



# Lessons learned from **auto-tuning** PostgreSQL

**PG Conf NYC**

Sep-Oct 2024



**Costa Alexoglou**  
Engineering Lead

# Agenda



1. Who am I?
2. Intro to DBtune
3. What is database tuning
4. Tuning tuning tuning
5. What is coming with DBtune v2
6. Community posts

# Who am I



@CostasAlexoglou



Costa Alexoglou

## 2017

MSc  
Electrical & Computer  
Engineering


## 2021

Joined  Neo4j  
to start Ops-Manager, in-  
house observability and  
ops product

## 2017 – 2021

Founded  
VisualEyes  
(Acquired by Neurons 🇩🇰)

## 2024

Joined  dbtune  
to help high performing teams  
optimize their PostgreSQL  
databases ⚡

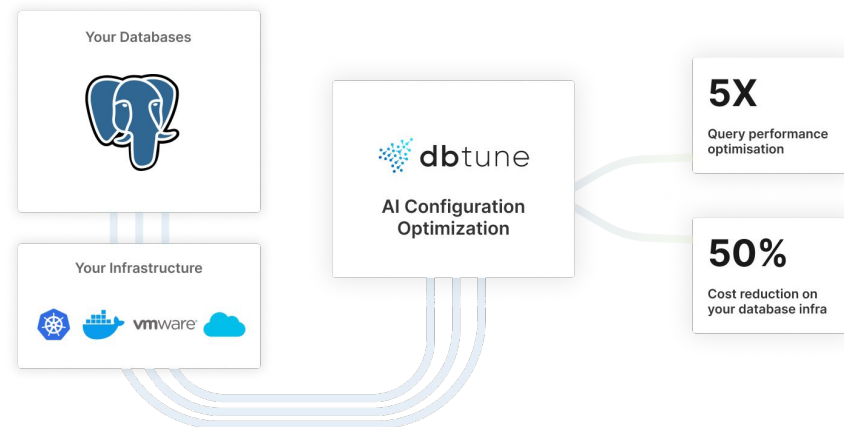
# What is DBtune



DBtune is an **AI-powered database parameter tuning service**.

Spun out of research at Stanford University, DBtune autonomously optimizes the configuration of databases through machine learning.

It **observes, iterates and adapts until converging and delivering the optimal settings** for any individual workload, use case and machine.

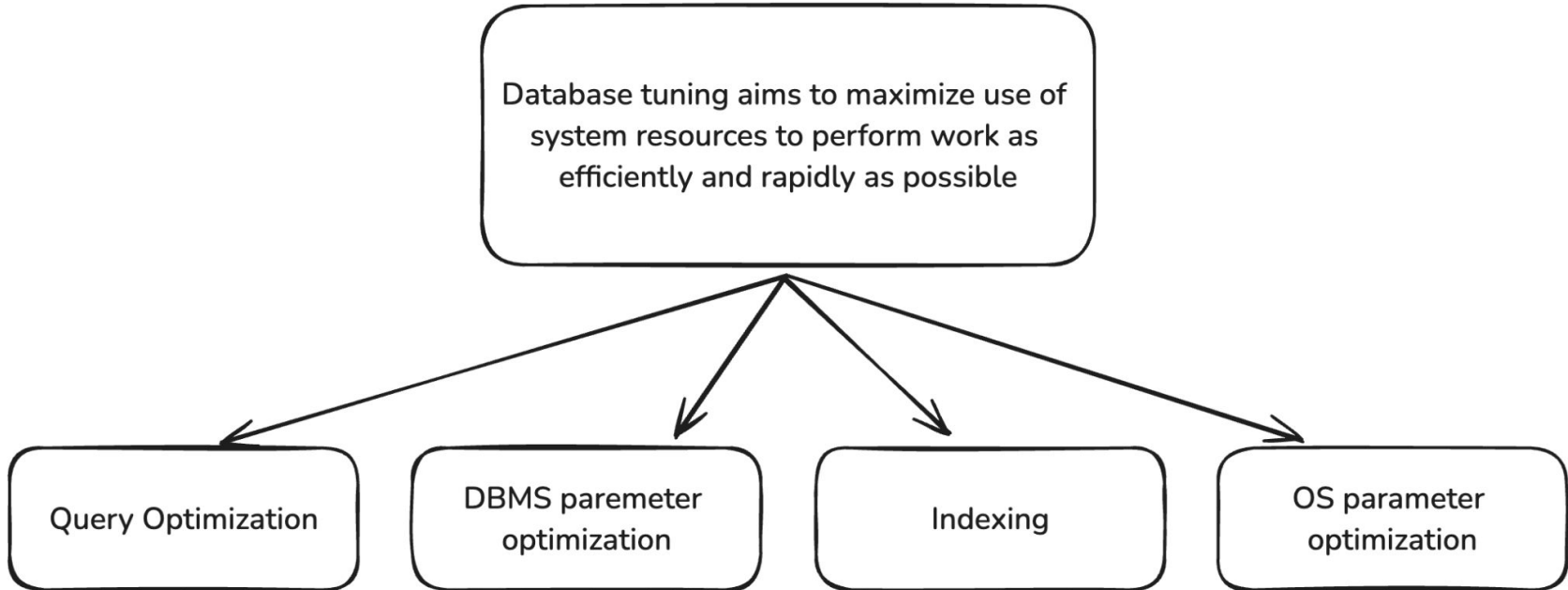


# What is database tuning

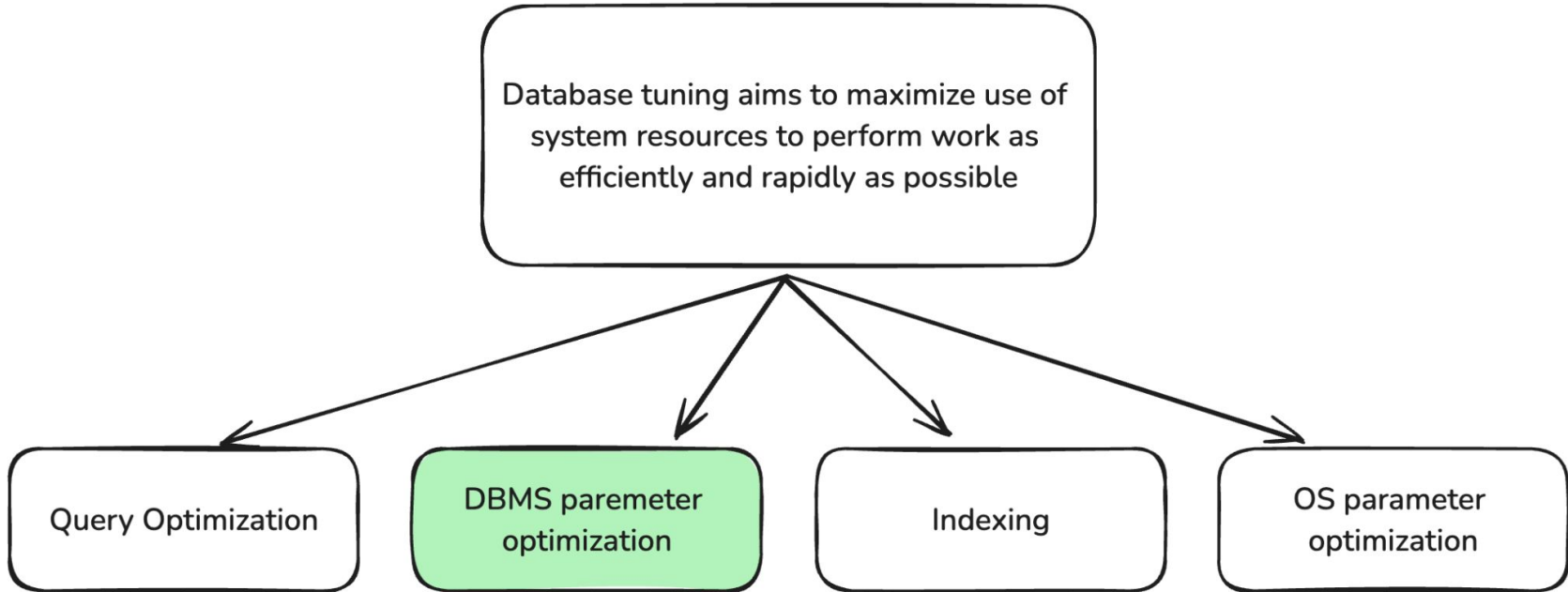


Database tuning aims to maximize use of system resources to perform work as efficiently and rapidly as possible

# What is database tuning



# What is database tuning



# Why do we need tuning



- **Databases change, grow and slow down**
- **Not all workloads and machines are the same**



# Why does it matter



## Technical perspective

1. Better system performance
  - a. Avg query runtime (latency)
  - b. Throughput (tx / s)
2. Better scalability
3. Better resource utilization
4. Better maintenance (vacuum, backups etc.)

