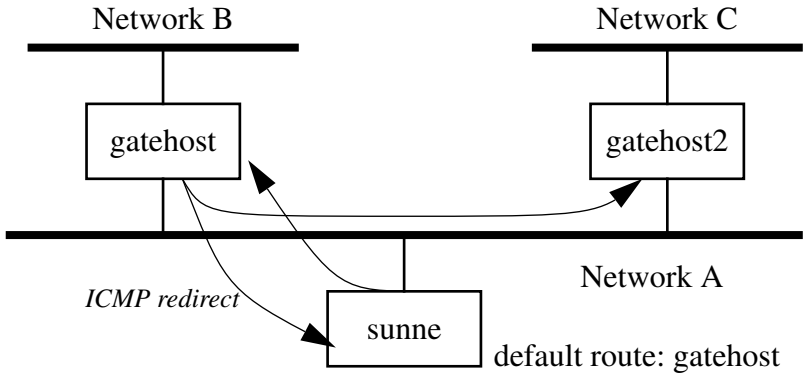
- Only broadcast routes from hosts that forward packets
- Other hosts build routing tables but don't advertise routes

```
in.routed -q
```

ICMP Redirect Messages

Sub-Optimal Routing

- Router may receive a packet that it must forward to host on the same network
- ICMP redirect message informs sender of better route



Suppressing redirect messages

- Hosts should update routing tables to reduce traffic level
- Some IP implementations don't handle ICMP redirect messages
- Turn off flag in `/sys/netinet/in_proto.c`

```
#define IPSENDREDIRECTS 0
```

NFS Performance Constraints

Congested networks

- Multiple-packet requests cannot reach server
- Collisions increase perceived server response time

Sluggish servers

- Remote Procedure Call timeouts cause retransmission of requests
- NFS clients look like virtual users

Mounting strategies

- Excessive symbolic link resolution
- Replicate frequently used filesystems

Lost Packets vs. Slow Servers

NFS clients cannot distinguish a slow server from a crashed server

No reply received within RPC timeout period if:

- Packets are lost on network
- Packets dropped at server's network interface
- Server takes too long to handle request

Differentiate between network and server problems using `nfsstat -rc`

```
[sunne]% nfsstat -rc
Client rpc:
calls badcalls retrans  badxid   timeout  wait  newcred timers
224978  487      64      265     549      0      0     696
```

timeout = RPC timeouts
retrans = retransmission count
badxid = duplicate responses from server
badcalls = RPC timeouts resulting in RPC error

`badxid` and `timeout` Interpretation

Request `xid` retained across retransmissions

`badxid` counts number of retransmitted requests successfully processed

- `badxid = timeout` implies that most requests are reaching server
- `badxid = 0` means retransmitted requests are dropped on the floor

`retrans` count affected by NFS dynamic retransmission algorithm

`badxid` count reduced by duplicate request cache

- Incoming request ignored if `xid` matches one in progress on server
- Eliminates redundant server operations

Accomodating Slow Servers

Increase RPC timeout period using `timeo` mount option

```
sunne:/home/sunne /home/sunne nfs bg,timeo=150 0 0
```

Don't increase to arbitrarily large value

- Keep as short as possible
- Use `nfsstat -m` as starting point
- Long timeout affects packets lost on network

Retransmission count

```
sunne:/home/sunne /home/sunne nfs bg,retrans=7 0 0
```

- RPC request is retransmitted `retrans` times
- Timeout period doubled between retransmissions
- After `retrans` attempts, operation fails on soft mounted filesystem
- Hard mounted filesystem doubles initial timeout and re-enters loop

Retransmission Count and Soft Mounts

Increased `retrans` count useful for soft-mounted filesystems

- Increasing count for hard-mounted filesystems may only generate more packets
- Can extend period before RPC call fails, if server is prone to slowdown or crashes

Only soft mount read-only data files

- Write operations may fail on soft-mounted writeable filesystem
- Virtual memory system pages executable images in directly from disk image
- RPC error during page in may hang system

`intr` option

- Use `intr` option to allow RPC retransmission loop to be broken

Network Bottlenecks

Server or network hardware ethernet interface may drop packets

- 8 Kbyte NFS buffer is 6 IP packets
- Not all devices can accept long packet trains

Decrease NFS buffer size

```
sunne:/home/sunne /home/sunne nfs bg,rsize=2048,wsize=2048 0 0
```

- Client sends/receives fewer IP packets for each NFS write/read operation
- Increase probability that complete requests arrive at server
- Recommended for using NFS over wide-area networks to compensate for smaller MTU
- Corresponding increase in server load: more RPCs and IP packets for same volume of data

Dynamic Retransmission

Dynamic retransmission enabled when client and server are on different networks

- Bridges don't trigger dynamic retransmission
- Anticipate bottlenecks at routers and gateways

