Scaling PostgreSQL
A Developer's Guide

Tech Talk @ EMUMBA

Sep 27, 2023

STORMATICS

Abstract

- PostgreSQL adoption is exploding and the move to the cloud is fueling it
- The difference between kicking things off and scaling in production
- The four areas of focus for scaling PostgreSQL
  - Query & SQL Optimization
  - Performance Features
  - Architectural Improvements
  - Parameter Tuning

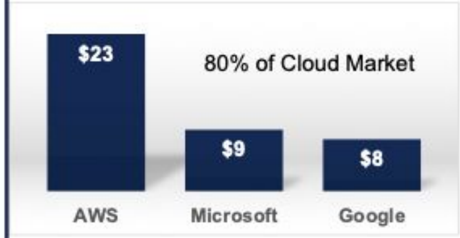
STORMATICS

Popularity

stackoverflow

2022 DBMS Market Snapshot

Top 3 Cloud Revenue ($B)



DBMS Subsegments ($B)

Top 10 (Top 5 = 81% of Market)			Fastest Growing	
AWS	**New #1!**	$23,023	Snowflake	83.2%
Microsoft		$21,970	Cockroach Labs	81.5%
Oracle		$16,875	Databricks	76.3%
Google		$7,616	EnterpriseDB	58.3%
IBM		$4,587	TigerGraph	49.3%
SAP		$3,611	MongoDB	48.7%
Alibaba		$1,939	Google	46.2%
Huawei		$1,251	Tencent	41.3%
Snowflake	**New!**	$1,223	Redis	40.7%
MongoDB	**New!**	$1,205	Singlestore	40.6%

So - what do you do when you need to scale your database in the cloud?

Scale by Credit Card!