# Why does it matter



## Technical perspective

1. Better system performance
  - a. Avg query runtime (latency)
  - b. Throughput (tx / s)
2. Better scalability
3. Better resource utilization
4. Better maintenance (vacuum, backups etc.)

## Business perspective

1. Avoidance of Over-Provisioning
2. Decrease cloud/on-prem spending
3. Faster response time (100ms for 1%)
4. Reduces downtime prob
5. Sustainability (ESG)

# What is database parameter tuning



- **Adjusting knobs to fit the current environment and workload**
- **PostgreSQL parameters that are typically important**  
`work_mem`, `shared_buffers`, `max_wal_size` etc
- **`shared_buffers`**  
How many pages we will cache in memory before reaching OS cache or filesystem.
- **`work_mem`**  
Sets the base maximum amount of memory to be used by a query operation (such as a sort or hash table) before writing to temporary disk files.



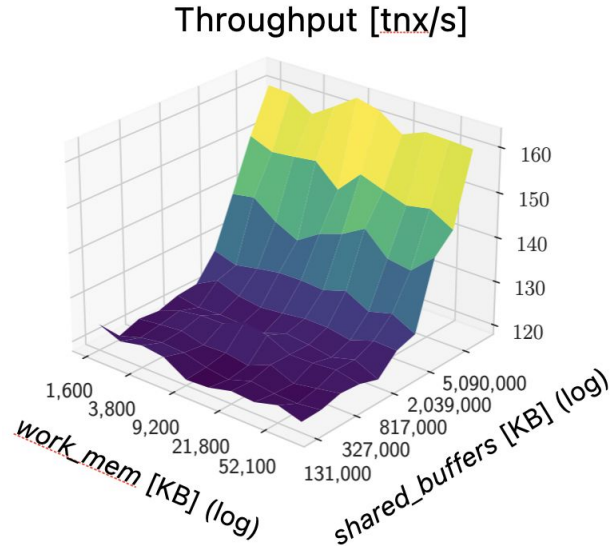


# Time for some plots



# Throughput (tx/s)

work\_mem and shared\_buffers tuning



## Resource Stresser benchmark

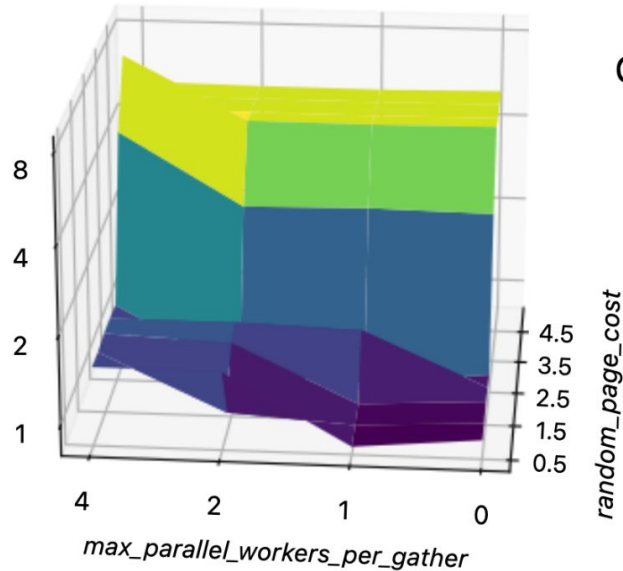
- Increasing **shared\_buffers** is important
- **work\_mem** is not - queries are simple



# Average query runtime (latency)

max\_parallel\_workers\_per\_gather and random\_page\_cost tuning

Epinions



Query runtime in ms  
**Lower the better**

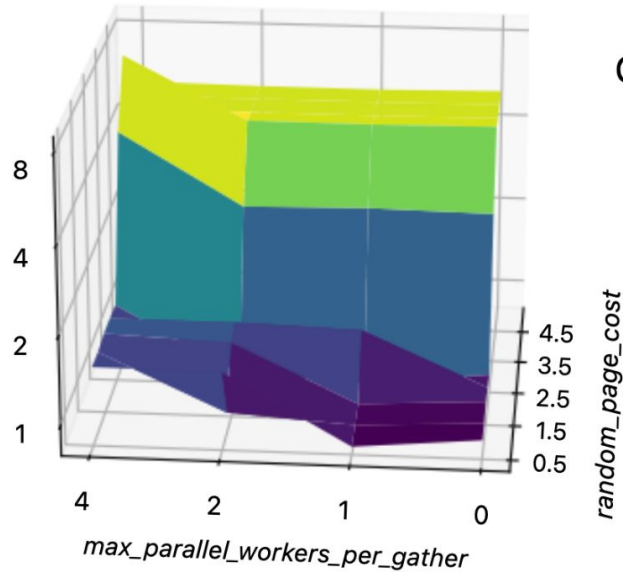




# Average query runtime (latency)

max\_parallel\_workers\_per\_gather and random\_page\_cost tuning

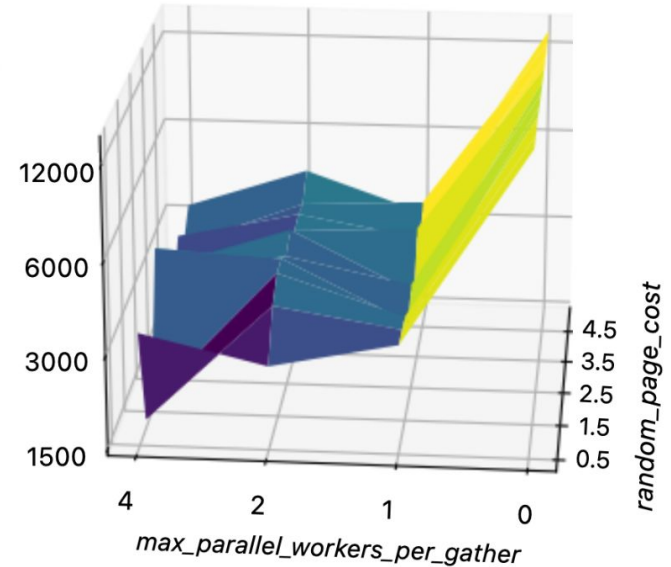
Epinions



Query runtime in ms  
**Lower the better**



TPC-H



# Growing complexity



## Tuning

1. shared\_buffers → **10 possible values each**
2. work\_mem
3. random\_page\_cost
4. seq\_page\_cost
5. checkpoint\_completion\_target
6. effective\_io\_concurrency
7. max\_parallel\_workers\_per\_gather
8. max\_worker\_processes
9. max\_wal\_size
10. min\_wal\_size
11. bgwriter\_lru\_maxpages
12. bgwriter\_delay

[www.dbtune.com](http://www.dbtune.com)

Docs: <https://docs.dbtune.com/Supported%20databases/PostgreSQL>



# Growing complexity



## Tuning

1. shared\_buffers → **10 possible values each**
2. work\_mem
3. random\_page\_cost
4. seq\_page\_cost
5. checkpoint\_completion\_target
6. effective\_io\_concurrency
7. max\_parallel\_workers\_per\_gather
8. max\_worker\_processes
9. max\_wal\_size
10. min\_wal\_size
11. bgwriter\_lru\_maxpages
12. bgwriter\_delay



