Database Hardware Benchmarking

Greg Smith
Truviso
05/27/2009
About this presentation

- The master source for these slides is
  http://www.westnet.com/~gsmith/content/postgresql

- Slides are released under the Creative Commons Attribution
  3.0 United States License
  http://creativecommons.org/licenses/by/3.0/us
Why should you always benchmark your hardware?

- Many useful tests will only run when the server isn’t being used yet
- Software stacks are complicated
- Spending money on upgrades only helps if you upgrade the right thing usefully
- Vendors lie
Systematic Benchmarking

- Memory
- CPU
- Disk
- Database server
- Application
Memory Tests

► DOS: memtest86+ (also on many Linux CDs, like the Ubuntu installer)
► Windows: SiSoftware Sandra
► UNIX: sysbench
► Linux: hdparm (only sometimes!)
memtest86+ sample

Memtest86+ v1.00

Pentium 4 (0.13) 3000 Mhz | Test 41x
L1 Cache: 8K 24589MB/s  | Test #4 [Moving inv, 32 bit pattern, cached]
Memory: 255M 2442MB/s   | Pattern: ffbfffff
Chipset: Intel i875P (ECC : Disabled) - FSB : 250 Mhz - PAT : Enabled
Settings: RAM : 200 Mhz (DDR400) / CAS : 2.5-2-2-5 / Dual Channel (120 bits)

<table>
<thead>
<tr>
<th>WallTime</th>
<th>Cached</th>
<th>RsvdMem</th>
<th>MemMap</th>
<th>Cache</th>
<th>ECC</th>
<th>Test</th>
<th>Pass</th>
<th>Errors</th>
<th>ECC Errs</th>
</tr>
</thead>
<tbody>
<tr>
<td>0:01:02</td>
<td>255M</td>
<td>864K</td>
<td>e820-Std</td>
<td>on</td>
<td>off</td>
<td>Std</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

(ESC)Reboot (c)onfiguration (SP)scroll_lock (CR)scroll_unlock
sysbench memory write and CPU tests

sysbench --test=memory --memory-oper=write
   --memory-block-size=1024MB --memory-total-size=1024MB
run

sysbench --test=cpu run
Memory test comparisons

<table>
<thead>
<tr>
<th>CPU</th>
<th>Frequency</th>
<th>RAM Speed</th>
<th>memtest86+</th>
<th>sysbench</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q6600</td>
<td>2.4GHz</td>
<td>DDR2/667</td>
<td>2678</td>
<td>-</td>
</tr>
<tr>
<td>Q6600</td>
<td>2.4GHz</td>
<td>DDR2/800</td>
<td>3352</td>
<td>2009</td>
</tr>
<tr>
<td>T9400</td>
<td>2.53GHz</td>
<td>DDR2/1066</td>
<td>3743</td>
<td>2226</td>
</tr>
<tr>
<td>Xeon X5450</td>
<td>3.00GHz</td>
<td>DDR2/1333</td>
<td>3575</td>
<td>2487</td>
</tr>
</tbody>
</table>

- In Q6600 examples, DDR2/800 should be 20% faster than DDR2/667
- The 25% gain is because of improved clock multipliers
Sources for slow memory results

- Single channel RAM/slot mistakes
- Incorrect SPD/timing/voltage
- Bad RAM/CPU multiplier combination
- Poor quality RAM
PostgreSQL and the CPU

- PostgreSQL uses only a single CPU per query
- Queries executing against cached data will bottleneck on CPU
- COPY is CPU intensive
CPU Tests

- Windows: SiSoftware Sandra
- UNIX: sysbench CPU test
- Custom test with timing and generate series
- pgbench select-only on small database
- Vary pgbench client count to test single or multiple CPUs
### sysbench CPU comparisons

<table>
<thead>
<tr>
<th>Processor</th>
<th>Frequency</th>
<th>sysbench CPU seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Q6600</td>
<td>2.40GHz</td>
<td>19.6</td>
</tr>
<tr>
<td>Intel T9400</td>
<td>2.53GHz</td>
<td>12.0</td>
</tr>
<tr>
<td>Intel Xeon X5450</td>
<td>3.00GHz</td>
<td>8.4</td>
</tr>
</tbody>
</table>
pgbench Read-Only Scaling - 8GB server

![Graph showing pgbench SELECT transactions/sec vs. Database Size (MB)](image)

- **TPS**
- **Database Size (MB)**

**Scaling factor**

**Database Hardware Benchmarking**
Sources for slow CPU results

- Slow memory
- Power management throttling
- Linux: /proc/cpuinfo shows 1000MHz suggests you need to adjust the CPUFreq Governor to “performance”
Disk Tests