Well, not really …

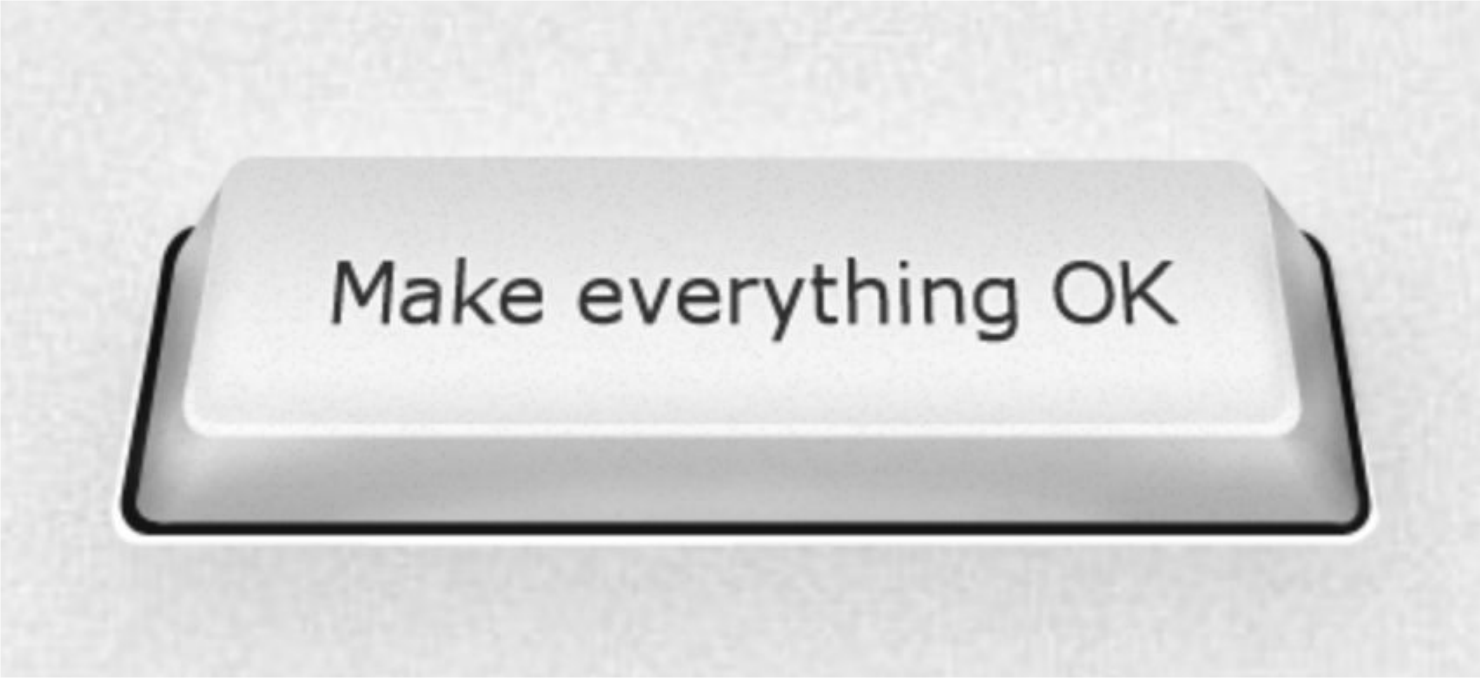
You are only delaying the inevitable

STORMATICS

You tested your application here …

… and this is what production looks like

There is no magic button or setting …



STORMATICS

Scaling PostgreSQL

A Developer's Guide

- Query & SQL Optimization
- Performance Features
- Architectural Improvements
- Parameter Tuning

Query & SQL Optimization

STORMATICS

pg_stat_statements is your friend

- PostgreSQL extension, included in distribution and off by default
- Logs statistics about SQL statements
- Easy stats to watch out for
  - Long running (`mean_exec_time`)
  - Most frequent (`calls`)
  - Standard deviation in execution time (`stddev_exec_time`)
  - I/O intensive (`blk_read_time,blk_write_time`)

Explain plan is your friend

```
                                                    QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------
Unique  (cost=22.67..22.70 rows=2 width=44) (actual time=0.066..0.067 rows=1 loops=1)
  ->  Sort  (cost=22.67..22.68 rows=3 width=44) (actual time=0.065..0.065 rows=2 loops=1)
        Sort Key: la.account_id, la.external_entity_id, la.created_at
        Sort Method: quicksort  Memory: 25kB
        ->  Hash Join  (cost=10.66..22.65 rows=3 width=44) (actual time=0.048..0.052 rows=2 loops=1)
              Hash Cond: (la.loan_application_status_id = loan_application_statuses.loan_application_status_id)
              ->  Bitmap Heap Scan on loan_applications la  (cost=9.21..21.16 rows=3 width=36) (actual time=0.022..0.025 rows=2 loops=1)
                    Recheck Cond: ((account_id = ANY ('{7812011}'::integer[])) OR (external_entity_id = ANY ('{NULL}'::integer[])))
                    ->  BitmapOr  (cost=9.21..9.21 rows=3 width=0) (actual time=0.016..0.016 rows=0 loops=1)
                          ->  Bitmap Index Scan on index_loan_applications_on_account_id  (cost=0.00..4.62 rows=3 width=0) (actual time=0.014..0.014 rows=2 loops=1)
                                Index Cond: (account_id = ANY ('{7812011}'::integer[]))
                          ->  Bitmap Index Scan on index_loan_applications_on_external_entity_id  (cost=0.00..4.60 rows=1 width=0) (actual time=0.001..0.001 rows=0 loops=1)
                                Index Cond: (external_entity_id = ANY ('{NULL}'::integer[]))
              ->  Hash  (cost=1.20..1.20 rows=20 width=16) (actual time=0.016..0.016 rows=20 loops=1)
                    Buckets: 1024  Batches: 1  Memory Usage: 1kB
                    ->  Seq Scan on loan_application_statuses  (cost=0.00..1.20 rows=20 width=16) (actual time=0.004..0.007 rows=20 loops=1)
 Total runtime: 0.122 ms
(17 rows)
```

Node: What is happening in this step? Feed result to parent Node.

Relation: What is it happening on? Table or result of child Node?

Cost: ***Relatively*** how expensive is this step?

Modifier: Tweak result before handoff.

Rows: How many rows will be returned by this Node.

Loops: How many times will this step be executed.

Watch out for locks!

