Linux Performance Tuning and Stabilization Tips

Yoshinori Matsunobu

Lead of MySQL Professional Services APAC
Sun Microsystems
Yoshinori.Matsunobu@sun.com

Table of contents

- Memory and Swap space management
- Synchronous I/O, Filesystem, and I/O scheduler
- Useful commands and tools
  - iostat, mpstat, oprofile, SystemTap, gdb

Random Access Memory

- The most important H/W component for RDBMS
- RAM access speed is much faster than HDD/SSD
  - RAM: -60ns
    - 100,000 queries per second is not impossible
  - HDD: -5ms
  - SSD: 100-500us
- 16-64GB RAM is now pretty common

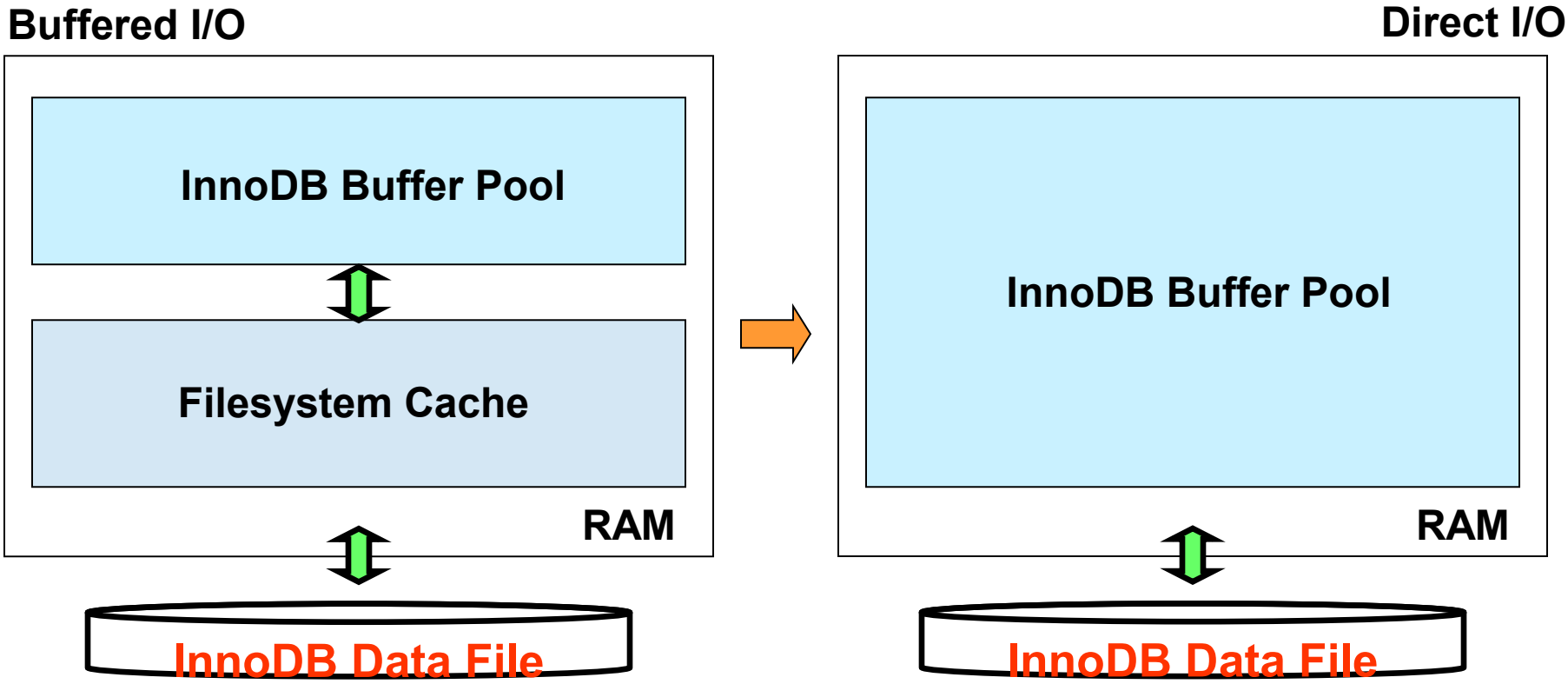
- *hot application data* should be cached in memory
- Minimizing hot application data size is important
  - Use compact data types (SMALLINT instead of VARCHAR/BIGINT, TIMESTAMP instead of DATETIME, etc)
  - Do not create unnecessary indexes
  - Delete records or move to archived tables, to keep hot tables smaller

Cache hot application data in memory

DBT-2 (W200)	Transactions per Minute	%user	%iowait
Buffer pool 1G	1125.44	2%	30%
Buffer pool 2G	1863.19	3%	28%
Buffer pool 5G	4385.18	5.5%	33%
Buffer pool 30G (All data in cache)	36784.76	36%	8%

- DBT-2 benchmark (write intensive)
- 20-25GB hot data (200 warehouses, running 1 hour)
- Nehalem 2.93GHz x 8 cores, MySQL 5.5.2, 4 RAID1+0 HDDs
- RAM size affects everything. Not only for SELECT, but also for INSERT/UPDATE/DELETE
  - INSERT: Random reads/writes happen when inserting into indexes in random order
  - UPDATE/DELETE: Random reads/writes happen when modifying records

Use Direct I/O



- Direct I/O is important to fully utilize Memory
- innodb_flush_method=O_DIRECT
- Alignment: File i/o unit must be a factor of 512 bytes
  - Can't use O_DIRECT for InnoDB Log File, Binary Log File, MyISAM, PostgreSQL data files, etc

Do not allocate too much memory

```
user$ top
Mem:  32967008k total, 32808696k used,   158312k free,    10240k buffers
Swap: 35650896k total,  4749460k used, 30901436k free,   819840k cached
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 5231 mysql     25   0 35.0g  30g  324 S  0.0 71.8   7:46.50 mysqld
```

- What happens if no free memory space is available?
  - Reducing filesystem cache to allocate memory space
  - Swapping process(es) to allocate memory space
- Swap is bad
  - Process spaces are written to disk (swap out)
  - Disk reads happen when accessing on-disk process spaces (swap in)
  - Massive random disk reads and writes will happen

What if setting swap size to zero?

