PostgreSQL Performance From the Ground Up

Lacey Powers
Moonshadow Mobile, Inc.

Overview

- Hardware
- Operating System (Linux)
- PostgreSQL
- Queries

Choosing Hardware

- These are nice, general guidelines that are useful for PostgreSQL, MySQL, SQLServer, Oracle, ect.
- Be Scientific!
- Take measurements, before and after, for your workload, so you know what changed, and what effect it has.
- Keep a journal, so you have a record over the life of the server.

Hardware Basics: Processor

- **Rule of Thumb**
- At least 2 cores
- More cores, or faster cores?

- **Why**
- PostgreSQL can only use a single core per query. There are no parallel queries (yet).
- Depends on workload.
- Lots of big queries. Faster cores
- Lots of little ones.

  More cores

Hardware Basics: Memory

- **Rule of Thumb**
- At least 4GB Memory
- Database >> RAM
- Database << RAM

Why

- Memory is cheap
- PostgreSQL efficiently caches
- Enough RAM to fit the DB – Get more cores.
- DB Won't fit in RAM, spend the money on disks.

Hardware Basics: Memory, Cont.

- STREAM/STREAM-SCALING
- Memtest

- Measures memory and processor speed.
- Allows burn-in of RAM.
- Bathtub curve of failure.

Hardware Basics: Memory, Cont

Function	Rate (MB/s)	Avg time	Min time	Max time
Copy:	7543.1208	0.0670	0.0670	0.0670
Scale:	7524.1314	0.0672	0.0671	0.0672
Add:	8163.0566	0.0929	0.0928	0.0929
Triad:	8136.9356	0.0932	0.0931	0.0932

Number of Threads requested = 2

Function	Rate (MB/s)	Avg time	Min time	Max time
Triad:	13619.7966	0.0557	0.0556	0.0558

Number of Threads requested = 3

Function	Rate (MB/s)	Avg time	Min time	Max time
Triad:	14481.8310	0.0524	0.0523	0.0524

Number of Threads requested = 4

Function	Rate (MB/s)	Avg time	Min time	Max time
Triad:	17828.8907	0.0426	0.0425	0.0426

Number of Threads requested = 8

Function	Rate (MB/s)	Avg time	Min time	Max time
Triad:	18729.0861	0.0405	0.0405	0.0405

Hardware Basics: Disks

- **Rule of Thumb**
- SAS 15k
- Enterprise Class Disks
- SATA ok, but 2:1, generally.

- **Why**
- Disks are generally the bottleneck.
- Stable firmware, better command queueing, quick failure.
- Slower, and less optimized throughput but higher storage
- Slow disks, slow array

Hardware Basics: Disks, cont.

- **Rule of Thumb**
- SSDs – not quite there yet.
-

- **Why**
- No reliable way to flush the write cache on the disk.
- Most RAID cards don't work nicely with them.
- Small capacity
- But useful for other things....

Hardware Basics: Disks, Cont.

- **Rule of Thumb**
- Battery Backed Raid Card with cache.
- LSI, Areca, HP, Dell are good cards.
- Promise, Highpoint (no cache or battery, very bad).

- **Why**
- Good cards queue and order writes when they are good for the disks, not the OS.
- Battery backup protects data integrity

Hardware Basics: Disks, Cont

Rotation speed	Rotation time (ms)	Max commits/second
5400	11.1	90
7200	8.3	120
10000	6.0	166
15000	4.0	250

Hardware Basics: Disks, Cont.

- **Rule of Thumb**
- Raid 1 – OS
- Raid 1 – WAL
- Raid 10 – Data
- Raid 0 – Very bad, no redundancy
- Raid 5 – Bad for writes

- **Why**
- OS – Random IO, low usage
- WAL – Sequential IO, constant usage.
- Data – Random IO, high usage
- Allows profiling of each area of the machine

Hardware Basics: Disks, Cont.

- dd
- Bonnie++
- time dd bs=8k if=/dev/zero of=test.bin count=131072

- Simple tests
- Easy to replicate results
- Leverage to get the vendor to fix things, esp. SAN
- Check the vendor's work.

Operating System

- Linux (*BSD, Solaris, Windows ok too).
- x86_64
- Ext3 or XFS (JFS, Reiser3/4, ZFS, ext4, btrfs not well supported/tested)
- Ext3 16TB Filesystem Limit. XFS 1 million TB
- Ext3 boot, ext3 /, xfs /srv/postgresql/data, ext3 /srv/postgresql/xlogs

Operating System, Cont.