Session 1

```
BEGIN;
UPDATE foo SET … WHERE id = 1;
UPDATE foo SET … WHERE id = 2;
UPDATE foo SET … WHERE id = 3;
COMMIT;
```

Locks

Session 2

```
UPDATE foo SET … WHERE id = 1;
```

(waits)

Performance Features

Indexes

- B-Tree - *default index*
- Hash - *equality checks*
- Composite - *multi column*
- Partial - *conditional index on subset of data*
- Covering - *includes an additional column*
- BRIN (block range index) - *space efficient for sorted tables*

Indexes - Not a one-size-fits-all!

- You need all or most of the data any ways
- Your workload is WRITE or UPDATE heavy with little READs
- ‘Over’ indexing can cause data bloat
- Your table is too small

Many performance features ‘just work’

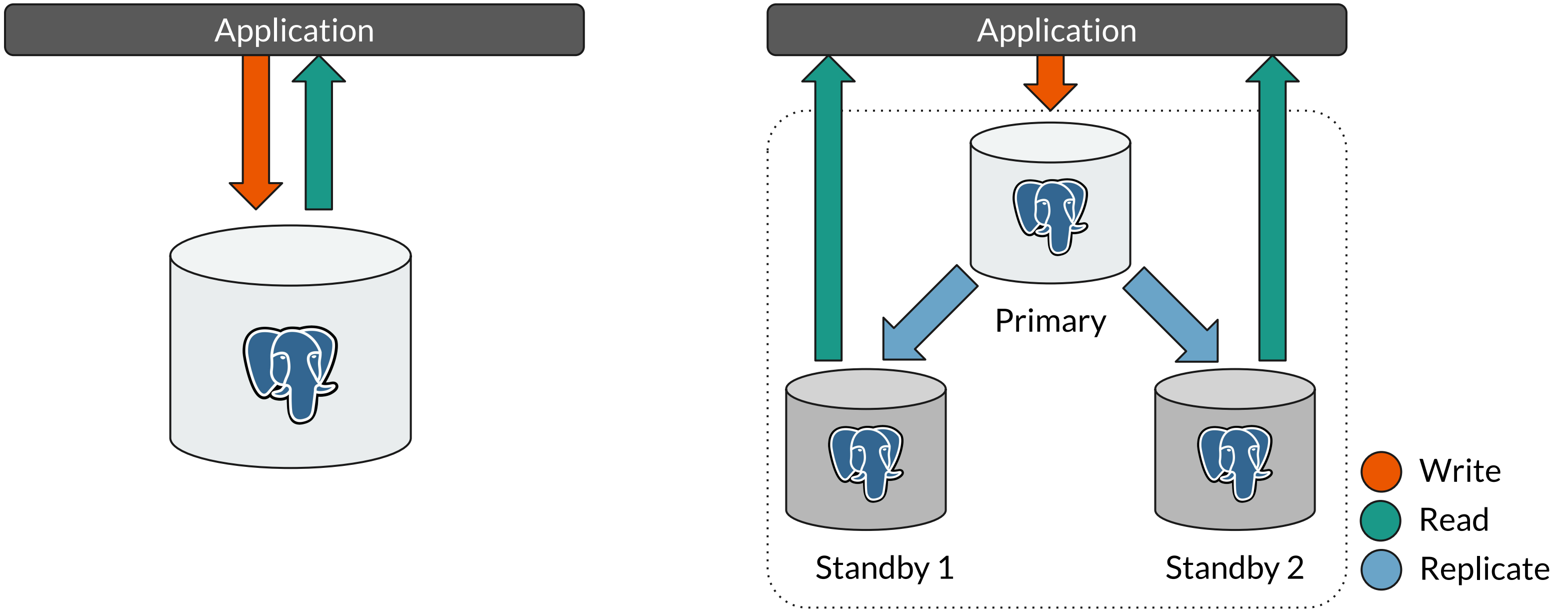
A few examples …

- Parallel queries
- Heap-Only Tuples (HOT)
- Incremental sort
- Autovacuum

Architectural Improvements

STORMATICS

Load Balancing



Load Balancing

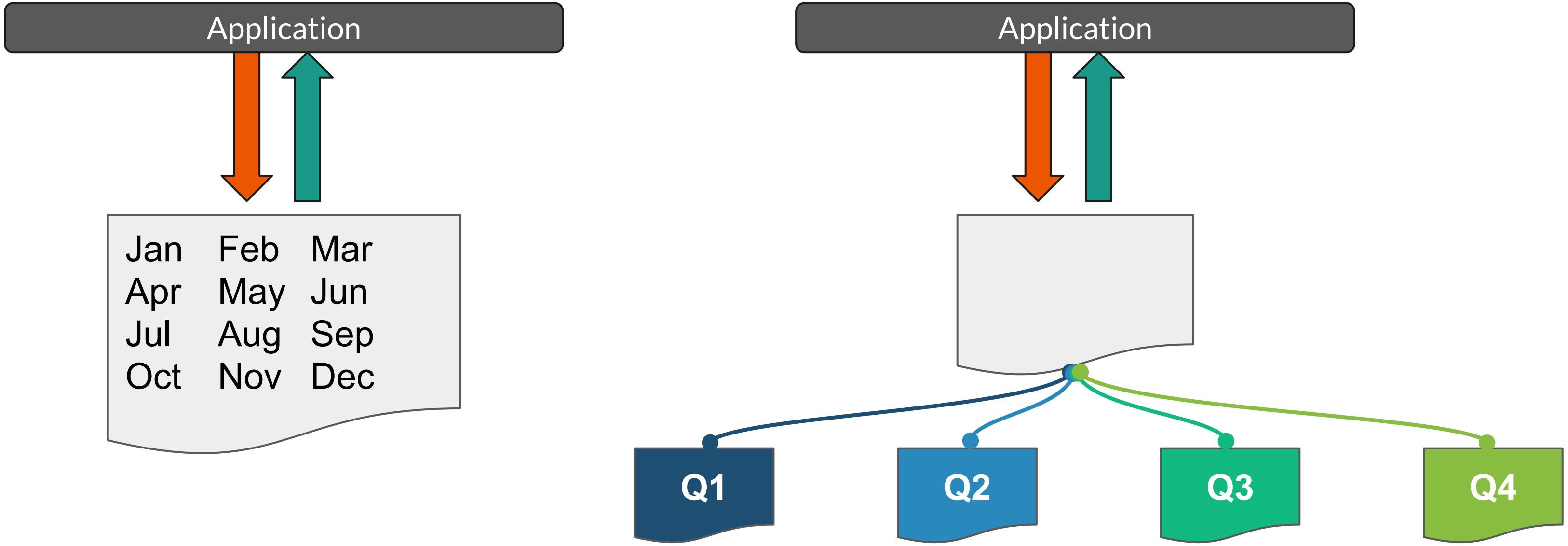
Single Node SELECTs

```
transaction type: <builtin: select only>
scaling factor: 10
query mode: simple
number of clients: 25
number of threads: 1
maximum number of tries: 1
duration: 60 s
number of transactions actually processed: 19139
number of failed transactions: 0 (0.000%)
latency average = 67.215 ms
initial connection time = 8620.897 ms
tps = 371.939402 (without initial connection time)
```