**What is the total number of combinations?**

# Growing complexity



## Tuning

1. shared\_buffers → **10 possible values each**
2. work\_mem
3. random\_page\_cost
4. seq\_page\_cost
5. checkpoint\_completion\_target
6. effective\_io\_concurrency
7. max\_parallel\_workers\_per\_gather
8. max\_worker\_processes
9. max\_wal\_size
10. min\_wal\_size
11. bgwriter\_lru\_maxpages
12. bgwriter\_delay

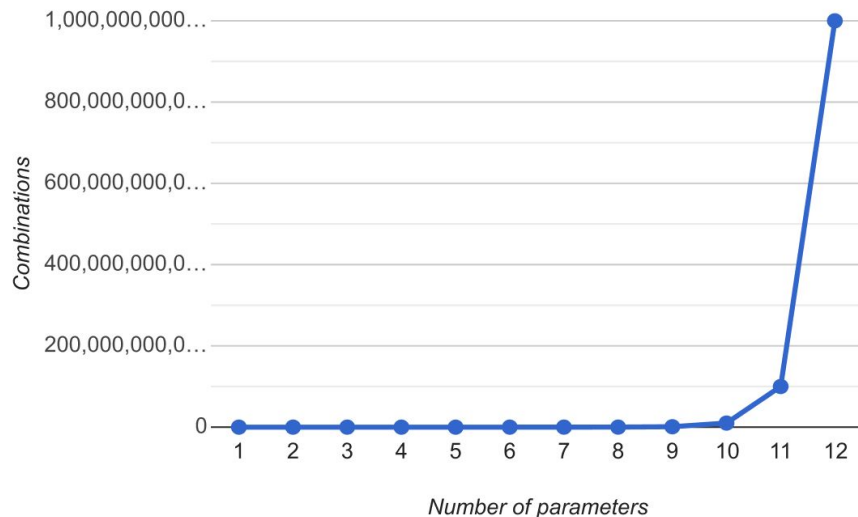
$$10^{12} = 10 \times 10 \times 10 \times 10 \times 10 \times 10 \times 10 \times 10 \times 10 \times 10 \times 10 \times 10 \times 10 \times 10 \times 10 \times 10 = 1 \text{ Trillion combinations}$$

# Growing complexity

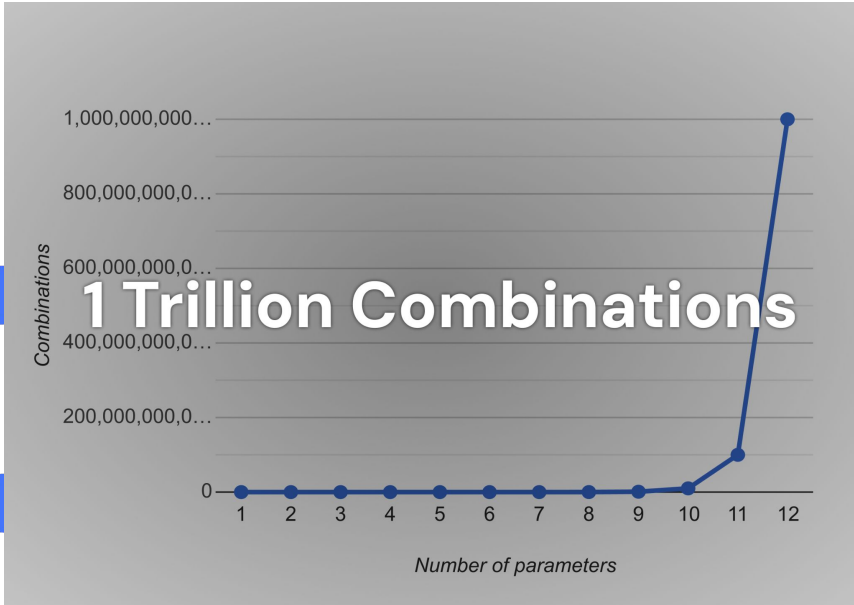


## Tuning

1. shared\_buffers → **10 possible values each**
2. work\_mem
3. random\_page\_cost
4. seq\_page\_cost
5. checkpoint\_completion\_target
6. effective\_io\_concurrency
7. max\_parallel\_workers\_per\_gather
8. max\_worker\_processes
9. max\_wal\_size
10. min\_wal\_size
11. bgwriter\_lru\_maxpages
12. bgwriter\_delay



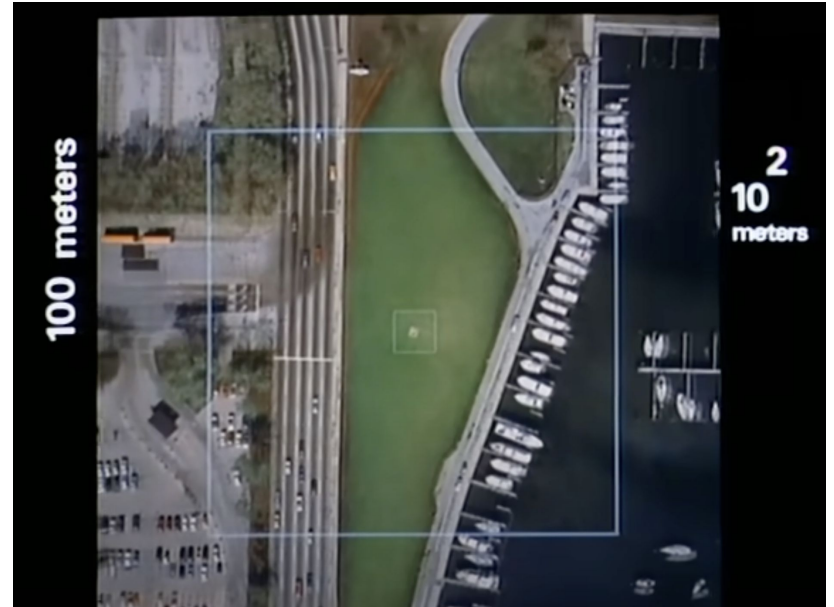
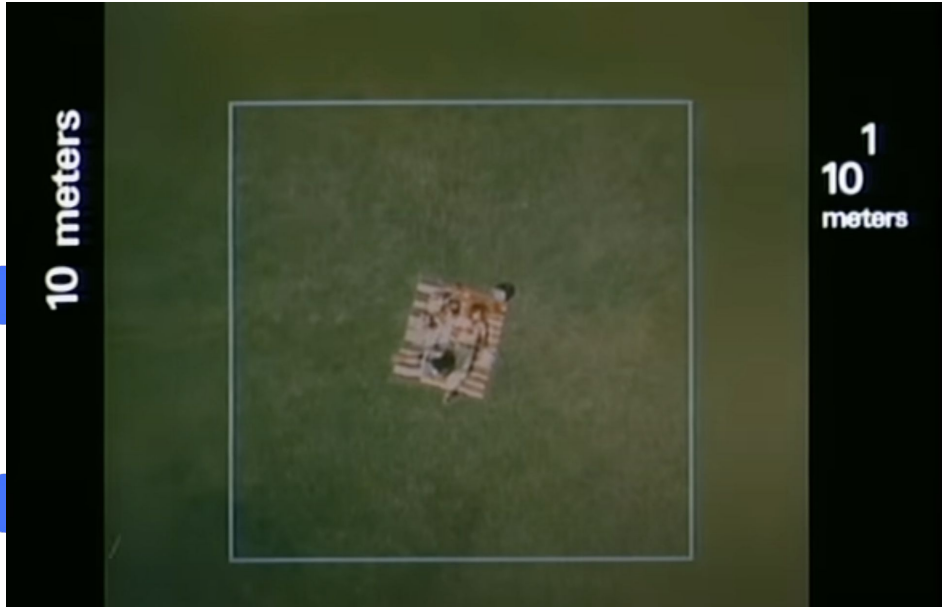
# Growing complexity