Client sets timer based on expected server response time

- Timer expiration is feedback for decreasing NFS buffer size and increasing RPC timeouts
- Explicit mount options are taken as initial hints for timeout and buffer size

```
[sunne]% nfsstat -m
/home/thud from thud:/home/thud (Addr 129.151.52.13)
   Flags: hard read size=8192, write size=8192, count = 5
   Lookups: srtt=7 (17ms), dev=4 (20ms), cur=2 (40ms)
   Reads: srtt=14 (35ms), dev=3 (15ms), cur=3 (60ms)
   Writes: srtt=56 (140ms), dev=5 (25ms), cur=9 (180ms)
  All: srtt=7 (17ms), dev=4 (20ms), cur=2 (40ms)
```

Mount Point Strategies

NFS pathnames resolved one component at a time

- Many lookup operations
- One component might be a mount point for another filesystem
- Symbolic links in path are resolved on server "owning" them

Ensure server *inode* tables are sufficiently large

- Directory name lookup cache (DNLC) size based on `maxusers` parameter
- Increment `maxusers` for each NFS client

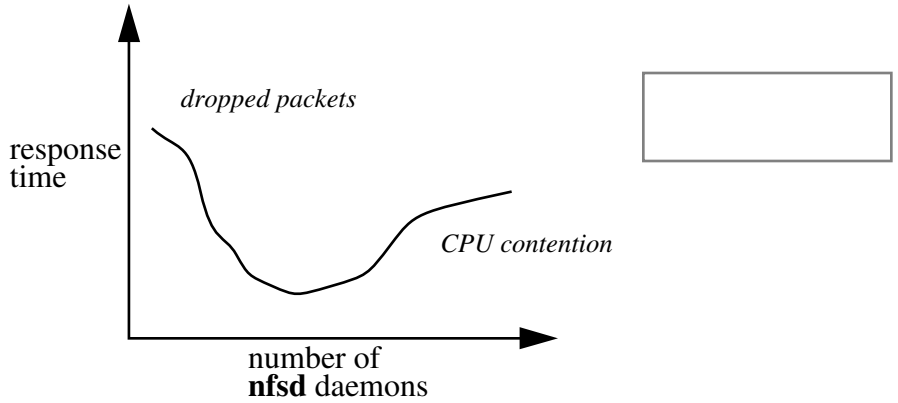
Replicate commonly used filesystems

- Use automounter to get NIS-like load balancing for replicated systems
- Examples: `man` pages, `/usr/local`

`nfsd` daemons

All daemons read from a single socket

- Requests drained from socket when `nfsd` daemon is scheduled and run
- Shortage of daemons may cause socket overflows



Too many daemons

- Increased load average
- Context switching/CPU contention

Symbolic Links

Examine server NFS demographics for symbolic link resolutions

```
[sunne]% nfsstat -ns

Server nfs:
calls  badcalls
1563212  41
 null   getattr   setattr root    lookup   readlink    read
 32 0% 527178 33% 9288 0% 0 0%  449726 28% 189466 12% 188665 15%
wrcache  write     create  remove    rename  link   symlink
 0 0%   134797 8% 13799 0% 15826 1%  2726 0% 4388 0% 74 0%
 mkdir   rmdir  readdir  fsstat
1575 0% 1532 0% 23898 1% 242 0%
```

Eliminate symbolic links near root of common pathname prefixes

- Replace link with a directory and mount filesystem on it
- Use automounter to enforce regular naming schemes

Swap Space Management

Variety of swap devices

- Raw partitions
- Local files
- NFS files

Multiple swap devices

- No explicit interleaving
- Use multiple devices if aggregate throughput is increased
- Two swap files on same disk may decrease swap performance
    - Two devices create two “swap queues”
    - Disk throughput is fixed, so each queue moves more slowly
    - Random elements cause queues to become uneven

tmpfs

tmpfs is a swap-based filesystem

- Uses same resources as processes
  - Memory to cache file pages
  - Swap space for paging out file pages

Files do not survive reboot

- Do not use it for `/var/tmp` or `vi` breaks

Not a RAM disk

- Size determined by size of swap space, not size of memory
- Files paged to disk as required

Using tmpfs

Replace any directory that is used for short-lived files

- /tmp is most common to improve compiler performance
- Explicitly mount or set up in fstab:

```
# mount -t tmp swap /usr/local/data
swap /usr/local/data tmp 0 0
```

Caveats

- Processes with large virtual address spaces use up swap space available for tmpfs
- Long-lived files belong in /var

Application Memory Management

Maximize hit rate to all caches

- MMU
- VAC
- I/O Cache

Memory management unit (MMU)

- Cache for page translations
- Sparse address spaces reduce effective size of cache

