PostgreSQL Performance
The basics

Joshua D. Drake

jd@commandprompt.com
Command Prompt, Inc.
United States PostgreSQL
Software in the Public Interest

The dumb simple

RAID 1 or 10 (RAID 5 is for chumps)

Lots of memory. It's cheap, get 4G (minimum)

At least two cores (PostgreSQL is process based)

If SATA consider twice as many disks

Always get a BBU

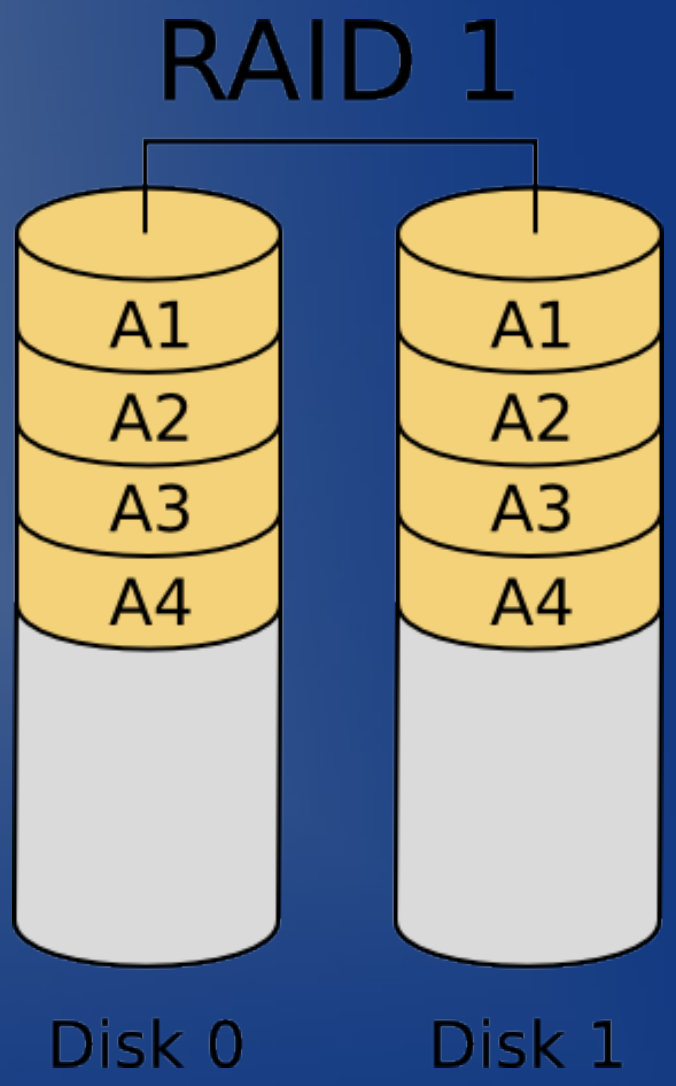
Split Transaction Logs and data.

Use autovacuum (or pay emergency rates)