[www.dbtune.com](http://www.dbtune.com)

\* Power of Ten (YouTube)

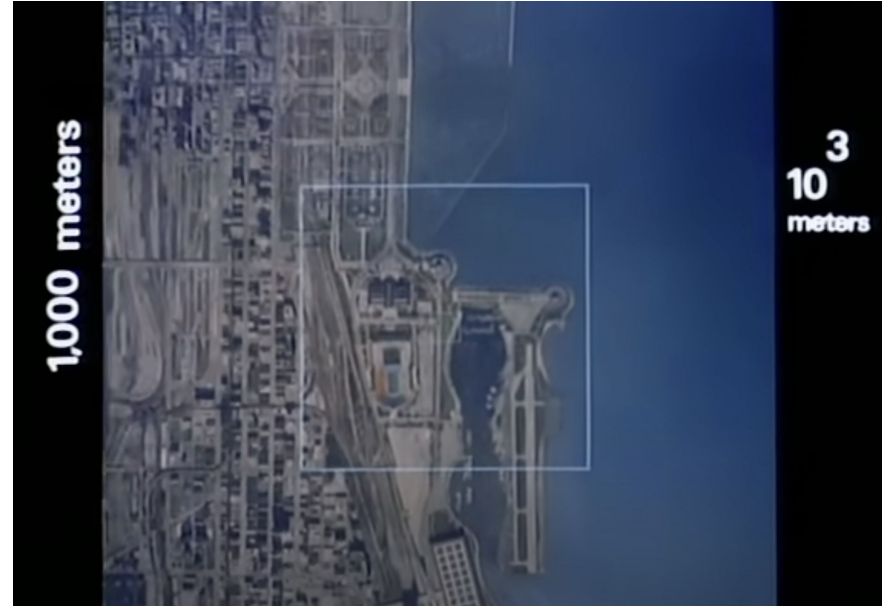
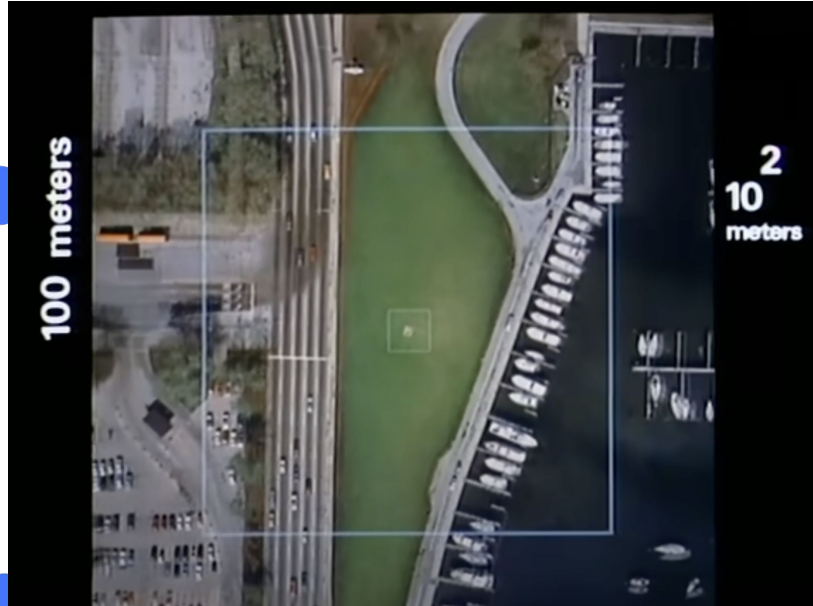
# Growing complexity



[www.dbtune.com](http://www.dbtune.com)

\* Power of Ten (YouTube)

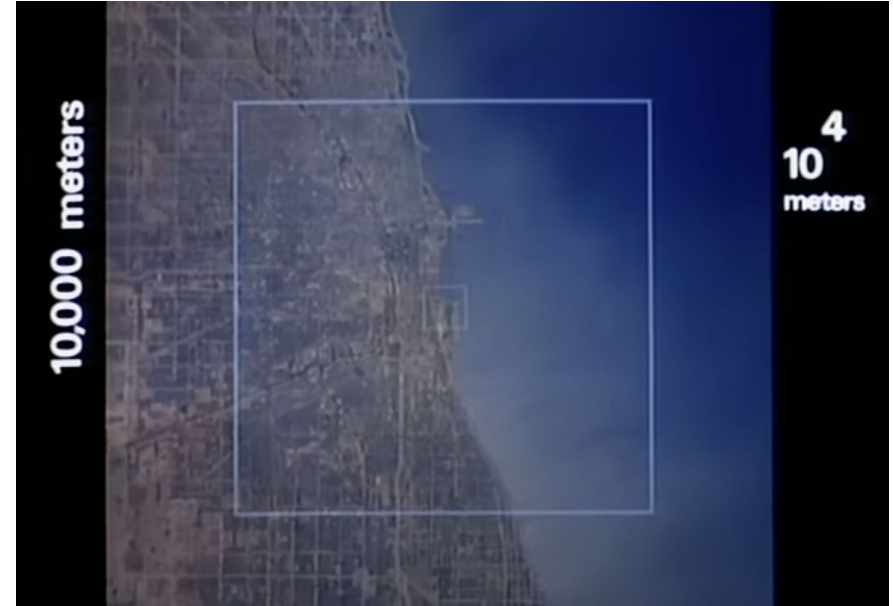
# Growing complexity



[www.dbtune.com](http://www.dbtune.com)

\* Power of Ten (YouTube)

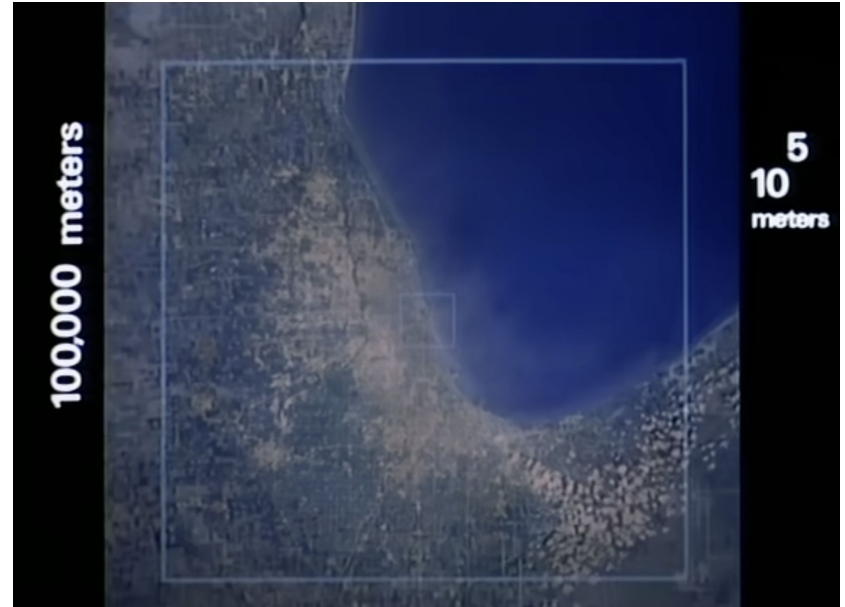
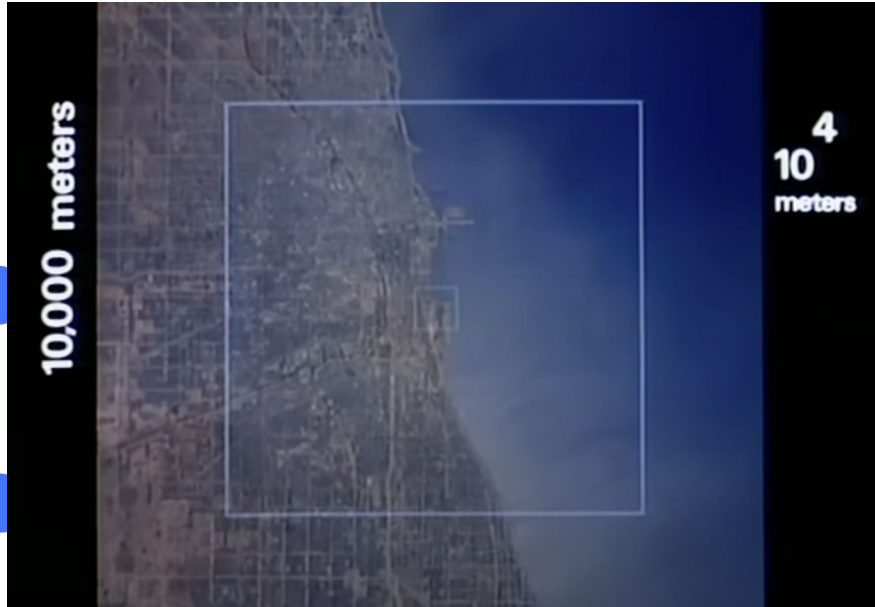
# Growing complexity



