
*October 12-14, 2004*

# Sun Solaris' UFS & NFS Performance CookBook




# Tuning Recipes for I/O Intensive Applications

Roch Bourbonnais

Performance and Availability Engineering
Grenoble, France

roch.bourbonnais@sun.com

# CookBook Tuning

- Recipes
  - Simple tuning recommendations
- Tell-Tale sign
  - Hopefully measurable sign indicating that tuning will be beneficial
- Stated Tradeoffs
  - What breaks
  - Additional resource requirements

# UFS & NFS CookBook

- Solaris 8, 9 Based
- NFSv3
- I/O Intensive
- Based on simple model for Disk, Storage, Volumes & Page Cache.

# Disk Subsystem

- In a steady state, the complexities of subsystem tend to disappear
- Simple Model
  - X IOPS (can go to 140 + i.e. 3 ms latency )
  - Y MB/s (can go to 40 MB/s)
- N disks wide

# Volume Management

- Stripe unit or interlace per disk
- Stripe width = N disks
- Full Stripe
  - Stripe Unit * Stripe Width
  - N * Interlace
- A Full Stripe is also the Smallest single I/O that may spread over all disks
- Interlace can govern per disk I/O size
  - Should be large enough to allow disk saturation (256K or 1M)
  - Not too large to allow spreading of I/O to multiple disks

# Filesystem

- 8 K blocksizes
- Maxcontig blocks laid sequentially
  - Lost its original purpose
    - that about rotational delays...
  - But still very important
    - Now used for *cluster* size computation
- A *cluster* is the I/O size used for large or sequential access patterns

# Filesystem

- The filesystem clustersize is set from the *maxcontig* parameter
- *Maxcontig* is expressed in FS pages (8K)
- Set with: Newfs -C *maxcontig*
- Dynamically tune with: Tunefs -a *maxcontig*
- Default based on device characteristics
- Solaris 8, 9 suffer of too small defaults requiring tuning.
- Desired value for all Solaris: 128 (1MB clusters)

# Page Cache Based Operations

- Clustering increases physical I/O sizes
- In UFS: Read Cluster == Write Cluster
- Readahead
  - Detected 2 sequential blocks -> full cluster read *ahead*
- Deferred Writes
  - Contiguous blocks form a cluster
  - I/O is deferred and initiated on a full cluster (or when contiguity is broken)

# 4 ways of Flushing Dirty pages

- 1) Under application control
  - Fsync, fdatasync
  - Open with flags: O_SYNC, O_DSYNC
  - Not to be confused with directio
    - Directio: performance optimisation attempted under certain conditions
    - fsync,etc: semantic implication about data integrity