- By setting swap size to zero, swap doesn't happen anymore. But..
  - Very dangerous
- When neither RAM nor swap space is available, OOM killer is invoked. OOM Killer may kill any process to allocate memory space
- The most memory-consuming process (mysqld) will be killed at first
  - It's abort shutdown. Crash recovery takes place on restart
  - Priority is determined by ORDER BY /proc/<PID>/oom_score DESC
  - Normally mysqld has the highest score
    - Depending on VMsize, CPU time, running time, etc
- It often takes very long time (minutes to hours) for OOM Killer to kill processes
  - We can't do anything until enough memory space is available

Do not set swap=zero

```
top - 01:01:29 up  5:53,  3 users,  load average: 0.66, 0.17, 0.06
Tasks: 170 total,   3 running, 167 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.0%us, 24.9%sy,  0.0%ni,75.0%id,0.2%wa,0.0%hi,  0.0%si,0.0%st
Mem:  32967008k total, 32815800k used,   151208k free,     8448k buffers
Swap:        0k total,        0k used,        0k free,   376880k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
26988 mysql     25   0   30g  30g 1452 R 98.5 97.7   0:42.18 mysqld
```

- If no memory space is available, OOM killer will be invoked
- Some CPU cores consume 100% system resources
  - 24.9% (average) = 1 / 4 core use 100% cpu resource in this case
  - Terminal freezed (SSH connections can't be established)
- Swap is bad, but OOM killer is much worse than swap

What if stopping OOM Killer?

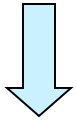
- If /proc/<PID>/oom_adj is set to -17, OOM Killer won't kill the process
  - Setting -17 to sshd is a good practice so that we can continue remote login
  - # echo -17 > /proc/<pid of sshd>/oom_adj
- But don't set -17 to mysqld
  - If over-memory-consuming process is not killed, Linux can't have any available memory space
  - We can't do anything for a long long time.. -> Long downtime

Swap space management

- Swap space is needed to stabilize systems
  - But we don't want mysqld swapped out
- What consumes memory?
  - RDBMS
    - Mainly process space is used (innodb_buffer_pool, key_buffer, sort_buffer, etc)
    - Sometimes filesystem cache is used (MyISAM files, etc)
  - Administration (backup, etc)
    - Mainly filesystem cache is used
- We want to keep mysqld in RAM, rather than allocating large filesystem cache

Be careful about backup operations

```
Mem:  32967008k total, 28947472k used,  4019536k free,   152520k buffers
Swap: 35650896k total,        0k used, 35650896k free,   197824k cached
PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 5231 mysql     25   0 27.0g  27g  288 S  0.0 92.6   7:40.88 mysqld
```



Copying 8GB datafile

```
Mem:  32967008k total, 32808696k used,   158312k free,    10240k buffers
Swap: 35650896k total,  4749460k used, 30901436k free,  8819840k cached
 PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 5231 mysql     25   0 27.0g  22g  324 S  0.0 71.8   7:46.50 mysqld
```

- Copying large files often causes swap

vm.swappiness = 0

```
Mem:  32967008k total, 28947472k used,  4019536k free,   152520k buffers
Swap: 35650896k total,        0k used, 35650896k free,   197824k cached
PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 5231 mysql     25   0 27.0g  27g  288 S  0.0 91.3   7:55.88 mysqld
```

Copying 8GB of datafile

```
Mem:  32967008k total, 32783668k used,   183340k free,     3940k buffers
Swap: 35650896k total,      216k used, 35650680k free,  4117432k cached
PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 5231 mysql     25   0 27.0g  27g  288 S  0.0 80.6   8:01.44 mysqld
```

- Set vm.swappiness=0 in /etc/sysctl.conf
  - Default is 60
- When physical RAM was fully consumed, Linux kernel reduces filesystem cache with high priority (lower swappiness increases priority)
- After no file system cache is available, swapping starts
  - OOM killer won't be invoked if large enough swap space is allocated. It's safer

Memory allocator

- mysqld uses malloc()/mmap() for memory allocation
- Faster and more concurrent memory allocator such as tcmalloc can be used
  - Install Google Perftools (tcmalloc is included)
    - # yum install libunwind
    - # cd google-perftools-1.5 ; ./configure --enable-frame-pointers; make; make install
  - export LD_PRELOAD=/usr/local/lib/tcmalloc_minimal.so; mysqld_safe &
- InnoDB internally uses its own memory allocator
  - Can be changed in InnoDB Plugin
    - If Innodb_use_sys_malloc = 1(default 1), InnoDB uses OS memory allocator
    - tcmalloc can be used by setting LD_PRELOAD

Memory allocator would matter for CPU bound workloads

	Default allocator	tcmalloc_minimal	%user	up
Buffer pool 1G	1125.44	1131.04	2%	+0.50%
Buffer pool 2G	1863.19	1881.47	3%	+0.98%
Buffer pool 5G	4385.18	4460.50	5.5%	+1.2%
Buffer pool 30G	36784.76	38400.30	36%	+4.4%

- DBT-2 benchmark (write intensive)
- Nehalem 2.93GHz x 8 cores, MySQL 5.5.2
- 20-25GB hot data (200 warehouses, running 1 hour)