[www.dbtune.com](http://www.dbtune.com)

\* Power of Ten (YouTube)

# Growing complexity

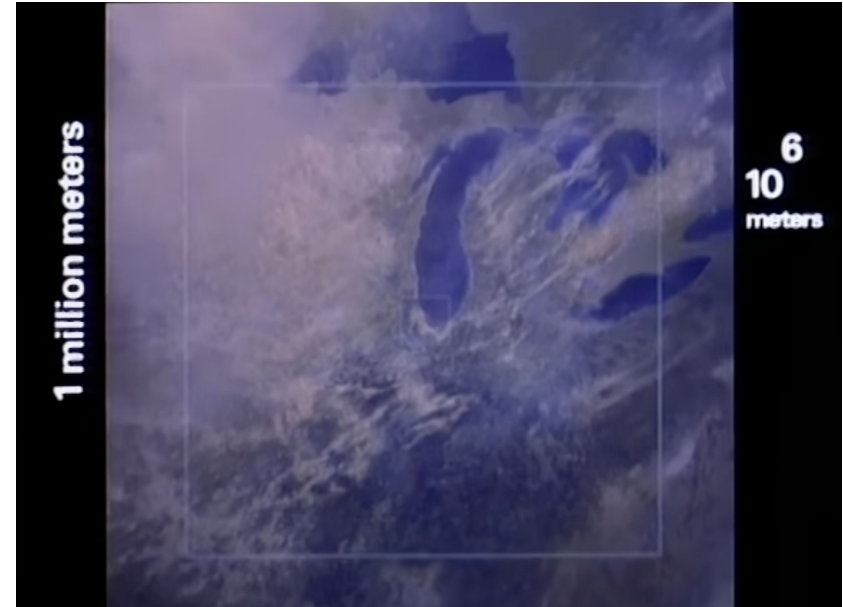
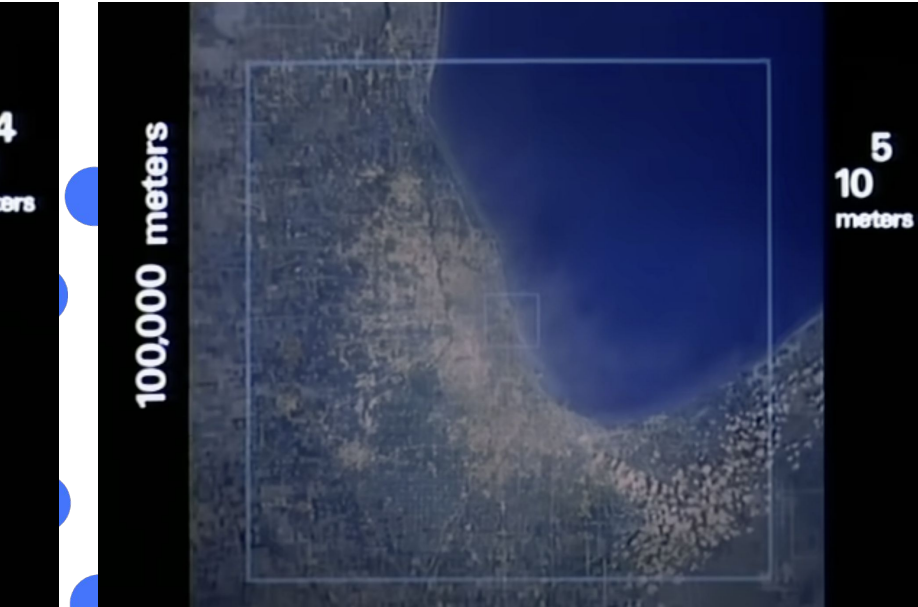


[www.dbtune.com](http://www.dbtune.com)

\* Power of Ten (YouTube)



# Growing complexity



[www.dbtune.com](http://www.dbtune.com)

\* Power of Ten (YouTube)

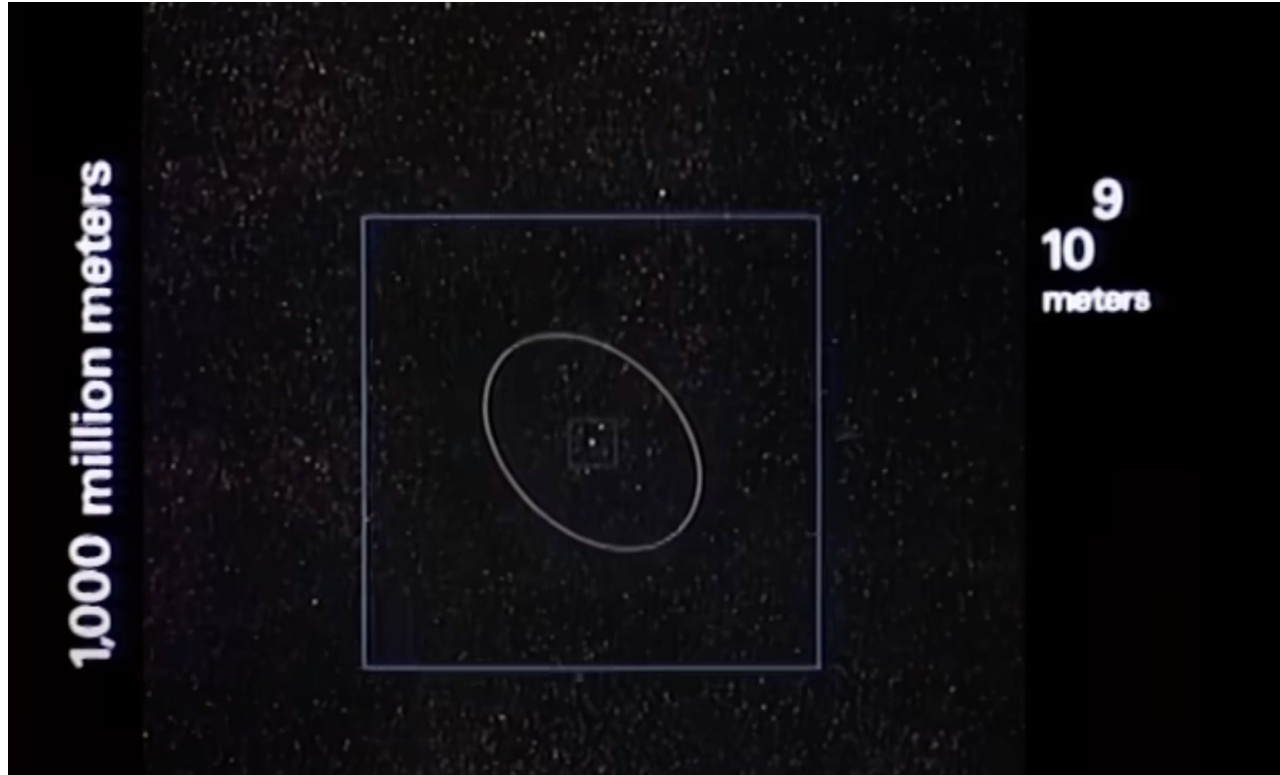
# Growing complexity



[www.dbtune.com](http://www.dbtune.com)