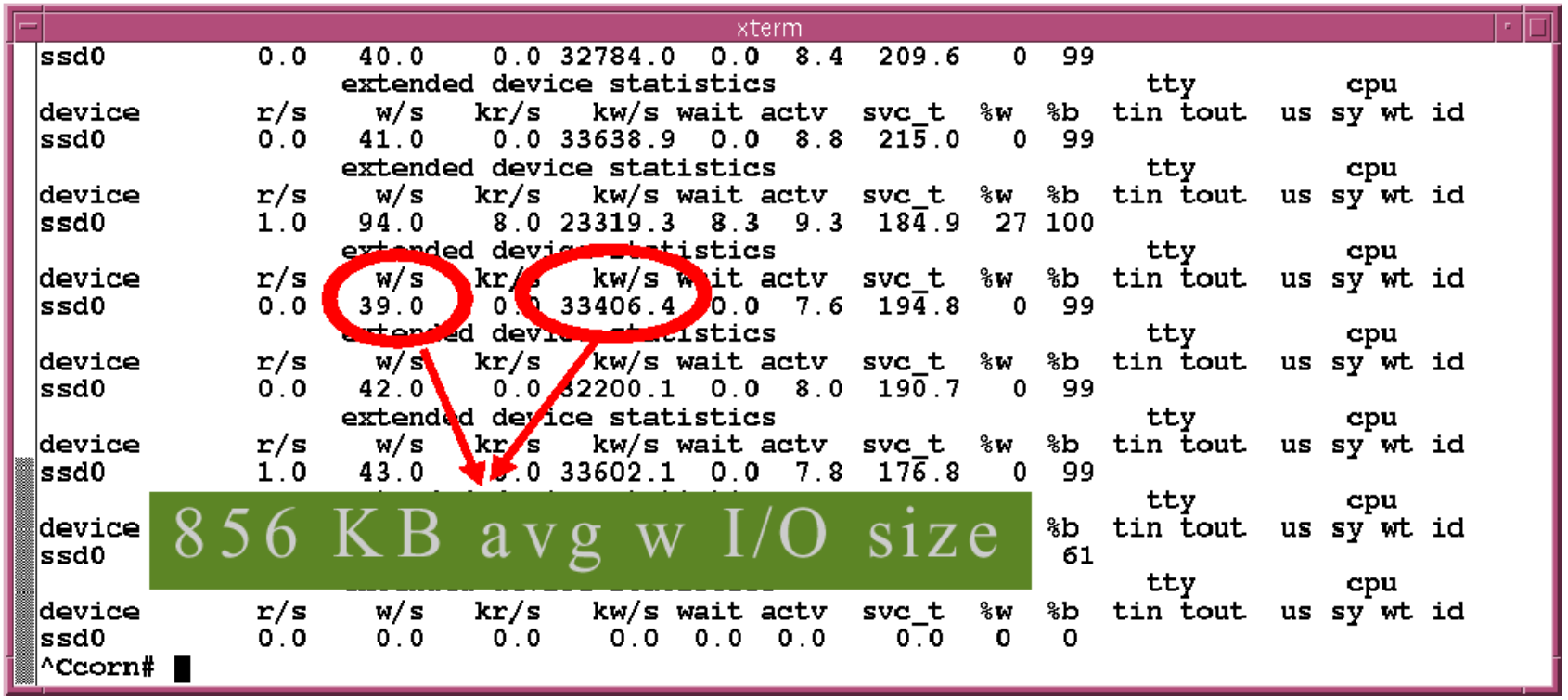
# 4 ways of Flushing Dirty pages

- 2) Cluster Fill up
  - Last Close of a file also sends the current cluster to disk
- 3) Fsflush
  - dirty pages flushed every autoup seconds
  - Fsflush runs every tune_t_fsflushr sec
  - Scans (autoup/tune_t_fsflushr) of total memory
  - Exercised by mmap I/O, random and sparse writes
- 4) Pageout
  - If and only if low memory

# Tools: iostat

- Iostat  xtc 1
  - kr/s, kw/s, r/s, w/s
    - Use it to compute avg I/O sizes
  - Per disk and per Volume
  - Shows BW or IOPS saturation
  - Bad disk distribution
- Discard first set of output (avg since boot)

# Tools: iostat



# Tools: iostat

- Iostat %b
  - 100% busy means that 100% of the time some I/O operation was in the pipe.
  - Not a good indicator of device saturation
- Iostat %w
  - The machine was idle while I/O was pending.
  - Iowait just means it possible to throw more work at the machine.
  - A major Spurious support call generator
    - The system is never waiting for I/O
    - Some application threads maybe

# Tools

- Vmstat -p 1
  - Free: in Solaris 8+, a meaningful value
  - sr, apo: indicator of memory shortage
- As always, discard first set of output

```
     memory            page          executable      anonymous      filesystem
   swap   free  re  mf  fr  de  sr  epi  epo  epf  api  apo  apf  fpi  fpo  fpf
 653104 253736   3  41   1   0   0    2    0    0    0    0    0   29    0    0
 664200 205296   2  27   0   0   0    0    0    0    0    0    0    0    0    0
```