- Sequential write: INSERT, COPY FROM (when not CPU limited)
- Sequential read: SELECT * FROM and similar table sequential scans
- Seeks: SELECT using index, UPDATE
- Commit fsync rate: INSERT, UPDATE
Compute 2X the size of your RAM in 8KB blocks
blocks = 250,000 * gigabytes of RAM

time sh -c "dd if=/dev/zero of=bigfile bs=8k count=X && sync"

time dd if=bigfile of=/dev/null bs=8k

Watch vmstat and/or iostat during disk tests
vmstat’s bi and bo will match current read/write rate
Note the CPU percentage required to reach the peak rate
./bonnie++

bon_csv2html

- Ignore the per-character and create results, look at the block output/input ones
- Random Seeks:
- The test runs SeekProcCount processes (default 3) in parallel, doing a total of 8000 random seek reads to locations in the file. In 10% of cases, the block read is changed and written back.
bonnie++ ZCAV

./zcav -f/dev/sda > t500

- Must get a recent version of bonnie++ for ZCAV to scale properly for TB drives (1.03e works)
- Download somewhat broken gnuplot script sample and typical results from:

unset autoscale x
set autoscale xmax
unset autoscale y
set autoscale ymax
set xlabel "Position GB"
set ylabel "MB/s"
set key right bottom
set terminal png
set output "zcav.png"
plot "raid0" title "7200RPM RAID 0 3 Spindles",
  "t500" title "7200RPM Laptop"
bonnie++ ZCAV: Laptop 7200RPM Disk
Comparison with 3-Disk RAID0 of 7200RPM SATA Disks
Read seeks/second - sysbench

THREADS=1
GB=10
MODE=rndrd
OPTIONS="--test=fileio --num-threads=$THREADS
   --file-block-size=8K --file-test-mode=$MODE
   --file-num=$GB --file-total-size=${GB}G
   --file-fsync-freq=0 --file-fsync-end=no"

sysbench prepare $OPTIONS
sysbench run --max-time=60 $OPTIONS
sysbench cleanup $OPTIONS
Sample sysbench random read results

Read 78.125Mb Written 0b
Total transferred 78.125Mb (1.0059Mb/sec)
128.75 Requests/sec executed

▶ That’s 128.75 seeks/second over 10GB, resulting in a net throughput of 128.75 * 8KB/s = 1.01MB/s
▶ Consider both the size of the disk used and the number of clients doing seeks
More customizable seek tests

- bonnie++ experimental (currently at 1.95)
- iozone
- fio
- Windows: HD Tune does everything but commit rate
Sources for slow disk results

- Poor mapping to underlying hardware
- Buggy driver
- Insufficient bandwidth to storage
- Bottlenecking at CPU/memory limits
- Bad performing filesystem or filesystem misaligned with stripe sizes
- Writes faster than reads? Probably low read-ahead settings somewhere.
- Vibration: don’t shout at your JBODs! They don’t like it!

http://www.youtube.com/watch?v=tDacjrSCeq4
sysbench --test=fileio --file-fsync-freq=1 --file-num=1 --file-total-size=16384 --file-test-mode=rndwr run | grep "Requests/sec"

➤ pgbench insert-only test
➤ PostgreSQL contrib/test_fsync might work, but isn’t really reliable
Sample laptop disk specification

- ST9160823AS Momentus 7200.2
- 7200 RPM
- 8MB Cache
- Average seek: 11ms
- Average rotational latency: 4.17ms
- Transfer rate: 59MB/s
Computed parameters

- Rotational latency = \( \frac{1}{\text{RPM} \times 60 \times 2} \)
- IOPS = \( \frac{1}{\text{latency} + \text{seek}} \)
- IOPS = \( \frac{1}{\left( \frac{1}{\left( \frac{1}{(\text{RPM}/60)/2} \right)} + S \right)} \)
- IOPS = \( \frac{1}{4.17\text{ms} + 11\text{ms}} \) = 65.9 IOPS
IOPS Calculators and Info

http://www.wmarow.com/strcalc/

http://www.dbasupport.com/oracle/ora10g/disk_IO_02.shtml

http://storageadvisors.adaptec.com/2007/03/20/sata-iops-measurement/
## Sample disk results