\* Power of Ten (YouTube)

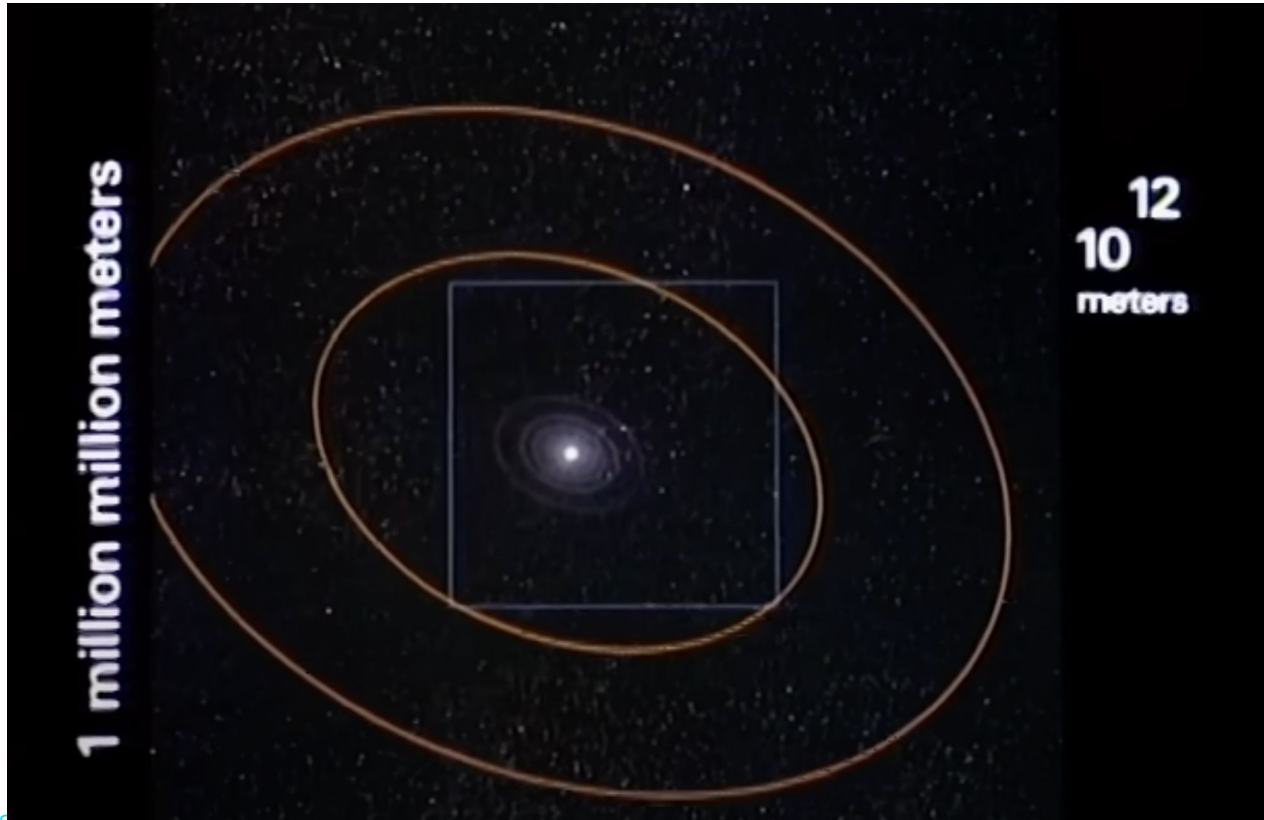
# Growing complexity



[www.dbtune.com](http://www.dbtune.com)

\* Power of Ten (YouTube)

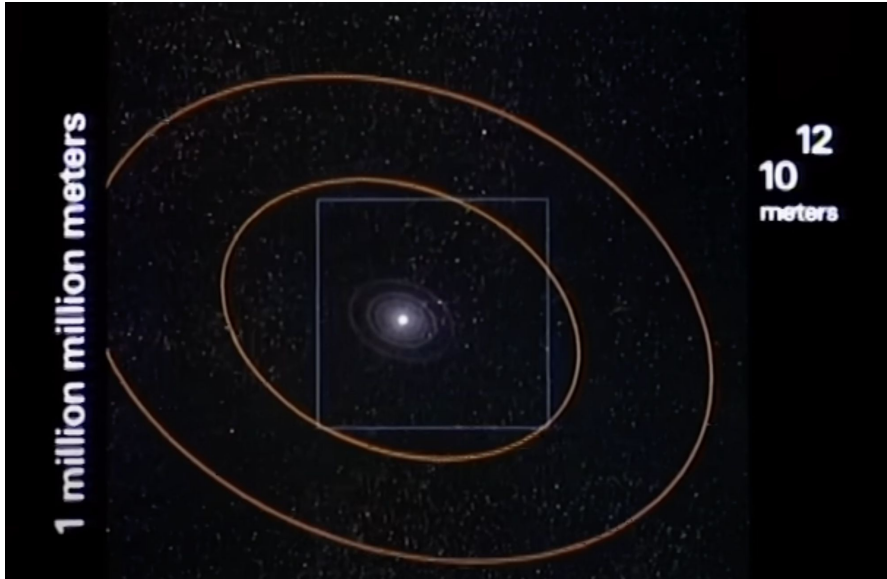
# Growing complexity



[www.dbtune.com](http://www.dbtune.com)

\* Power of Ten (YouTube)

# Growing complexity

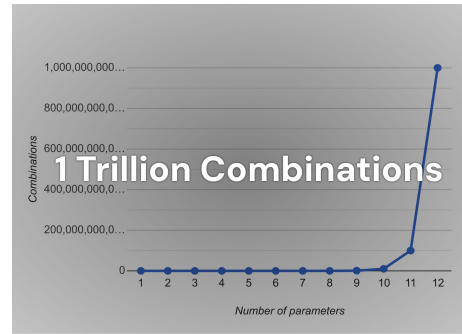


Goes in reverse to  
 $0.01 \text{ \AA} = 10^{-12} \text{ meters}$

[www.dbtune.com](http://www.dbtune.com)

\* Power of Ten (YouTube)

# Growing complexity



**Computers are fast. Can we test them all? (bruteforce)**

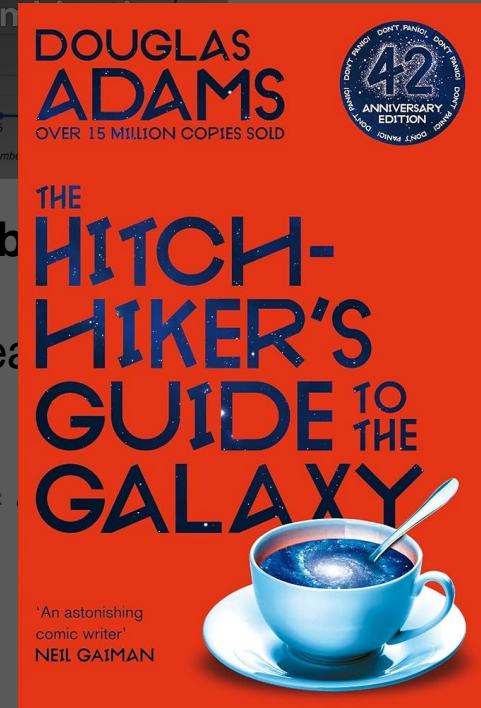
5 minutes per proposed configuration would lead to

$$5,000,000,000,000 \text{ minutes} = \frac{5,000,000,000,000}{60} \text{ hours} = 83.333.333.333 \text{ hours}$$
$$\approx 9,504,273.50 \text{ years}$$

# Growing complexity

More than the time the supercomputer in The Hitchhiker's Guide to the Galaxy, took to calculate the answer to Life and Everything (spoiler: 42).