- Ext3 data=ordered or data=journal
- Difference between journal and writeback is tiny with a BBWC, and won't give garbage or old/new mix in files during a crash.
- Xfs – more consistent performance, tweak sunit, swidth, logbufs.
- Aggressive syncing/flushing to ensure that the data is there. With a BBWC, enable nobarrier.
- DO NOT TURN OFF full_page_writes

Operating System, Cont.

- Benchmarking will show if the readahead needs adjusted.
- blockdev --setra 4096 /dev/sda. Put this in rc.local to make sure it's set on reboot.
- Use noatime to disable file access times
- vm.swappiness = 0 – Shrink filesystem cache instead of swapping
- vm.overcommit_memory = 2 – Disable OOMKiller

Operating System, Cont.

- vm.dirty_ratio = 10 and vm.dirty_background_ratio = 5 (if not set already) – Smooths out syncs of memory to disk
- Schedulers: noop good for SAN/RAID with a lot of cache.

  Deadline – not always the best, be sure to test

  cfq – if in doubt, use this. Most tested, and widely used.

Operating System, Cont

- Move the xlogs to their own disks.

- cd /var/lib/postgresql/8.4/main
- mv pg_xlog /srv/postgresql/xlogs/pg_xlog
- ln -s /srv/postgresql/xlogs/pg_xlog pg_xlog

Operating System, Cont.

- Use tablespaces on the snazzy RAID 10, for ease.
- mkdir /srv/postgresql/data/my_db_tables
- CREATE TABLESPACE my_db_tables LOCATION '/srv/postgresql/data/my_db_tables';
- CREATE DATABASE my_db OWNER my_owner TABLESPACE my_db_tables;
- Can do this by table and index, to spread IO around if you get more disks
- Temp table tablespaces on SSDs!

Operating System, Cont.

- kernel.shmall = PAGE_SIZE / 2
- kernel.shmmax = kernel.shmall * PAGE_SIZE
- Good estimates of changing shared memory sizes.

PostgreSQL

- shared_buffers make the most difference.

  They consist of:

  1. Connections/Autovacuum Workers.
  2. Prepared transactions
  3. Shared Disk Buffers*
  4. Wal Buffers
  5. Fixed Space.

PostgreSQL, cont

- Start at 25% of system physical memory
- If you need a better measure, use pg_buffercache in contrib
- You can resize upwards or downwards depending on the results of your analysis
- Usagecount (0,1) you could decrease the size of the shared_buffers.
- Usagecount (4,5) you could increase shared_buffers.

PostgreSQL, Cont.

- effective_cache_size
- 50-75% of memory
- Add shared_buffers to this, if the DB is running. Otherwise use the output of free.
- Estimate of space available to load pages from disk into memory for processing.
- If you overdo it, you'll push other things out of cache.

PostgreSQL, Cont.

- listen_addresses
- Listen on as few interfaces as possible. Connections are expensive.
- Filter at the firewall level instead, so there is less of a chance of a DDOS on your DB.
- Make lots of use of the pg_hba.conf

PostgreSQL, cont.

- max_connections = 100
- Default. Connections take memory and processing to set up and tear down.
- Use a connection pooler to handle more connections and to keep them persistent.
- Pgbouncer and pgpool are good choices

PostgreSQL, Cont.

- maintenance_work_mem = 50 MB/1GB of memory
- default_statistics_target = 100 (10 on pre-8.4 systems, generally results in better query plans at the cost of additional statistics and planning time)
- Can be set per-column. Good candidates are LIKE comparison columns.

PostgreSQL, Cont.

- effective_io_concurrency – Only in 8.4+.
- Number of concurrent IO operations that can be expected to be executed at once.
- Good starting point is the number of spindles in a RAID 1. For RAID 10, start with 4 and if it is very busy, go to 2.
- Experimentation is good here, since it's a newer variable.

PostgreSQL, Cont

- checkpoint_segments = 10
- Controls the number of WAL files. Can increase recovery time.
- Don't increase over 32 unless you have a high-velocity database.
- checkpoint_completion_target = 0.9
- Smooth out checkpoint IO, steady and flat, no spikes.

PostgreSQL, Cont.

- checkpoint_timeout = 5min
- Generally ok if you change other values.
- Ok to increase if you find a lot of checkpoints in your log. Use log_checkpoints parameter in the postgresql.conf
- wal_buffers = 16MB
- wal_sync_method – Generally don't have to change this. PostgreSQL and the OS determine the best one at initdb time.