<table>
<thead>
<tr>
<th>Disks</th>
<th>Read</th>
<th>Write</th>
<th>bonnie++ seeks</th>
<th>RO seeks</th>
<th>Commit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>41</td>
<td>35</td>
<td>178 @ 6GB</td>
<td>129/s @ 10GB</td>
<td>2653/s or 58/s</td>
</tr>
<tr>
<td>3</td>
<td>125</td>
<td>119</td>
<td>371 @ 16GB</td>
<td>60/s @ 100GB</td>
<td>10855/s</td>
</tr>
</tbody>
</table>

- Two commit rates for 1 disk setup (the laptop drive) are with/without an unsafe write cache
- 3 disk RAID0 includes a 256MB battery-backed write cache
Custom PostgreSQL tests

- Quick CPU test (about 1140ms on my laptop):

```sql
\timing
select sum(generate_series) from generate_series(1,1000000);
```

- Quick insert/plan test:

```sql
\timing
CREATE TABLE test (id INTEGER PRIMARY KEY);
INSERT INTO test VALUES (generate_series(1,100000));
EXPLAIN ANALYZE SELECT COUNT(*) FROM test;
```
Benchmarks functions

http://justatheory.com/computers/databases/postgresql/benchmarking_upc_validation.html
http://justatheory.com/computers/databases/postgresql/benchmarking_functions.html

- `pg_stat_user_functions` handles this specific job in 8.4
- The general technique is applicable for all sorts of custom benchmarks
\set naccounts 100000 * :scale
\setrandom aid 1 :naccounts
SELECT abalance FROM accounts WHERE aid = :aid;
\set nbranches :scale
\set ntellers 10 * :scale
\set naccounts 100000 * :scale
\setrandom aid 1 :naccounts
\setrandom aid 1 :naccounts
\setrandom bid 1 :nbranches
\setrandom tid 1 :ntellers
\setrandom delta -5000 5000
BEGIN
INSERT INTO history (tid, bid, aid, delta, mtime)
  VALUES (:tid, :bid, :aid, :delta, CURRENT_TIMESTAMP);
END;
BEGIN;

UPDATE accounts SET abalance = abalance + :delta
    WHERE aid = :aid;

SELECT abalance FROM accounts WHERE aid = :aid;

UPDATE tellers SET tbalance = tbalance + :delta
    WHERE tid = :tid;

UPDATE branches SET bbalance = bbalance + :delta
    WHERE bid = :bid;

INSERT INTO history (tid, bid, aid, delta, mtime)
    VALUES (:tid, :bid, :aid, :delta, CURRENT_TIMESTAMP);

END;
pgbench TPC-B Size Scaling

The graph shows the relationship between pgbench transactions per second (TPS) and database size in MB as a function of the scaling factor. The red line represents the TPS, while the green line shows the database size. As the scaling factor increases, the TPS decreases, indicating a trade-off between performance and database size.
pgbench TPC-B Client Scaling

![Graph showing TPC transactions per second (TPS) versus number of clients. The graph indicates a steep increase in TPS as the number of clients increases, reaching a peak before declining with further increases in the number of clients.]
pgbench TPC-B Size and Client Scaling
(3D glasses not included)
pgbench select-only warm cache: scale=100, clients=4, 1M transactions
Cold cache: scale=100, clients=4, 1M transactions
create table data(filler text);

insert into data (filler) values (repeat('X',:scale));

SCALES="10 100 1000 10000"

SCRIPT="insert-size.sql"

TOTTRANS=100000

SETTIMES=1

SETCLIENTS="1 2 4 8 16"

SKIPINIT=1
### Insert size quick test

<table>
<thead>
<tr>
<th>Scale (Clients)</th>
<th>10</th>
<th>100</th>
<th>1000</th>
<th>10000</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1642</td>
<td>1611</td>
<td>1621</td>
<td>1625</td>
</tr>
<tr>
<td>2</td>
<td>2139</td>
<td>2142</td>
<td>2111</td>
<td>2232</td>
</tr>
<tr>
<td>4</td>
<td>3196</td>
<td>3306</td>
<td>3232</td>
<td>3296</td>
</tr>
<tr>
<td>8</td>
<td>4728</td>
<td>5243</td>
<td>5445</td>
<td>5038</td>
</tr>
<tr>
<td>16</td>
<td>9372</td>
<td>8309</td>
<td>8140</td>
<td>7238</td>
</tr>
</tbody>
</table>
What should you do?

- Trust no one
- Don’t start on application benchmarks until you’ve proven basic performance
- Vendors alternate among lying, misunderstanding what you want, and trying to make you feel dumb
- Use simple, standard tools whenever possible to minimize vendor disputes
- Be prepared to translate to your vendor’s language and subvert their agenda
- Never spend real money on hardware unless you can return it if it sucks