Be careful about per-session memory

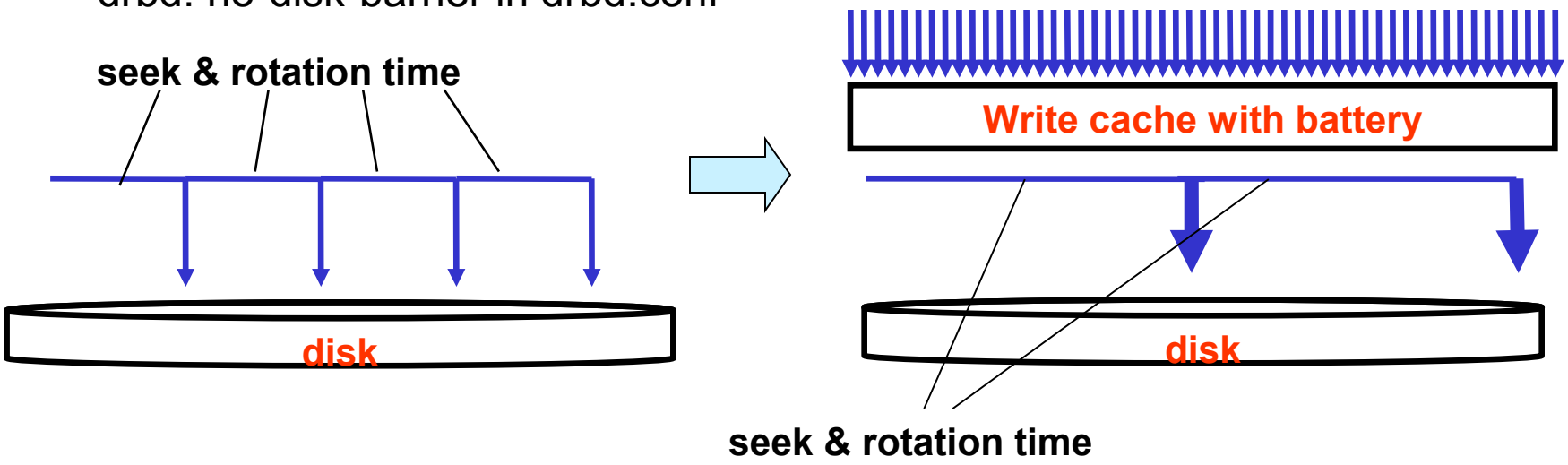
- Do not allocate much more memory than needed (especially for per-session memory)
- Allocating 2MB takes much longer time than allocating 128KB
  - Linux malloc() internally calls brk() if size <= 512KB, else calling mmap()
- In some cases too high per-session memory allocation causes negative performance impacts
  - SELECT * FROM huge_myisam_table LIMIT 1;
  - SET read_buffer_size = 256*1024; (256KB)
    - -> 0.68 second to run 10,000 times
  - SET read_buffer_size = 2048*1024; (2MB)
    - -> 18.81 seconds to run 10,000 times
- In many cases MySQL does not allocate per-session memory than needed. But be careful about some extreme cases (like above: MyISAM+LIMIT+FullScan)

Table of contents

- Memory and Swap space management
- Synchronous I/O, Filesystem, and I/O scheduler
- Useful commands and tools
  - iostat, mpstat, oprofile, SystemTap, gdb

File I/O and synchronous writes

- RDBMS calls fsync() many times (per transaction commit, checkpoints, etc)
- Make sure to use Battery Backed up Write Cache (BBWC) on raid cards
  - 10,000+ fsync() per second, without BBWC less than 200 on HDD
  - Disable write cache on disks for safety reasons
- Do not set “write barrier” on filesystems (enabled by default in some cases)
  - Write-through to disks even though BBWC is enabled (very slow)
  - ext3: mount -o barrier=0
  - xfs: mount -o nobarrier
  - drbd: no-disk-barrier in drbd.conf



Overwriting or Appending?

- Some files are overwritten (fixed file size), others are appended (increasing file size)
  - Overwritten: InnoDB Logfile
  - Appended: Binary Logfile

- Appending + fsync() is much slower than overwriting + fsync()
    - Additional file space needs to be allocated & file metadata needs to be flushed per fsync()
  - 10,000+ fsync/sec for overwriting, 3,000 or less fsync/sec for appending
    - Appending speed highly depends on filesystems
    - Copy-on-write filesystems such as Solaris ZFS is fast enough for appending (7,000+)
  - Be careful when using sync-binlog=1 for binary logs
    - Consider using ZFS
    - Check “preallocating binlog” worklog: WL#4925
  - Do not extend files too frequently
    - innodb-autoextend-increment = 20 (default 8)

Quick file i/o health check

- Checking BBWC is enabled, and write barrier is disabled
  - Overwriting + fsync() test
    - Run mysqlslap insert(InnoDB, single-threaded, innodb_flush_log_at_trx_commit=1), check qps is over 1,000

```
$ mysqlslap --concurrency=1 --iterations=1 --engine=innodb \
  --auto-generate-sql --auto-generate-sql-load-type=write \
  --number-of-queries=100000
```

Buffered and asynchronous writes

- Some file i/o operations are not direct i/o, not synchronous
  - file copy, MyISAM, mysqldump, innodb_flush_log_at_trx_commit=2, etc
- Dirty pages in filesystem cache needs to be flushed to disks in the end
  - pdflush takes care of it, maximum 8 threads
- When? -> highly depending on vm.dirty_background_ratio and vm.dirty_ratio
  - Flushing dirty pages starts in background after reaching dirty_background_ratio * RAM (Default is 10%, 10% of 64GB is 6.4GB)
  - Forced flush starts after reaching dirty_ratio * RAM (Default is 40%)
- Forced, and burst dirty page flushing is problematic
  - All buffered write operations become synchronous, which hugely increase latency
- Do flush dirty pages aggressively
  - Execute sync; while doing massive write operations
  - Reduce vm.dirty_background_ratio
  - Upgrade to 2.6.32 or higher
    - pdflush threads are allocated per device. Flushing to slow devices won't block other pdflush threads

Filesystem – ext3

- By far the most widely used filesystem
- But not always the best
- Deleting large files takes long time
  - Internally has to do a lot of random disk i/o (slow on HDD)
  - In MySQL, if it takes long time to DROP table, all client threads will be blocked to open/close tables (by LOCK_open mutex)
  - Be careful when using MyISAM, InnoDB with innodb_file_per_table, PBXT, etc
- Writing to a file is serialized
  - Serialized by “i-mutex”, allocated per i-node
  - Sometimes it is faster to allocate many files instead of single huge file
  - Less optimized for faster storage (like PCI-Express SSD)
- Use “dir_index” to speed up searching files
- Use barrier=0 to disable write-through