RAID 1

Redundancy through use of mirror

Increased performance (sometimes) through shared or partitioned reads

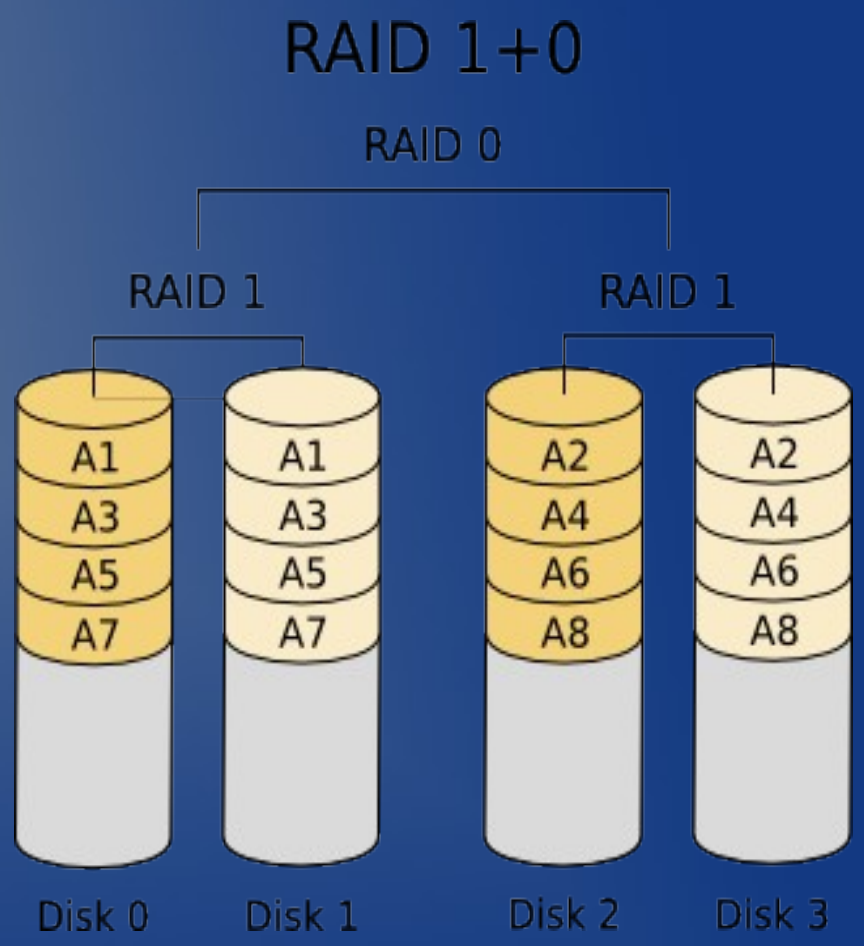


RAID 1 + 0

Minimum 4 Spindles

Increased performance through use of stripe

Increased reliability through use of mirror



Lots of memory

PostgreSQL is efficient and it is possible to run effectively in as little as 256Mb of memory. Memory is cheap, most data sets are less than 4Gb. If you have at least 4Gb your active data set can remain in file and or shared_buffer cache.

SATA

SATA is great. It is cheap. It is reliable. Just make sure you have a BBU (because of large caches) and you use twice as many spindles.

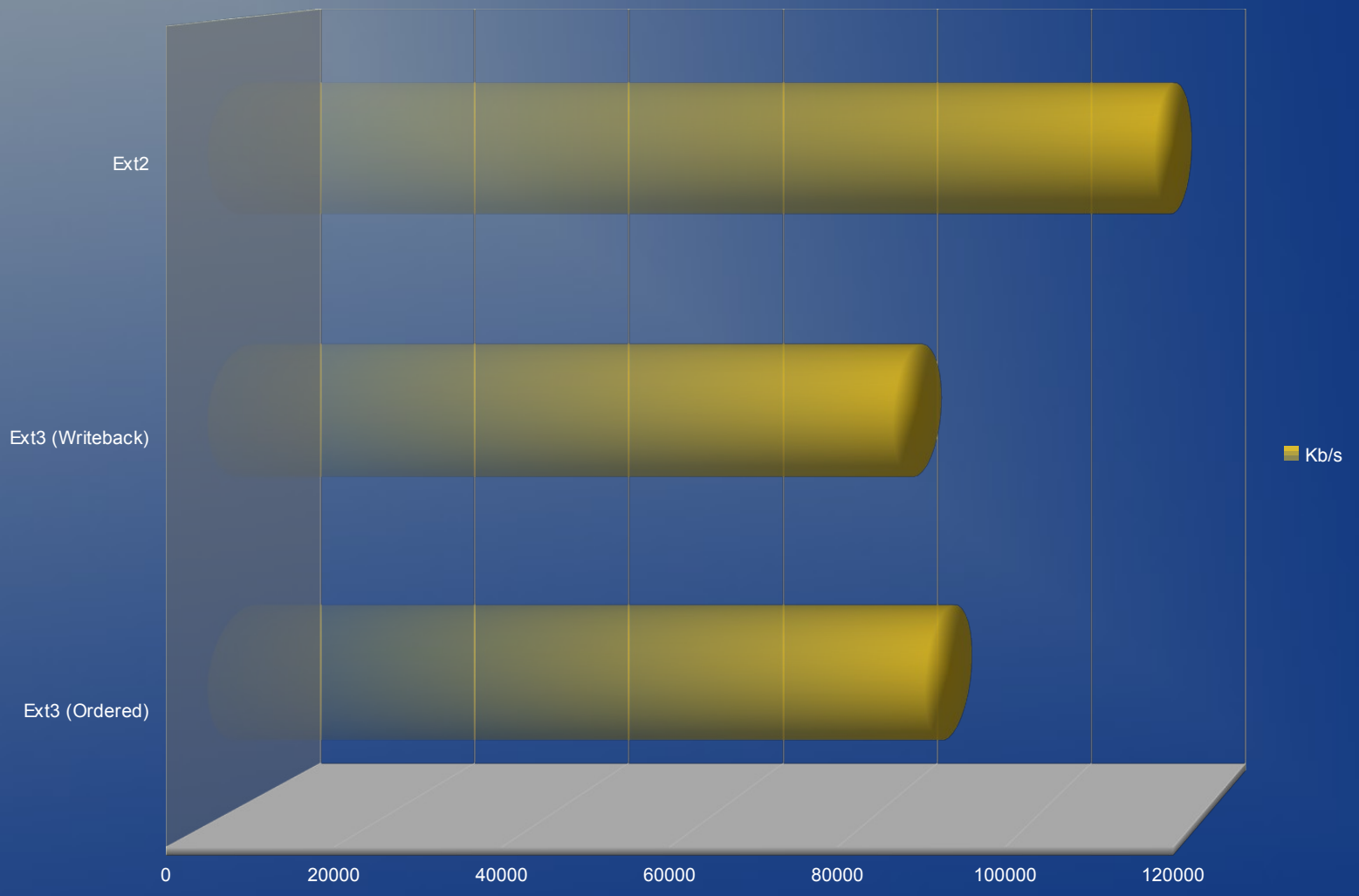
Two or More

PostgreSQL is processed based. The more cores you have, the better concurrency you will get.