# Tools

xterm

```
481488 17360   0   0   0   0   0   0   0   0   0   0   0   0   0   0
481488 17360   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    memory           page              executable      anonymous      filesystem
  swap  free   re  mf  fr  de  sr  epi  epo  epf  api  apo  apf  fpi  fpo  fpf
481488 17360   0   0   0   0   0   0   0   0   0   0   0   0   0   0
481488 17360   0   0   0   0   0   0   0   0   0   0   0   0   0   0
481488 17360   0   0   0   0   0   0   0   0   0   0   0   0   0   0
481488 17360   3   5   0   0   0   0   0   0  16   0   0   0   0   0
481488 17320   0   0   0   0   0   0   0   0   0   0   0   0   0   0
481488 17320   6 155   0   0   0   8   0   0   0   0   0   0   0   0
480384 94760   0   0   0   0   0   0   0   0   0   0   0   0   0   0
480376 62520   0   0   0   0   0   0   0   0   0   0   0   0   0   0
480376 38888   0   0 6232   0 2435   0   0  48   0   0  32   0 4008 6152
480528 18336   0   0 27264  0 13931  0   0  664   0 1280 1536   0 14224 25064
480528 17176   3   4 4520   0   0   8   0   0   0   0   0   0 4520 4520
480528 17208   0   4 4608   0   0   0   0   0  32   0   0   0 4608 4608
480504 17216   0   5 15456  0 3966   8   0  216  24  384  448   0 10688 14792
480504 16240   0   2 9920   0 660    0   0   16  16  128  144   0 9040 9760
480504 16848   0   0 7368   0   0   0   0   0   0   0   0   0 7368 7368
480504 15800   0   0 7424   0   0   0   0   0   0   0   0   0 7424 7424
480504 16696   0   1 12912  0 1618   0   0   16   8  128  128   0 11160 12768
480504 16424   0   0 6264   0   0   0   0   0   0   0   0   0 6264 6264
^Ccorn#
```

# Tools

- Truss: application behavior
- Mpstat: cpu imbalance
- Lockstat: contended locks
- Lockstat -I: contended cpu
- Netstat -i: monitor errors
- Nestat -s: tcp retransmits
- Kstat ce, ge etc...
  - Roch's Bytemeter

# Roch's Bytemeter

rbourbon@corn(274): bytemeter ce0

Outbound 0.000534 MB/s; Inbound 0.000000 MB/s
Outbound 61.935863 MB/s; Inbound 0.001239 MB/s
Outbound 62.364147 MB/s; Inbound 0.001295 MB/s
...

```
AWKSCRIPT='
NF == 0 {getline line;}
$1 == "obytes64" { obytes = $2; }
$1 == "rbytes64" { rbytes = $2; }
$1 == "snaptime" {
   time = $2;
   obytes_curr = obytes - prev_obytes;
   rbytes_curr = rbytes - prev_rbytes;
   elapse = (time - prev_time)*1e6;
   elapse = (elapse==0)?1:elapse;
   printf "Outbound %f MB/s; Inbound %f MB/s\n", obytes_curr/elapse, rbytes_curr/elapse;
   prev_obytes = obytes;
   prev_rbytes = rbytes;
   prev_time = time;
}
'
```

**Send Request: roch.bourbonnais@sun.com**

# S10
# The Dawn of Dtrace era

- Mother of all Tools coming soon to the Solaris world
- Dynamic Instrumentation of live kernel and application
- Flexible scripting language

# On the menu

- UFS throttling
- NFS daemons
- Stripping
- NFS blocksize
- Fsflush
- Freebehind
- Solaris Segmap subsystem
- Close to open

# Recipe ufs_HW

- Situation: GBs of data written to a file
- Tell tale Sign:
  - kernel variable ufs_throttles  increasing
  - As root: echo ufs_throttles/1D | mdb -k
- Fix: increase ufs_HW
- Memory Requirement
  - MAX(cluster,ufs_HW) * # active files
- UFS server, NFS server
- S9 default increased to 16MB (S8 at risk)