Filesystem – xfs/ext2

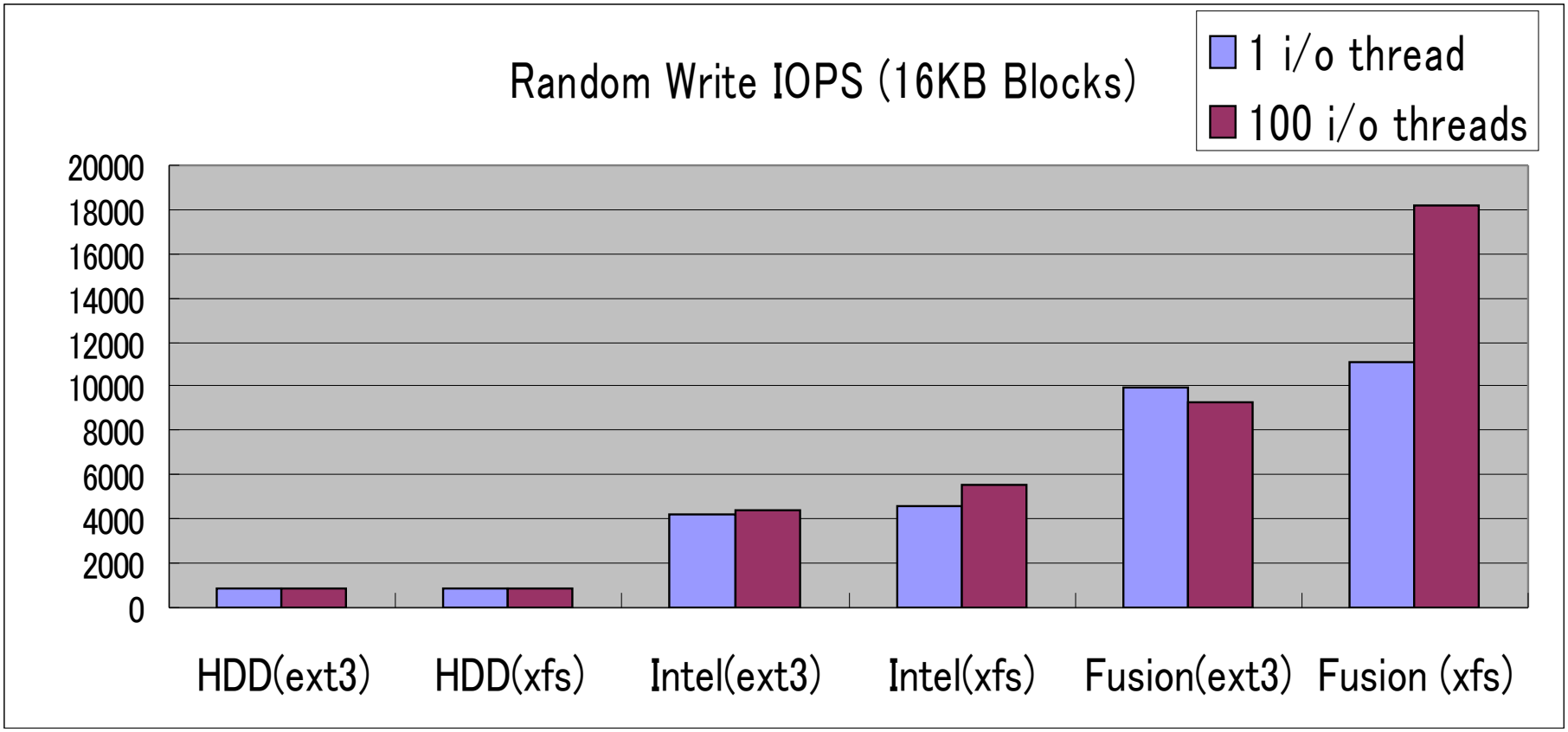
- xfs
  - Fast for dropping files
  - Concurrent writes to a file is possible when using O_DIRECT
  - Not officially supported in current RHEL (Supported in SuSE)
  - Disable write barrier by setting “nobarrier”

- ext2
  - Faster for writes because ext2 doesn’t support journaling
  - It takes very long time for fsck
  - On active-active redundancy environment (i.e. MySQL Replication),

    in some cases ext2 is used to gain performance

- Btrfs (under development)
  - Copy-on-write filesystem
  - Supporting transactions (no half-block updates)
  - Snapshot backup with no overhead

Concurrent write matters on fast storage



- Negligible on HDD (4 SAS RAID1)
- 1.8 times difference on Fusion I/O

I/O scheduler

- Note: RDBMS (especially InnoDB) also schedules I/O requests so theoretically Linux I/O scheduler is not needed

- Linux has I/O schedulers
  - to efficiently handle lots of I/O requests
  - “I/O scheduler type” and “Queue Size” matters

- Types of I/O schedulers (introduced in 2.6.10: RHEL5)
  - noop: Sorting incoming i/o requests by logical block address, that’s all
  - deadlilne: Prioritize read (sync) requests rather than write requests (async) to some extent (to avoid “write-starving-reads” problem)
  - cfq(default): Fairly scheduling i/o requests per i/o thread
  - anticipatory: Removed in 2.6.33 (bad scheduler. Don’t use it)

- Default is cfq, but noop / deadline is better in many cases
  - # echo noop > /sys/block/sdX/queue/scheduler

cfq madness

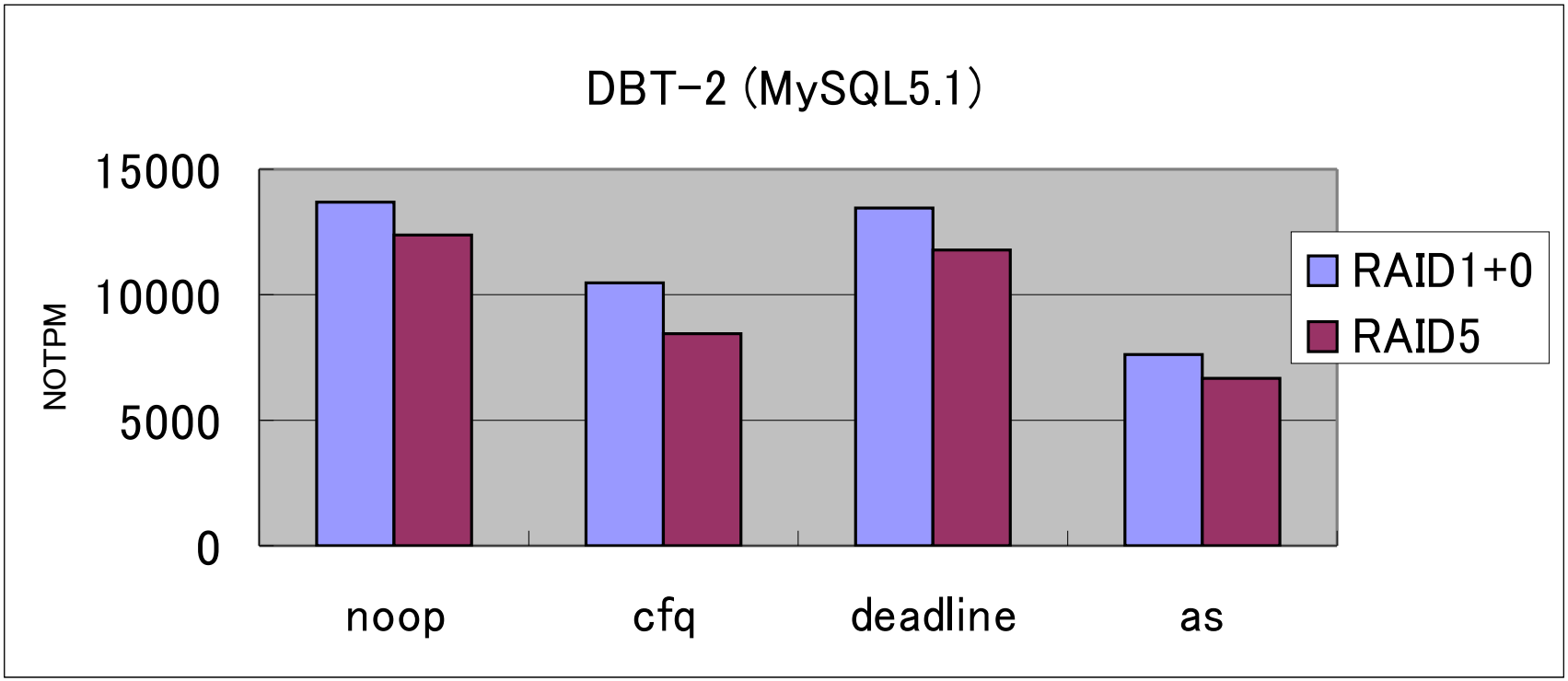
Running two benchmark programs concurrently