PostgreSQL, Cont.

- work_mem = 1MB
- Used for sorting and hashing in queries.
- You should generally change this on a per-connection basis.
- You can tune it higher, if you need by ((os_cache_size / max_connections)/2)

PostgreSQL, Cont.

- Don't change these:
- a. fsync - Affects durability
- b. full_page_writes – Affects durability
- c. commit_delay and commit_siblings – Taken care of by synchronous_commit
- d. max_prepared_transactions – Only need to change if you use them
- e. Any of the query enable parameters: e.g. enable_seqscan – Messes up the planner, most of the time.

PostgreSQL, Cont.

- Autovacuum
- DO NOT TURN THIS OFF. =)
- Used to clean up dead tuples, update statistics, and xids.
- Will run in 8.3+ for xid wraparound, even if you turn it off.

PostgreSQL, Cont.

- Autovacuum Common Problems
- Constantly Running
- Could need to increase your maintenance_work_mem
- Could need to increase the autovacuum_naptime, especially if you have a lot of databases.

PostgreSQL, cont.

- Autovacuum running out of memory.
- Reduce allocation of maintenance_work_mem or max_autovacuum_workers

- Autovacuum is too disruptive
- Increase the autovacuum_vacuum_cost_delay ~100ms is generally ok.

PostgreSQL, Cont.

- Autovacuum is not keeping up.
- Decrease autovacuum_vacuum_cost_delay. Between 1ms and 5ms.
- Increase autovacuum_cost_limit after that.
- If bloat is too high (~20% or more dead tuples), CLUSTER and REINDEX to reduce space and increase performance. Will cause downtime, though. =(

PostgreSQL, Cont.

- Logging. Allows you to gather reasonable data on your server.

  log_destination = 'stderr'
- logging_collector = on
- log_directory = '/var/log/postgresql/8.4'
- log_filename = 'postgresql-%a.log'
- log_truncate_on_rotation = on

PostgreSQL, Cont.

- log_rotation_age = 1d
- log_rotation_size = 0
- log_min_duration_statement = 1000
- log_checkpoints = on
- log_connections = on
- log_disconnections = on
- log_line_prefix = '%t [%p]: [%l-1] user=%u,db=%d,remote=%r '
- log_statement = 'ddl'

Queries

- The postgresql.log is your friend.
- Logging all queries for a couple of days (if you can) will give you a good enough variation.
- Analysis Tools:

  pg_fouine

  maatkit
- auto_explain configuration option
- EXPLAIN ANALYZE

Queries, Cont.

```
lacey@[local]:5432:lacey:=# EXPLAIN ANALYZE SELECT * FROM test_table
ORDER BY description;
                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------------
-----
 Sort  (cost=989.07..1012.99 rows=9571 width=18) (actual time=94.330..124.852
rows=9555 loops=1)
   Sort Key: description
   Sort Method:  external merge  Disk: 272kB
   ->  Seq Scan on test_table  (cost=0.00..156.71 rows=9571 width=18) (actual
time=0.011..12.198 rows=9555 loops=1)
 Total runtime: 135.944 ms
(5 rows)

lacey@[local]:5432:lacey:=#
```

Queries, Cont.

```
lacey@[local]:5432:lacey:=# SHOW work_mem;
 work_mem
----------
 64kB
(1 row)

lacey@[local]:5432:lacey:=# SET work_mem = "8MB";
SET
lacey@[local]:5432:lacey:=# EXPLAIN ANALYZE SELECT * FROM test_table
ORDER BY description;
                                                  QUERY PLAN
--------------------------------------------------------------------------------------------------------------
--
 Sort  (cost=789.57..813.49 rows=9571 width=18) (actual time=90.804..102.717
rows=9555 loops=1)
   Sort Key: description
   Sort Method:  quicksort  Memory: 1131kB
   ->  Seq Scan on test_table  (cost=0.00..156.71 rows=9571 width=18) (actual
time=0.012..16.551 rows=9555 loops=1)
 Total runtime: 113.794 ms
(5 rows)

lacey@[local]:5432:lacey:=#
```

Queries, Cont.

- Biggest query issues are generally lack of sorting memory, a lack of statistics, or a **very** badly written query.
- Rewriting badly written queries can be a presentation all on its own.
- http://explain.depesz.com/
- Invaluable for figuring out what in a query plan is causing the most trouble.

And that's it! =)

- Questions?
-
- For more info:

  pgsql-performance@postgresql.org

  Look up Greg Smith's notes

- Thanks for listening! =)