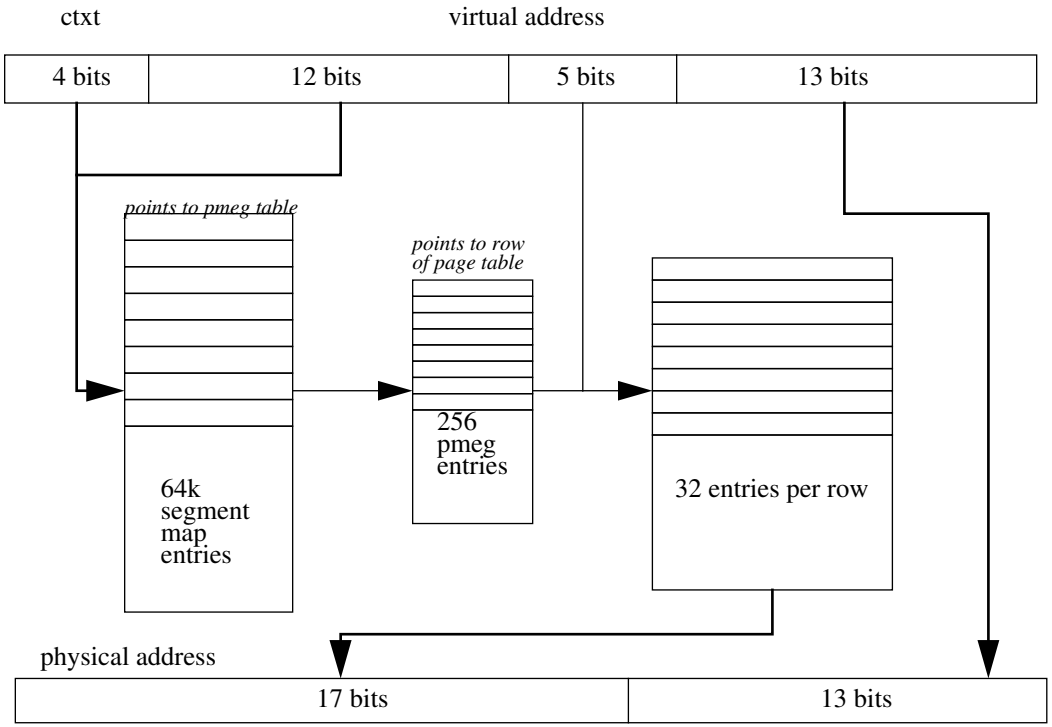
Virtual address cache (VAC)

- Cache sits between CPU and MMU
- Tagged cache accessed by virtual address
- Data stride will affect cache hit rate

SPARC MMU

Two-level MMU

- High order address bits select a segment map entry
- Segment map points to page map entry group (pmeg)
- pmeg points to 32 page table entries



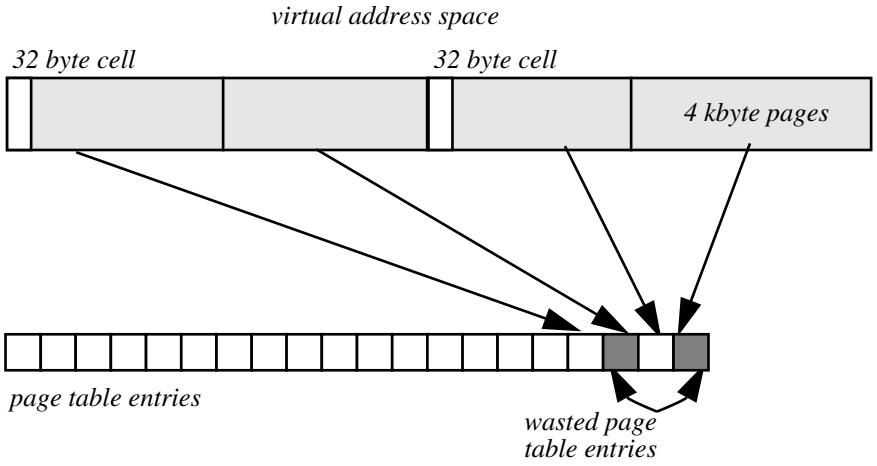
Sparse Address Spaces

Holes in address space make pmeg mappings less efficient

- pmeg still points to 32 consecutive virtual pages, but only some are used
- Reduces effective size of each pmeg

Causes

- Mixing `malloc()` and `valloc()`
- Alternately allocating and freeing buffers of varying sizes



Collapsing Sparse Address Spaces

Process size grows over time, even after repeated calls to `free()`

Help `malloc()` coalesce cells

- Define several commonly used cell sizes
- Allocate cells in multiples of some fixed size
- Adjacent free cells can be used for next larger size
- Avoid forcing page alignment

Data Structure Stride

Cache aliasing degrades performance

Aliasing occurs when related data structures have cache-sized strides between them