1. **Multi-threaded random disk reads (Simulating RDBMS reads)**
2. **Single-threaded overwriting + fsync() (Simulating redo log writes)**

Random Read i/o threads	write+fsync() running	Scheduler	reads/sec from iostat	writes/sec from iostat
1	No	noop/deadline	260	0
		cfq	260	0
100	No	noop/deadline	2100	0
		cfq	2100	0
1	Yes	noop/deadline	212	14480
		cfq	248	246
		noop/deadline	1915	12084
		cfq	2084	0

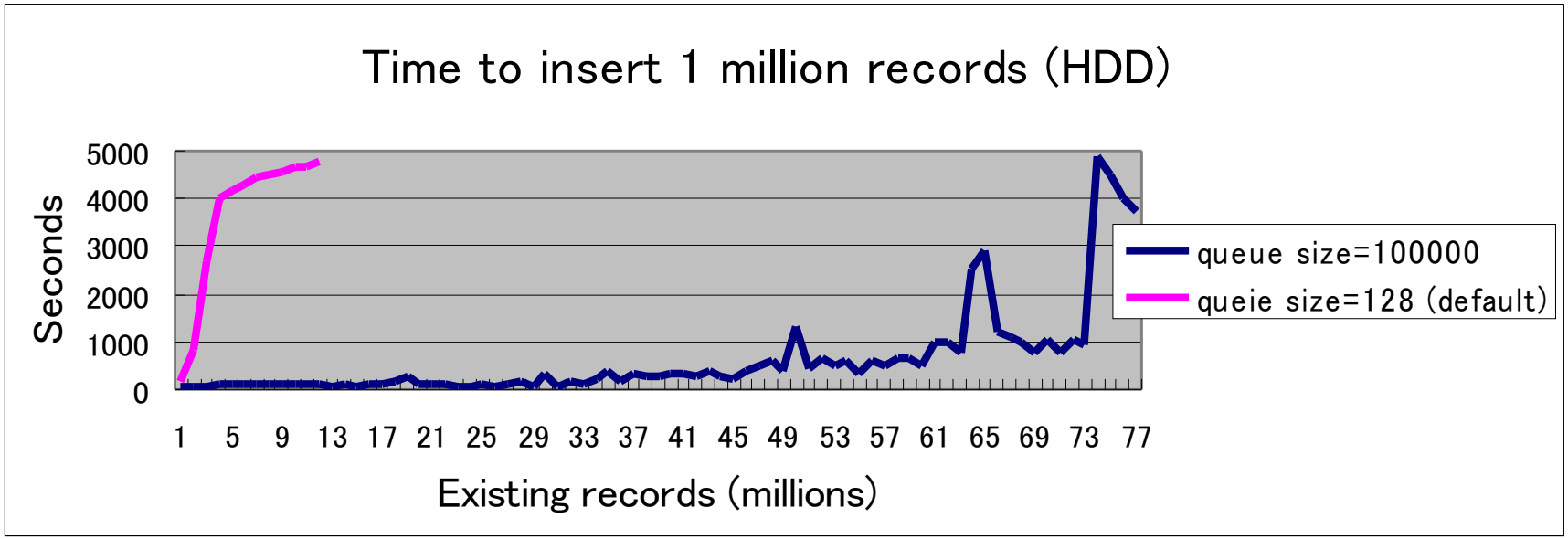
- In RDBMS, write IOPS is often very high because HDD + write cache can handle thousands of transaction commits per second (write+fsync)
- Write iops was adjusted to per-thread read iops in cfq, which reduced total iops significantly
- Verified on RHEL5.3 and SuSE 11, Sun Fire X4150, 4 HDD H/W RAID1+0

Changing I/O scheduler (InnoDB)



- Sun Fire X4150 (4 HDDs, H/W RAID controller+BBWC)
- RHEL5.3 (2.6.18-128)
- Built-in InnoDB 5.1

Changing I/O scheduler queue size (MyISAM)



- Queue size = N
  - Sorting N outstanding I/O requests to optimize disk seeks
- MyISAM does not optimize I/O requests internally
  - Highly depending on OS and storage
  - When inserting into indexes, massive random disk writes/reads happen
- Increasing I/O queue size reduces disk seek overheads
  - # echo 100000 > /sys/block/sdX/queue/nr_requests
- No impact in InnoDB
  - Many RDBMS including InnoDB internally sort I/O requests

Useful commands and tools

- iostat
- mpstat
- oprofile
- SystemTap (stap)
- gdb

iostat

- Detailed I/O statistics per device
- Very important tool because in most cases RDBMS becomes I/O bound
- iostat -x
- Check r/s, w/s, svctm, %util
  - IOPS is much more important than transfer size
- Always %util = (r/s + w/s) * svctm (hard coded in the iostat source file)

```
# iostat -xm 10
avg-cpu: %user %nice %system %iowait %steal %idle
         21.16  0.00    6.14   29.77   0.00 42.93
Device: rqm/s wrqm/s     r/s   w/s rMB/s wMB/s avgrq-sz avgqu-sz await svctm %util
   sdb   2.60 389.01  283.12 47.35  4.86  2.19    43.67      4.89 14.76  3.02 99.83
```