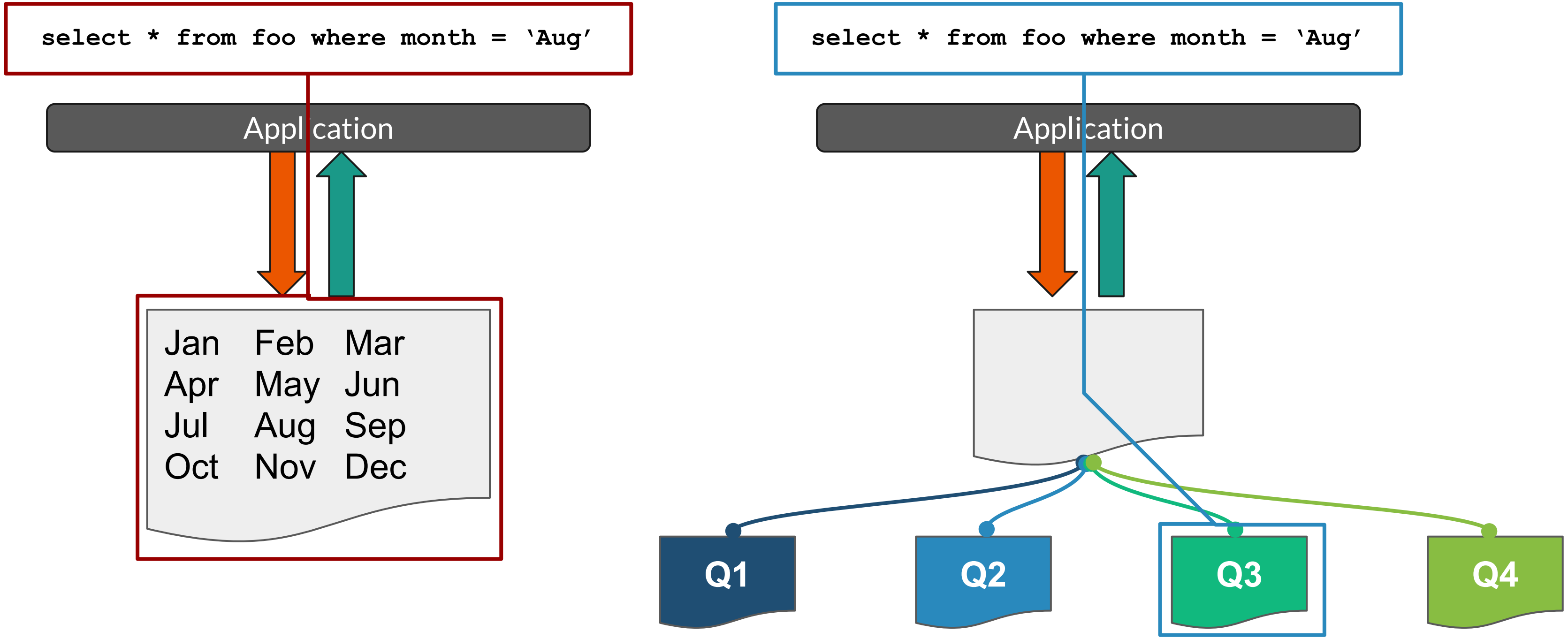
Load Balanced 3-node Cluster

```
transaction type: <builtin: select only>
scaling factor: 10
query mode: simple
number of clients: 25
number of threads: 1
maximum number of tries: 1
duration: 60 s
number of transactions actually processed: 24885
number of failed transactions: 0 (0.000%)
latency average = 51.449 ms
initial connection time = 8896.110 ms
tps = 485.918972 (without initial connection time)
```

Partitioning



Partitioning



Parameter Tuning

STORMATICS

Easily tuned database parameters

- **Most defaults are good enough!**
- shared_buffers
  - Cache for frequently accessed data
  - Default is 128MB
  - Recommended is between 25% and 40% of system memory
- wal_buffers
  - Shared memory not yet written to disk
  - Default is 3% of shared_buffers
  - A value of up to 16MB can improve performance in high concurrency commits
- work_mem
  - Memory available for a query operation
  - Default is 4MB
  - High I/O activity for a query is an indicator that an increase in work_mem can help
  - Each parallel operation is allowed to use memory up to this value

Easily tuned database parameters

- maintenance_work_mem
  - Memory used by maintenance operations like VACUUM and ANALYZE
  - Default is 64MB
  - Higher values can improve maintenance performance
  - Each worker is allowed to use up to this value
- effective_cache_size
  - Value of effective disk cache to be used by query planner
  - Not an allocation!
  - Default is 4GB
  - Higher values encourage index scans
- random_page_cost
  - Value of non-sequential disk page access cost
  - Not an allocation!
  - Default is 4.0
  - Lower values encourage index scans

Conclusion

Database performance involves a lot of variables. Optimize how data is accessed before scaling by credit card!

Questions?

pg_umair