BBU

Battery Backup Unit
Used on good RAID cards in case of power outage or sudden crash. Allows for storage of pending writes until the machine comes back on line. A requirement if you are running any kind of CACHE on the RAID or Drives.

Splitting Transaction Logs



Autovacuum defaults

```
#autovacuum = on
#autovacuum_max_workers = 3
#autovacuum_naptime = 1min
#autovacuum_vacuum_scale_factor = 0.2
#autovacuum_analyze_scale_factor = 0.1
```

Memory

shared_buffers

work_mem

maintenance_work_mem

effective_cache_size

What are shared_buffers

The working cache of all hot tuples (and Index entries) within PostgreSQL.

shared_buffers

Pre-allocated cache (buffers).

On Linux sysctl.conf – kernel.shmmax

Use 20% of available memory (up to 40%)

Watch out for IO Storms

What is work_mem

The working memory available for work operations (sorts) before PostgreSQL will swap.

work_mem

- Don't set globally (postgresql.conf)
- Use per transaction
- Can be bad, per query, per connection, per sort
- Use EXPLAIN ANALYZE to see if you are overflowing

Django – SET

```
def my_custom_sql(self):
   from django.db import connection, transaction
   cursor = connection.cursor()
   # Data retrieval operation - no commit required
   cursor.execute("""
      SET work_mem TO '%s'""", [self.baz])
```

Example EXPLAIN ANALYZE

```
                                QUERY PLAN
-----------------------------------------------------------------------------
 Sort  (cost=0.02..0.03 rows=1 width=0) (actual time=2270.744..2588.341
rows=1000000 loops=1)
   Sort Key: (generate_series(1, 1000000))
    Sort Method:  external merge  Disk: 13696kB
   ->  Result  (cost=0.00..0.01 rows=1 width=0) (actual
time=0.006..144.720 rows=1000000 loops=1)
 Total runtime: 3009.218 ms
(5 rows)
```

What is maintenance_work_mem

The amount of memory (RAM) allowed for maintenance tasks before PostgreSQL swaps. Typical tasks are ANALYZE, VACUUM, CREATE INDEX, REINDEX

maintenance_work_mem

Set to a reasonable amount for autovacuum
Use SET for per session changes such as CREATE INDEX

```
SET maintenance_work_mem to '1GB';
CREATE INDEX foo ON bar(baz);
RESET maintenance_work_mem;
```

What is effective_cache_size

A pointer for the PostgreSQL planner to hint at how much of the database will be cached. This is not an allocation setting.

effective_cache_size

Take into account shared_buffers

```
             total       used       free     shared    buffers     cached
Mem:       6126208    3168356    2957852          0     480884    1258304
```

% of cached + shared_buffers = effective_cache_size

% depends on workload. Generally between 40% and 70%

Let's talk IO

log_checkpoints

checkpoint_timeout

checkpoint_completion_target

checkpoint_segments

wal_sync_method

synchronous_commit

log_checkpoints

By default this is off. Turn on to correlate between checkpoints and spikes in %IOWait from sar.

checkpoint_timeout

The amount of time PostgreSQL will wait before it forces a checkpoint. Properly configured it reduces IO utilization. Set to 15 or 20 (minutes). It is affected by:

checkpoint_segments

checkpoint_completion_target

checkpoint_completion_target

This paramater is used to reduce spikes in IO by completing a checkpoint over a period of time.

Do not change this paramater, increase checkpoint_timeout instead.

checkpoint_segments

The number of transaction logs that will be utilized before a checkpoint is forced. Each segment is 16 Mb. The default is 3. Use checkpoint_warning to see if you need more.

Change to at least 10.

Use checkpoint_warning and logging to get accurate setting.

wal_sync_method

The type of fsync that will be called to flush file modifications to disk. Leave commented to have PostgreSQL figure it out. On Linux it should look like:

```
postgres=# show wal_sync_method ;
 wal_sync_method
-----------------
 fdatasync
```

synchronous_commit

Specifies whether transaction commit will wait for WAL records to be written to disk before the command returns a "success" indication to the client.

Depends on application. Turn off for faster commits. Low risk of lost commits (but not integrity).