(283.12+47.35) * 3.02(ms)/1000 = 0.9980 = 100% util

iostat example (DBT-2)

```
# iostat -xm 10
avg-cpu: %user %nice %system %iowait %steal %idle
         21.16  0.00    6.14   29.77   0.00 42.93
Device: rqm/s wrqm/s    r/s    w/s rMB/s wMB/s avgrq-sz avgqu-sz await svctm %util
   sdb   2.60 389.01 283.12  47.35  4.86  2.19    43.67     4.89 14.76  3.02 99.83
```

(283.12+47.35) * 3.02(ms)/1000 = 0.9980 = 100% util

```
# iostat -xm 10
avg-cpu: %user %nice %system %iowait %steal %idle
         40.03  0.00   16.51   16.52   0.00 26.94
Device: rrqm/s wrqm/s    r/s    w/s rMB/s wMB/s avgrq-sz avgqu-sz await svctm %util
   sdb    6.39 368.53 543.06 490.41  6.71  3.90    21.02     3.29  3.20  0.90 92.66
```

(543.06+490.41) * 0.90(ms)/1000 = 0.9301 = 93% util

- Sometimes throughput gets higher even though %util reaches 100%
  - Write cache, Command Queuing, etc
- In both cases %util is almost 100%, but r/s and w/s are far different
- Do not trust %util too much
- Check svctm rather than %util
  - If your storage can handle 1000 IOPS, svctm should be less than 1.00 (ms) so you can send alerts if svctm is higher than 1.00 for a couple of minutes

mpstat

- Per CPU core statistics
- vmstat displays average statistics
- It's very commom that only one of CPU cores consumes 100% CPU resources
  - The rest CPU cores are idle
  - Especially applies to batch jobs
- If you check only vmstat/top/iostat/sar you will not notice single threaded bottleneck
- You can also check network bottlenecks (%irq, %soft) from mpstat
  - vmstat counts them as %idle

vmstat and mpstat

```
# vmstat 1
...
procs -----------memory---------- ---swap-- -----io---- --system-- -----cpu------
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 0  1 2096472 1645132  18648  19292    0    0  4848     0 1223  517  0  0 88 12  0
 0  1 2096472 1645132  18648  19292    0    0  4176     0 1287  623  0  0 87 12  0
 0  1 2096472 1645132  18648  19292    0    0  4320     0 1202  470  0  0 88 12  0
 0  1 2096472 1645132  18648  19292    0    0  3872     0 1289  627  0  0 87 12  0
```

```
# mpstat -P ALL 1
...
11:04:37 AM  CPU    %user   %nice    %sys %iowait    %irq   %soft  %steal  %idle   intr/s
11:04:38 AM  all    0.00    0.00    0.12   12.33    0.00    0.00    0.00   87.55 1201.98
11:04:38 AM    0    0.00    0.00    0.00    0.00    0.00    0.00    0.00  100.00  990.10
11:04:38 AM    1    0.00    0.00    0.00    0.00    0.00    0.00    0.00  100.00    0.00
11:04:38 AM    2    0.00    0.00    0.00    0.00    0.00    0.00    0.00  100.00    0.00
11:04:38 AM    3    0.00    0.00    0.00    0.00    0.00    0.00    0.00  100.00    0.00
11:04:38 AM    4    0.99    0.00    0.99   98.02    0.00    0.00    0.00    0.00  206.93
11:04:38 AM    5    0.00    0.00    0.00    0.00    0.00    0.00    0.00  100.00    0.00
11:04:38 AM    6    0.00    0.00    0.00    0.00    0.00    0.00    0.00  100.00    4.95
11:04:38 AM    7    0.00    0.00    0.00    0.00    0.00    0.00    0.00  100.00    0.00
```

- vmstat displays average statistics. 12% * 8 (average) = 100% * 1 + 0% * 7

Oprofile

- Profiling CPU usage from running processes
- You can easily identify which functions consume CPU resources
- Supporting both user space and system space profiling
- Mainly used by database-internal developers
- If specific functions consume most of recourses, applications might be re-designed to skip calling them
- Not useful to check low-CPU activities
  - I/O bound, mutex waits, etc

- How to use
  - opcontrol --start (--no-vmlinux)
  - benchmarking
  - opcontrol --dump
  - opcontrol --shutdown
  - opreport -l /usr/local/bin/mysqld

Oprofile example

```
# opreport -l /usr/local/bin/mysqld
samples  %        symbol name
83003     8.8858  String::copy(char const*, unsigned int, charset_info_st*,
charset_info_st*, unsigned int*)
79125     8.4706  MYSQLparse(void*)
68253     7.3067  my_wc_mb_latin1
55410     5.9318  my_pthread_fastmutex_lock
34677     3.7123  my_utf8_uni
18359     1.9654  MYSQLlex(void*, void*)
12044     1.2894  _ZL15get_hash_symbolPKcjb
11425     1.2231
_ZL20make_join_statisticsP4JOINP10TABLE_LISTP4ItemP16st_dynamic_array
```

- You can see quite a lot of CPU resources were spent for character conversions (latin1 <-> utf8)
- Disabling character code conversions on application side will improve performance (20% in this case)

```
samples  %        symbol name
83107    10.6202  MYSQLparse(void*)
68680     8.7765  my_pthread_fastmutex_lock
20469     2.6157  MYSQLlex(void*, void*)
13083     1.6719  _ZL15get_hash_symbolPKcjb
12148     1.5524  JOIN::optimize()
11529     1.4733
_ZL20make_join_statisticsP4JOINP10TABLE_LISTP4ItemP16st_dynamic_array
```

SystemTap

- SystemTap provides a simple command line interface and scripting language for writing instrumentation for a live running kernel (and applications).