It took 7.5 million years.

$$5,000,000,000,000 \text{ minutes} = \frac{5,000,000,000,000}{60} \text{ hours} =$$
$$\approx 9,504,273.50 \text{ years}$$


# How parameter tuning is happening now?



## Manual



Tuning Guru

**Slow**

Tuning takes days

**Painstaking**

Need high expertise

**Ineffective**

What happens with large  
fleets



# How parameter tuning is happening now?



## Manual



Tuning Guru

**Slow**

Tuning takes days

**Painstaking**

Tuning takes days

**Ineffective**

Tuning takes days

**Inadequate**

Seasonal workload

## Heuristics



PGTune



**One size fits all**

Generic rules (not adapting)

**Workload agnostic**

Tuning takes days

**Ineffective**

Tune again when you  
change infrastructure

**Inadequate**

Tuning takes days

# How parameter tuning is happening now?



## Manual



Tuning Guru

**Slow**

Tuning takes days

**Painstaking**

Tuning takes days

**Ineffective**

Tuning takes days

**Inadequate**

Tuning takes days

## Heuristics



PGTune



POSTGRESQL  
CONFIGURATOR

**One size fits all**

Generic rules (not adapting)

**Workload agnostic**

Tuning takes days

**Ineffective**

Tune again when you  
change infrastructure

**Inadequate**

Tuning takes days

## ML based



dbtune

**Fast**

Tuning takes hours

**Performance**

Does not leave performance  
on the table

**Workload specific**

Tunes based on the  
workload and not only the  
machine

# How often do you tune?

## Examples cases



### Frequent cases

1. Your workload changes —  
Change queries and application
2. Your database grows and  
changes
3. You scale your instances — Up  
or down

### Infrequent cases

1. You migrate from on-prem to the  
cloud — or vice-versa
2. You migrate DBMS — E.g., from  
Oracle to PostgreSQL
3. You upgrade your version of  
PostgreSQL

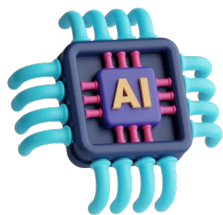


# How often do you tune? (the reality)

The sad reality

- Tuning is typically reactive to something going wrong – **Not proactive**
- Maybe looked at once or twice a year – as part of maintenance work
- Often engage expensive external experts, in-house expertise is missing
- Different workloads are not treated differently
- Modus operandi: Throw more hardware / compute at any issue (\$\$\$)

# Why DBtune exists



## Machine learning approach

DBtune learns how to solve optimization challenges



## Easy to work

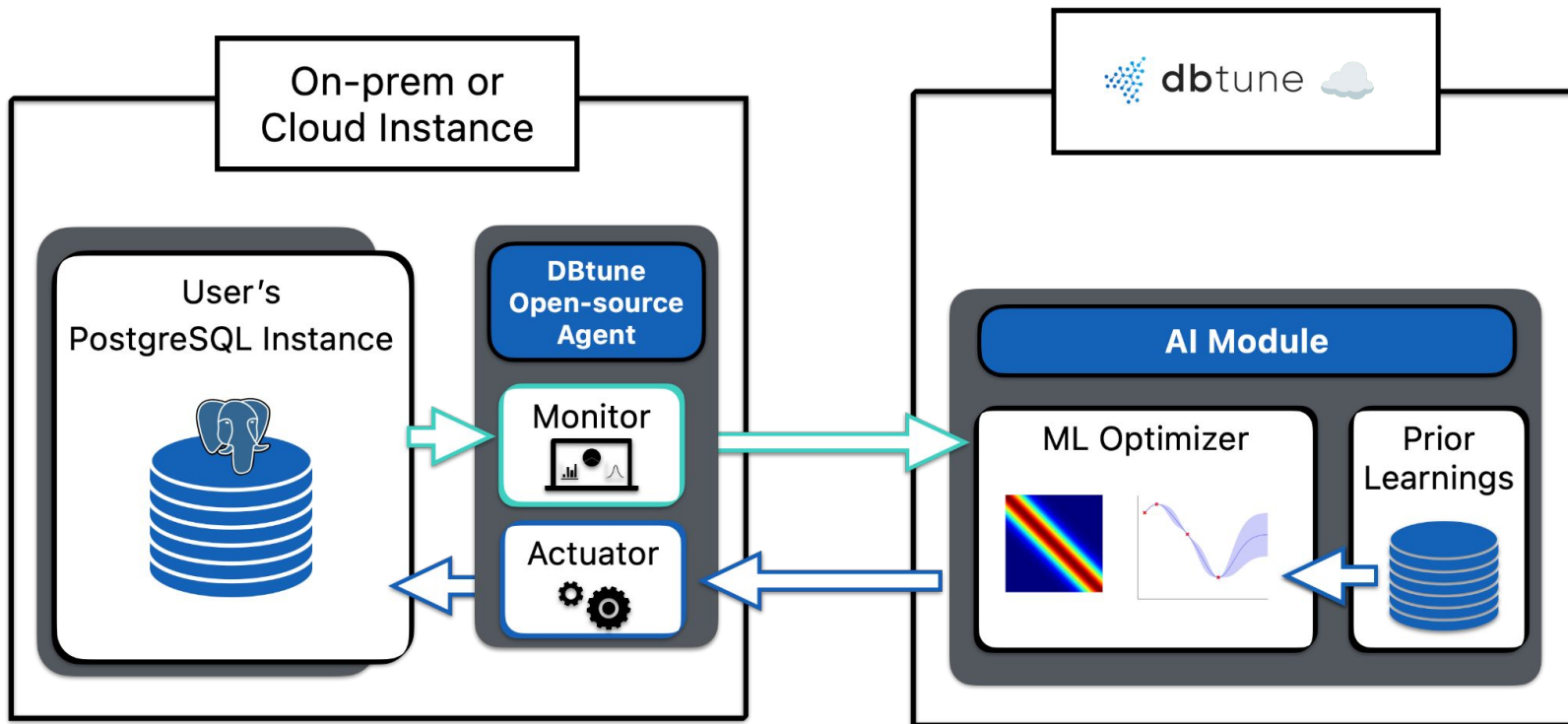
No need for background in ML or database tuning