# Recipe NFSD

- Tell tale Sign:
  - #active nfs threads equals nfsd parameter
  - S8: echo $<threadlist | mdb -k  | grep svc_run | wc
  - S9+: pstack `pgrep nfsd` | grep nfssys | wc
- Fix:
  - S8: increase nfsd argument in /etc/init.d/nfs.server
  - S9: increase NFSD_SERVERS entry in /etc/default/nfs
- Memory Requirement, approximately
  - KMEM: 16K of kernel stack per ***active*** thread
  - KMEM: 32K (NFS block) of page cache
  - 1000 nfsd threads consume ~ 40MB (used and released on demand)
- Drawback:
  - S8: starvation of user's non-NFS related work
  - S9: nfs can be controled with Solaris Resource Management

# Recipe: Stripe #1

- Situation:
  - Applications doing lots of Large I/O (>>8k)
- Tell tale Sign:
  - Sustained disk activity, apps doing large read/write
- Fix:
  - stripe your volume, set UFS maxcontig and adjust maxphys, set interlace to 256K or 1MB
- Expected Throughput in ideal situation
  - Every disk BW saturated
- Memory Requirement
  - MAX (clustersz, ufs_HW) * "# active file"

# Recipe: Stripe #2

- Situation:
  - Many Small I/O (< 8k)
- FIX:
  - Recipe Stripe #1 applies if files are large and access sequential:
    - Readahead means clustersize access to files
  - Filesize may dictate I/O sizes
    - Bandwidth per disk ~= AVG file size / disk latency
  - You need application concurrency of at least the number of disks

# Recipe: Stripe #3

- **NFS:**
  - Client issue large (>>32K) read/write
  - Server shows 32K (nfs block) physical I/O sizes and keeps small number of disks active
- Why:
  - NFS requests are multi-threaded (client side) to nfs3_max_threads per mount point
  - NFSv3 uses 32K block size per thread
  - NFS Server threads often cannot coalesce multiple requests into larger I/O, and Server thus issue 32K Physical I/O sizes
  - Spindle may saturate at 32K/latency: underacheiving splindle
- Fix:
  - Increase stripe utilisation by increasing application concurrency
  - Increase load per client by increasing number of mounts
  - Increase I/O sizes by increasing NFS blocksize

# Recipe : nfs3_blocksize

- Situation:
  - Default NFS blocksize may throttle purely data intensive setup.
- Tell tale Sign:
  - Large client read/write call, but I/O sizes smaller than maxphys/maxcontig.
- Fix:
  - Increase the nfs blocksize
- Drawbacks:
  - Possibly not suitable for small I/O

# Recipe : nfs3_blocksize

- The problem is that large requests are broken up into blocks. Multiple blocks are then handled concurrently by the multiple nfs kernel threads (on the client).
- The requests are not ordered so that sequential nature of the I/O may be lost.
- Increasing the nfs blocksize; set kernel parameter
  - nfs3_bsize on client and
  - nfs3_max_transfer_size on client & server.
- Negotiate *down* (if necessary) the blocksize with mount's rsize,wsize option
  - Options not effective to negociate *up*

# Recipe autoup

- Situation:
  - CPU usage > 90%
- Tell tale Sign:
  - Cyclical (5 or 30 second) drops in business metric
- Fix:
  - increase autoup
  - autoup=300 is common value
- Drawbacks
  - file changes can be lost in case of system failure
  - Data integrity where critical should be managed with fsync() and friends

# Recipe : freebehind

- Situation:
  - Read files do not get cached
- Tell tale Sign:
  - Repeated sequential reading of file causes I/O each time
- Fix:
  - Keep freebehind only for very large files
    - Set smallfile to, for example, 1/8th of mem
    - Or Disable freebehind altogether (set freebehind=0)
- UFS or NFS server
- Drawbacks:
  - Caching large sequential read displaces *many* other small files.