- Similar to DTrace
- SystemTap script runs as a Linux Kernel module
- No-need to rebuild applications to profile
  - kernel-header/devel, kernel-debuginfo packages are needed
- Supported in RHEL 5 by default (5.4 is more stable than older versions, but be careful about bug reports)

- User level functions can be profiled if a target program has DWARF debugging symbols
  - MySQL official binary has DWARF symbols, so you do not need to rebuild mysqld
  - Add -g if you build MySQL by yourselves

- You can write custom C code inside a SystemTap script, but it's limited
  - This is called "guru" mode
  - Easily causes kernel panic. Be extremely careful
  - Since it is kernel module, user-side libraries can not be used

SystemTap use-case 1 : Per-file i/o statistics

```
# filestat 10
2010-04-05 11:18:55
     iotime      r/s      w/s   rBytes/s   wBytes/s   file
      6.12s    182.4      1.6      2.85M      30.4K   /hdd/data/dbt2/stock.ibd
      2.36s     64.1      0.7      1.00M      11.2K   /hdd/data/dbt2/customer.ibd
      1.05s     25.7      6.1     411.2K     100.8K   /hdd/data/dbt2/orders.ibd
    826.6ms     15.1      0.5     241.6K       9.6K   /hdd/data/dbt2/order_line.ibd
    645.1ms     16.6      2.1     265.6K     107.2K   /hdd/data/dbt2/new_order.ibd

2010-04-05 11:19:05
     iotime      r/s      w/s   rBytes/s   wBytes/s   file
      4.76s    173.9      0.6      2.72M      12.8K   /hdd/data/dbt2/stock.ibd
      2.22s     68.1      2.3      1.06M      36.8K   /hdd/data/dbt2/customer.ibd
      1.23s     18.4      1.2     294.4K      25.6K   /hdd/data/dbt2/order_line.ibd
    919.6ms     14.3     11.0     228.8K     521.6K   /hdd/data/dbt2/new_order.ibd
...
```

- iostat provides per-device i/o statistics, iotop provides per-process i/o statistics
  - Not enough for mysqld

Sample Code

```
probe syscall.read, syscall.pread {
 if (execname() == “mysqld”) {
  readstats[pid(), fd] <<< count
  fdlist[pid(), fd] = fd
 }
}
probe timer.s($1) {
 foreach ([pid+, fd] in fdlist) {
  reads=@count(readstats[pid, fd])
  rbytes=@sum(rdbs[pid, fd])
  print  “%d %d %d\n”, fd, reads, rbytes
...
```

```
#!/bin/sh
stap  filestat.stap 10 | perl sum.pl
```

- Programming within SystemTap is possible, but difficult
  - Most of utility libraries can not be used
  - limited to 1000 statements per probe
- Typical coding style:
  - Print raw statistical information (i.e. file descriptor, iotime, reads, writes, bytes-read, bytes-written, etc) to STDOUT
  - Pipe to Perl script (or python/ruby/etc)
  - Filtering/Grouping/Sorting/Decorating etc in Perl

SystemTap use-case 2 : Userspace profiling

```
mysql> EXPLAIN SELECT user_id, post_date, title
    -> FROM diary ORDER BY rating DESC limit 100\G
*********
 select_type: SIMPLE
 table:       diary
 type:        ALL
 key:         NULL
 rows:        1163
 Extra:       Using filesort

mysql> SELECT user_id, post_date, title
    -> FROM diary ORDER BY rating DESC limit 100;
...
100 rows in set (0.73 sec)
```

```
[root #] stap sort.stp
# of returned rows sorted by old algorithm: 0
# of returned rows sorted by new algorithm: 100
```

Background: MySQL Sorting Algorithm

- MySQL has two sorting algorithms (old algorithm / new algorithm)
- Choosing either of the two, depending on column length, data types, etc..
- Currently there is no MySQL status variable to check which algorithm is used
- Sometimes performance difference is huge (especially when used with LIMIT)
- Inside MySQL, rr_from_pointers() is called by old algorithm, rr_unpack_from_buffer() by new algorithm

Old algorithm

1) Load into sort buffer

rating	RowID
4.71	1
3.32	2
4.10	3
4.50	4

2) Sort

rating	RowID
4.71	1
4.50	4
4.10	3
3.32	3

3) Fetch the rest columns

user_id	post_date	rating	title
100	2010-03-29	4.71	UEFA CL: Inter vs Chelsea
2	2010-03-30	3.32	Denmark vs Japan, 3-0
3	2010-03-31	4.10	MySQL Administration
10	2010-04-01	4.50	Linux tuning

New algorithm

1) Load all columns into sort buffer

user_id	post_date	rating	title
100	2009-03-29	4.71	UEFA CL: Inter vs Chelsea
2	2009-03-30	3.32	Denmark vs Japan, 3-0
3	2009-03-31	4.10	MySQL Administration
10	2009-04-01	4.50	Linux tuning

2) Sort

SystemTap Script 2

```
global oldsort=0;
global newsort=0;

probe process("/usr/local/bin/mysqld").function("*rr_from_pointers*").return
{
  oldsort++;
}

probe
process("/usr/local/bin/mysqld").function("*rr_unpack_from_buffer*").return
{
  newsort++;
}

probe end
{
  printf("# of returned rows sorted by old algorithm: %d \n", oldsort);
  printf("# of returned rows sorted by new algorithm: %d \n", newsort);
}
-----
[root #] stap sort.stp
# of returned rows sorted by old algorithm: 0
# of returned rows sorted by new algorithm: 100
```

gdb

- Debugging tool
- gdb has a functionality to take thread stack dumps from a running process (similar to Solaris “truss”)
- Useful to identify where and why mysqld hangs up, slows down, etc
  - But you have to read MySQL source code
- Debugging symbol is required on the target program

gdb case study

```
mysql> SELECT query_time, start_time, sql_text
    -> FROM mysql.slow_log WHERE start_time
    -> BETWEEN '2010-02-05 23:00:00' AND '2010-02-05 01:00:00'

    -> ORDER BY query_time DESC LIMIT 10;
+------------+---------------------+----------+
| query_time | start_time          | sql_text |
+------------+---------------------+----------+
| 00:00:11   | 2010-02-05 23:09:55 | begin    |
| 00:00:09   | 2010-02-05 23:09:55 | Prepare  |
| 00:00:08   | 2010-02-05 23:09:55 | Prepare  |
| 00:00:08   | 2010-02-05 23:09:55 | Init DB  |
| 00:00:08   | 2010-02-05 23:09:55 | Init DB  |
| 00:00:07   | 2010-02-05 23:09:55 | Prepare  |
| 00:00:07   | 2010-02-05 23:09:55 | Init DB  |
| 00:00:07   | 2010-02-05 23:09:55 | Init DB  |
| 00:00:07   | 2010-02-05 23:09:55 | Init DB  |
| 00:00:06   | 2010-02-05 23:09:55 | Prepare  |
+------------+---------------------+----------+
10 rows in set (0.02 sec)
```