Let's talk brains

default_statistics_target

seq_page_cost

random_page_cost (to fix)

cpu_operator_cost

cpu_tuple_cost

default_statistics_target

An arbitrary value used to determine the volume of statistics collected on a relation. The larger the value the longer analyze takes but generally the better the plan. Can be set per column.

default_statistics_target

```
set default_statistics_target to 100;
pggraph_2_2=# analyze verbose pggraph_indexrollup;
INFO:  analyzing "aweber_shoggoth.pggraph_indexrollup"
INFO:  "pggraph_indexrollup": scanned 30000 of 1448084
pages, containing 1355449 live rows and 0 dead rows; 30000 rows in
sample, 65426800 estimated total rows
ANALYZE
```

default_statistics_target

```
set default_statistics_target to 300;
pggraph_2_2=# analyze verbose pggraph_indexrollup;
INFO:  analyzing "aweber_shoggoth.pggraph_indexrollup"
INFO:  "pggraph_indexrollup": scanned 90000 of 1448084
pages, containing 4066431 live rows and 137 dead rows; 90000 rows in
sample, 65428152 estimated total rows
ANALYZE
pggraph_2_2=#
```

default_statistics_target

How do I know to increase it?

```
Unique  (cost=264.65..282.65 rows=100 width=2) (actual time=8.665..12.460
rows=100 loops=1)
   ->  Sort  (cost=264.65..273.65 rows=3600 width=2) (actual
time=8.664..10.423 rows=3600 loops=1)
         Sort Key: one
         Sort Method:  quicksort  Memory: 265kB
         ->  Seq Scan on bar
(cost=0.00..52.00 rows=52 width=2) (actual time=0.007..1.894 rows=3600
loops=1)
 Total runtime: 12.553 ms
```

Increasing per column

```
ALTER TABLE foo
   ALTER COLUMN BAR
      SET STATISTICS 120
```

seq_page_cost

Tells the planner how expensive a sequential scan is. It directly relates to random_page_cost.

random_page_cost

Tells the planner the expense of fetching a random page. If using RAID 10, the value should be inverted with seq_page_cost (1.0 vs 4.0)

cpu_operator_cost

Sets the planner's estimate of the cost of processing each operator or function executed during a query. The default is 0.0025.

In real world tests, a setting of 0.5 generally provides a better plan. Test using SET in a session.

```
SET cpu_operator_cost TO 0.5;
EXPLAIN ANALYZE SELECT ...
```

cpu_tuple_cost

Sets the planner's estimate of the cost of processing each row during a query. The default is 0.01.

In real world tests, a setting of 0.5 generally provides a better plan. Test using SET in a session.

```
SET cpu_tuple_cost TO 0.5;
EXPLAIN ANALYZE SELECT ...
```

Design

Connection Pooling

Prepared Statements

Functions

Batch Commits

Connection Pooling

Reduces CPU utilization

Keeps relations hot (in cache)

pgbouncer:

https://developer.skype.com/SkypeGarage/DbProjects/PgBouncer

Prepared Statements

Reduces planning time

Good for recurring and similar transactions

```
Start: 01:43:08 PM
 Insert 1000000 rows 1000 at a time
End: 01:43:18 PM – Time: 10 seconds

Start: 01:45:16 PM
 Insert 1000000 prepared rows 1000 at a time
End: 01:45:23 PM – Time: 7 seconds
```

Functions

- Reduces processing overhead
  - Multiple round trips
  - Manipulating data inside instead of outside
  - Make sure to test and increase cost as required

Functions - execution_cost

A positive number giving the estimated execution cost for the function, in units of cpu_operator_cost. If the function returns a set, this is the cost per returned row. If the cost is not specified, 1 unit is assumed for C-language and internal functions, and 100 units for functions in all other languages. Larger values cause the planner to try to avoid evaluating the function more often than necessary.
result_rows

Functions - SET

The SET clause causes the specified configuration parameter to be set to the specified value when the function is entered, and then restored to its prior value when the function exits. SET FROM CURRENT saves the session's current value of the parameter as the value to be applied when the function is entered.

Functions - ROWS

A positive number giving the estimated number of rows that the planner should expect the function to return. This is only allowed when the function is declared to return a set. The default assumption is 1000 rows.

CREATE FUNCTION

```
CREATE OR REPLACE FUNCTION RETURN_LOTS(INT) RETURNS SETOF INT AS
$$
   SELECT generate_series(1,$1);
$$
    COST 0.5 ROWS 10000000 SET work_mem TO '5MB'

LANGUAGE 'SQL';
```

Batch Commits

Reduces commit costs

Increases commit efficiency

```
Start: 01:45:56 PM
 Insert 1000000 rows 1000 at a time
End: 01:46:05 PM – Total: 9 seconds

Start: 01:46:05 PM
 Insert 1000000 rows one at a time
End: 01:48:07 PM – Total: 00:2:02
```

Questions?

Technical?

Community?