# Recipe : segmap_percent

- Situation: Dedicated I/O server which has some free memory
- By default 12% of memory is segmap-ed
  - Segmap is a subset of cached pages
  - read,write from segmap-ed pages is faster and costs less cpu to access
- NFS Server or UFS
- Tell tale Sign:
  - Check kstat unix:0:segmap:get* 1
  - Segmap efficientcy: (get_reclaim + get_user) / getmap

# Recipe : segmap_percent

- Fix: Increase kernel parameter segmap_percent (default 12)
- Memory Requirements: segmap_percent of memory will not be in the freelist.
- Drawbacks:
  - segmap_percent = 100 is like priority_paging=0.
  - Doing I/O causes paging storm.
  - Pre-Solaris 7 behavior :-(
  - Must keep segmap_percent at reasonable value (e.g 50)

# Recipe : nocto

- Situation:
  - Apps does many open,write,close over NFS
- Tell tale Sign:
  - constant open/close activity; significant wait time
- Fix:
  - mount -F nfs -o nocto
- Drawbacks:
  - file changes not seen immediately on other clients
- NFS Client

# Recipe : gigabit ethernet CE (cassini)

- Do as ce driver version1.137
  - echo "interrupts=1;" > /platform/sun4u/kernel/drv/ce.conf
- Use Cassini helper thread all the time:
  - set ce:ce_cpu_threshold = 1;
  - set ce:ce_taskq_disable = 0; !ie. do not change default value.
- Increase the interrupt blanking parameter
  - ndd -set /dev/ce instance 0 !repeats for each ce instance
  - ndd -set /dev/ce rx_intr_time 30 !this now applies to instance 0
  - Note: increase per packet latency (~50 microseconds) in exchange for higher CPU efficientcy

# Follow-on Project

- NFS V4

# Part 4: Everyone's Favorite Snide Remark

# Recipe : RTFM

- Situation:
  - performance not as expected
- Tell tale Sign:
  - I'm lost
- Fix:
  - READ
- MEMORY Requirements :
  - you're allowed to take notes
- Drawbacks:
  - it takes time

# Conclusion

- Keep it simple
- Knowing about tuning is the simple part
- Designing a re-produceable test and measure of success is the tough part (left as exercise)
- The Job of Solaris engineering is to make tuning disappear. So re-evaluate tuning with each Solaris release.
- Please report if things don't work as you expected
  - roch.bourbonnais@sun.com

# References

- /etc/system variables
  - maxphys
  - autoup
  - tune_t_fsflushr
  - segmap_percent

- /etc/system variables
  - ufs:ufs_HW
  - ufs:freebehind
  - ufs:smallfile
  - ufs:ufs_throttles (readonly)
  - nfs:nfs3_max_threads
  - nfs:nfs3_bsize
  - nfs:nfs3_max_transfer_size

- Nfsd count
  - /etc/init.d/nfs.server
  - /etc/default/nfs
- Mount  F nfs -o nocto

- Maxcontig:
  - newfs -C or  tunefs -a

# References

- NFS Performance and Tuning Guide for Sun Hardware:
  - docs.sun.com/ab2/coll.28.24/NFSPERFTUNGU (needs major updating)
- Solaris Tunable Parameters Reference Manual:
  - docs.sun.com/ab2/coll.736.2/SOLTUNEPARAMREF:
- OOBP (Out Of the Box Performance Project)

  http://bigadmin.eng.sun.com:7777/bigadmin/content/perf_tuning/
- Jim Mauro, Richard McDougall Solaris Internals: Core Kernel Architecture
  - ISBN 0-13-022496-0 (C) Prentice Hall, 2000
- http://www.sean.de/Solaris/tune.html: Solaris - Tuning Your TCP/IP Stack
- Brent Callaghan, NFS Illustrated
  - ISBN 0-201-32570-5 (C) Addison-Wesley

# Have Nice Dinner