- **Suddenly all queries were not responding for 1-10 seconds**
- **Checking slow query log**
- **All queries are simple enough, it's strange to take 10 seconds**
- **CPU util (%us, %sy) were almost zero**
- **SHOW GLOBAL STATUS, SHOW FULL PROCESSLIST were not helpful**

Taking thread dumps with gdb

```
gdbtrace() {
…
  PID=`cat /var/lib/mysql/mysql.pid`
  STACKDUMP=/tmp/stackdump.$$
  echo 'thread apply all bt' >
$STACKDUMP
  echo 'detach' >> $STACKDUMP
  echo 'quit' >> $STACKDUMP
  gdb --batch --pid=$PID -x $STACKDUMP
}

while loop
do
  CONN=`netstat -an | grep 3306 | grep
ESTABLISHED | wc | awk '{print $1}'`
  if [ $CONN -gt 100 ]; then
    gdbtrace()
  done
  sleep 3
done
```

- Attaching running mysqld, then taking a thread dump
- Taking dumps every 3 seconds
- Attaching & Dumping with gdb is expensive so invoke only when exceptional scenario (i.e. conn > threshold) happens

- Check if the same LWPs are waiting at the same place

Stack Trace

```
.....
Thread 73 (Thread 0x46c1d950 (LWP 28494)):
#0  0x00007ffda5474384 in __lll_lock_wait () from /lib/libpthread.so.0
#1  0x00007ffda546fc5c in _L_lock_1054 () from /lib/libpthread.so.0
#2  0x00007ffda546fb30 in pthread_mutex_lock () from
/lib/libpthread.so.0
#3  0x0000000000a0f67d in my_pthread_fastmutex_lock (mp=0xf46d30) at
thr_mutex.c:487
#4  0x000000000060cbe4 in dispatch_command (command=16018736, thd=0x80,
    packet=0x65 <Address 0x65 out of bounds>, packet_length=4294967295)
    at sql_parse.cc:969
#5  0x000000000060cb56 in do_command (thd=0xf46d30) at sql_parse.cc:854
#6  0x0000000000607f0c in handle_one_connection (arg=0xf46d30) at
sql_connect.cc:1127
#7  0x00007ffda546dfc7 in start_thread () from /lib/libpthread.so.0
#8  0x00007ffda46305ad in clone () from /lib/libc.so.6
#9  0x0000000000000000 in ?? ()
```

- Many threads were waiting at pthread_mutex_lock(), called from sql_parse.cc:969

Reading sql_parse.cc:969

```
953  bool dispatch_command(enum enum_server_command command, THD *thd,
954                        char* packet, uint packet_length)
955  {
956    NET *net= &thd->net;
957    bool error= 0;
958    DBUG_ENTER("dispatch_command");
959    DBUG_PRINT("info",("packet: '%*.s'; command: %d", packet_length,
packet, command));
960
961    thd->command=command;
962    /*
963      Commands which always take a long time are logged into
964      the slow log only if opt_log_slow_admin_statements is set.
965    */
966    thd->enable_slow_log= TRUE;
967    thd->lex->sql_command= SQLCOM_END; /* to avoid confusing VIEW
detectors */
968    thd->set_time();
969    VOID(pthread_mutex_lock(&LOCK_thread_count));
```

Who locked LOCK_thread_count for seconds?

```
Thread 1 (Thread 0x7ffda58936e0 (LWP 15380)):
#0  0x00007ffda4630571 in clone () from /lib/libc.so.6
#1  0x00007ffda546d396 in do_clone () from /lib/libpthread.so.0
#2  0x00007ffda546db48 in pthread_create@@GLIBC_2.2.5 () from
/lib/libpthread.so.0
#3  0x0000000000600a66 in create_thread_to_handle_connection (thd=0x3d0f00)
    at mysqld.cc:4811
#4  0x00000000005ff65a in handle_connections_sockets (arg=0x3d0f00) at
mysqld.cc:5134
#5  0x00000000005fe6fd in main (argc=4001536, argv=0x4578c260) at
mysqld.cc:4471
#0  0x00007ffda4630571 in clone () from /lib/libc.so.6
```

- gdb stack dumps were taken every 3 seconds
- In all cases, Thread 1 (LWP 15380) was stopped at the same point
- clone() (called by pthread_create()) seemed to take a long time

Reading mysqld.cc:4811

```
4795 void create_thread_to_handle_connection(THD *thd)
4796 {
4797   if (cached_thread_count > wake_thread)
4798   {
4799     /* Get thread from cache */
4800     thread_cache.append(thd);
4801     wake_thread++;
4802     pthread_cond_signal(&COND_thread_cache);
4803   }
4804   else
4805   {
4811     if ((error=pthread_create(&thd->real_id,&connection_attrib,
4812                               handle_one_connection,
4813                               (void*) thd)))
4839   }
4840   (void) pthread_mutex_unlock(&LOCK_thread_count);
```

- pthread_create is called under critical section (LOCK_thread_count is released after that)
- If cached_thread_count > wake_thread, pthread_create is not called
- Increasing thread_cache_size will fix the problem!

Configuration Summary

- Install at least sar, mpstat, iostat (sysstat package)
  - Oprofile, gdb and SystemTap(stap) are recommended
- Allocate swap space (approx half of RAM size)
- Set vm.swappiness = 0 and use O_DIRECT
- Set /sys/block/sdX/queue/scheduler = deadline or noop
- Filesystem Tuning
  - relatime (noatime)
  - ext3: tune2fs –O dir_index -c –l –i 0
  - xfs: nobarrier
  - Make sure write cache with battery is enabled
- Others
  - Make sure to allocate separate database partitions (/var/lib/mysql, /tmp) from root partition (/)
    - When database size becomes full, it should not affect Linux kernels
  - /etc/security/limits.conf
    - soft nofile 8192
    - hard nofile 8192
  - Restart linux if kernel panic happens
    - kernel.panic_on_oops = 1
    - kernel.panic = 1

Enjoy the conference!

- The slides will be published at Slideshare very soon
- My talks on Wed/Thu
  - More Mastering the Art of Indexing
    - April 14th (Wed), 14:00-15:00, Ballroom A
  - SSD Deployment Strategies for MySQL
    - April 15th (Thu), 14:00-14:45, Ballroom E
- Contact:
  - E-mail: Yoshinori.Matsunobu@sun.com
  - Blog http://yoshinorimatsunobu.blogspot.com
  - @matsunobu on Twitter

The preceding is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.