- Desktops: 64 Kbyte cache
- Servers: 128 Kbyte cache

Cache has 32 byte line size

- Bits 0 - 4 select a byte
- Bits 5 - 20 select a line from cache
- Addresses with bits 5-20 in common share a cache line

Skew data structure addresses to avoid aliasing

- Re-arrange global declarations
- Add 32 bytes to end of `malloc()` requests

An Example of Cache Aliasing

Examine global data structure addresses with `nm`

```
% nm a.out
00004098 B _stats
00044098 B _sample
```

Every reference to `stats` flushes a cache line

```
struct {
     int hi;
     int lo;
     int n; } stats = { 0, 999, 0 };

...other declaractions...

int sample;

while (...) {
     if (sample > stats.hi)
          stats.hi = sample;
     if (sample < stats.lo)
          stats.lo = sample;
     stats.n++;
}
```

Optimizing File I/O

Berkeley model

- Physical memory is a cache for process pages
- Kernel buffer pool is a cache for disk pages
- Disk pages copied into/out of process pages

SunOS 4.1 model

- Everything is page based: executables, files, process images, devices
- Physical memory is a cache for all pages
- Memory managed at the page level: copy on write, shared mappings
- Kernel can map in process pages: no more copying
- All I/O done through page faults

I/O Caching

Sun 4/400 and 3/400 servers have I/O caches

- I/O cache requires that buffers be aligned on 32-byte boundaries
- Kernel maps in user buffer: user buffer must be properly aligned

Forcing alignment

- `malloc()` does not guarantee alignment
- `memalign()` aligns to a given power-of-2 boundary
- `valloc()` aligns to page boundary

Sequential File I/O

Fairness vs. Performance

- Process doing sequential file I/O will use all available memory to cache file pages
- When free memory dips below minimum threshold, page daemon is scheduled to free pages in LRU fashion
- Pages used for sequential I/O are the “freshest” and least likely to be replaced
- Page replacement policy is driven by performance, not fairness

VM management hints

- `madvise()` supplies hints about the usefulness of an address range
- Can explicitly perform “free behind” during sequential I/O

`madvise()` Example

```
int fd;
char *fp;

fd = open("/home/thud/stern/bigfile", O_RDONLY);
fp = mmap(0, 1024 * 1024, PROT_READ, MAP_SHARED, fd, 0);


for (i = 0; i < 1024; i += 64) {
    /* hint to keep the "working set" in core */
    madvise(fp + i * 64*1024, 64 * 1024, MADV_SEQUENTIAL);
    ...sequential processing on 64 Kbytes of file ...

    /* hint to explicitly free the pages just used */
    madvise(fp + i * 64*1024, 64 * 1024, MADV_DONTNEED);
}
```

- `mmap()` maps file into process address space

  - Mapping is allocated but frames are not filled until page fault occurs

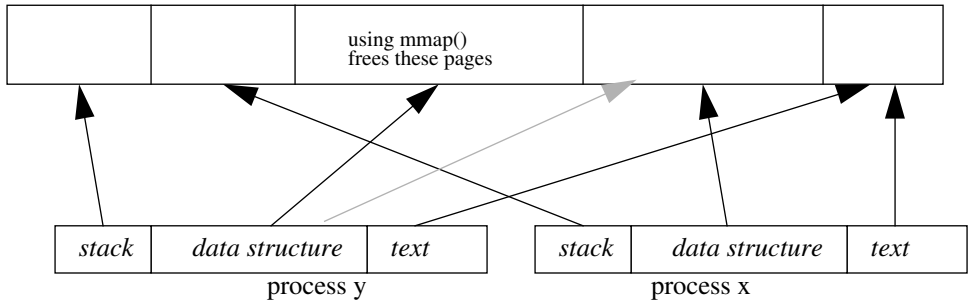
- First `madvise()` defines working set of pages to VM system

- Second `madvise()` frees them before next set of page faults

Shared File I/O

Large read-only files should be shared using `mmap()`

- Multiple instances of one application will map the same physical page frames
- Greatly reduce aggregate memory requirements
- Increase probability that most of file remains resident in memory



Some Thoughts

Performance analysis is best done pro-actively

- Educate user community about good *system* programming habits
- Identify problems before the next
  - QA build
  - Student project due date
  - pre-holiday crunch

Well-tuned system provides benefits to system administrators

- Focus on development and not fire-fighting

More Information

Protocol Tuning and Automounter

- SunOS System & Network Administration Guide

NFS Server Monitoring

- Dave Hitz: “System Administrator’s Performance Monitor for Tuning NFS Network Servers”

mmap()

- Kevin Sheehan: “MMAP as an Alternative to Traditional I/O”

Caches and SPARC MMU

- Neil Mammen: “SPARC MMU and Cache Tutorial”