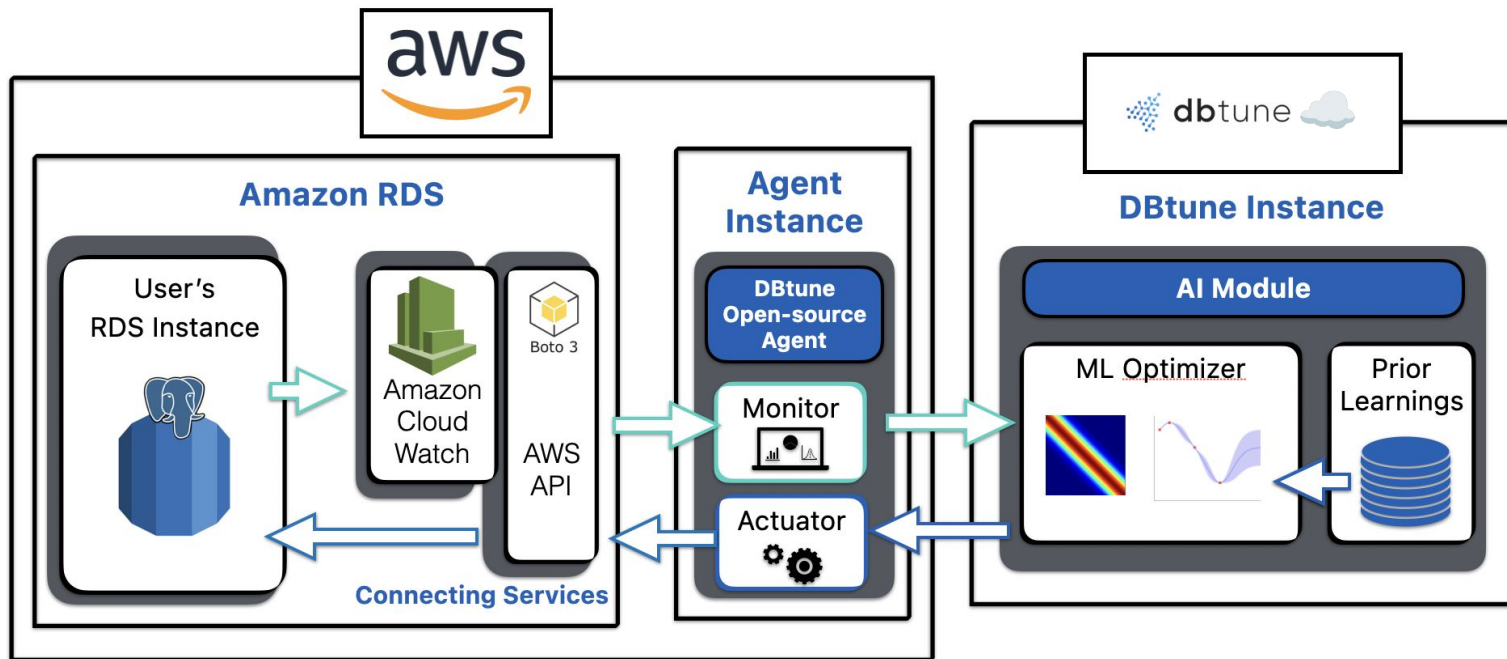
## Operational scalability

Companies with 10s or 1000s of Databases can tune with less humans in the loop.

# How DBtune works



# How DBtune works



# Parameters we tune



## Database Reload (11 parameters)

1. `work_mem`
2. `random_page_cost`
3. `seq_page_cost`
4. `checkpoint_completion_target`
5. `effective_io_concurrency`
6. `max_parallel_workers_per_gather`
7. `max_parallel_workers`
8. `max_wal_size`
9. `min_wal_size`
10. `bgwriter_lru_maxpages`
11. `bgwriter_delay`

## Database Restart (+2 parameters)

1. `shared_buffers`
2. `max_worker_processes`

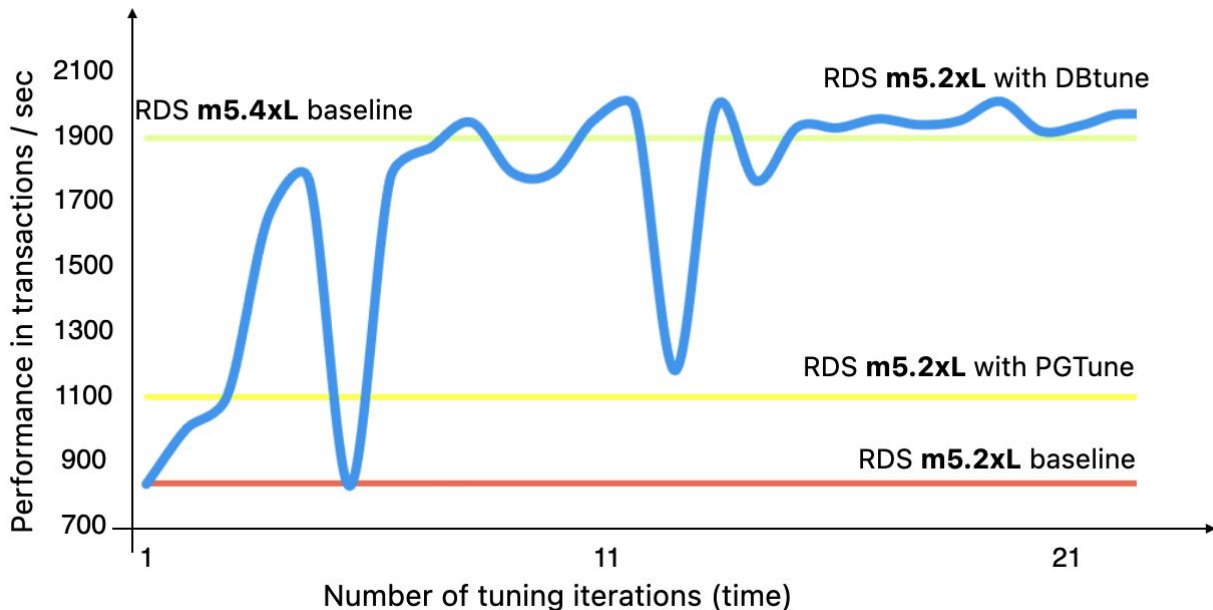




**Time for more plots** 

# Auto-tuning AWS RDS PostgreSQL

## Stop over-provisioning

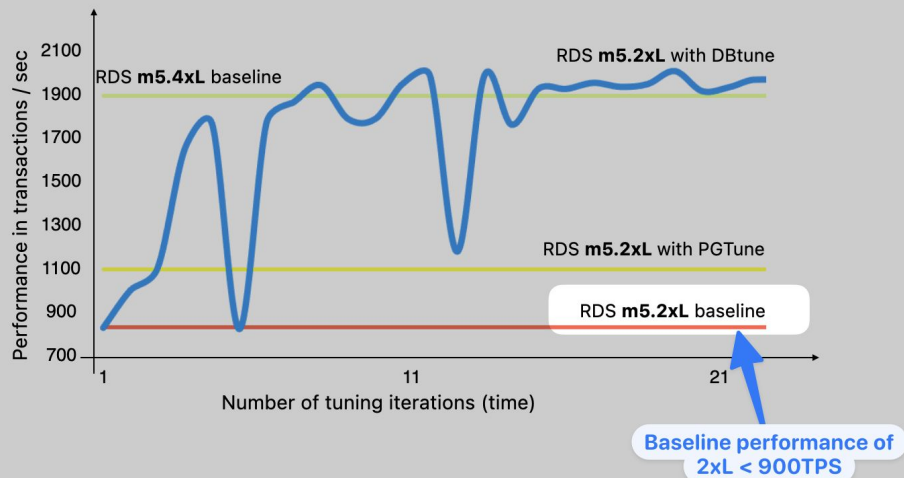


**TPCC benchmark** on a m5.2/4xL instances.

TL;dr:  
DBTune doubles the performance of PostgreSQL Amazon RDS

# Auto-tuning RDS PostgreSQL

## Stop over-provisioning

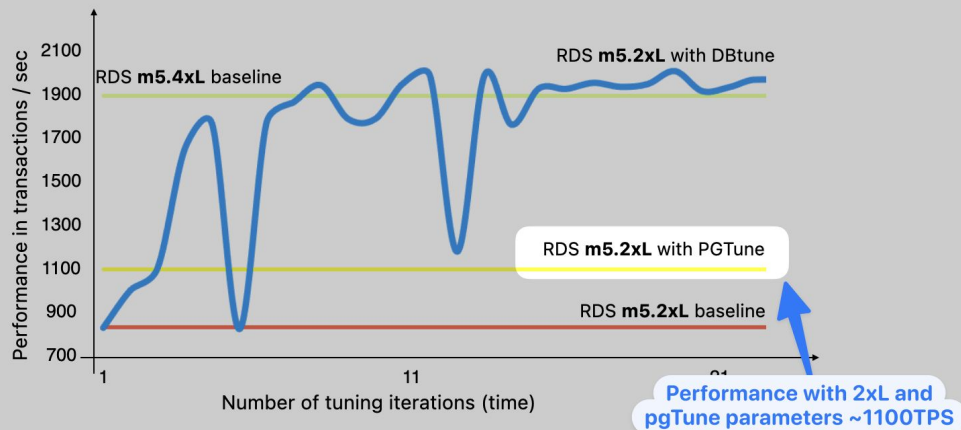


**TPCC benchmark** on a md.2/4xL instances.

**TL;dr:**  
DBtune doubles the performance of PostgreSQL Amazon RDS

# Auto-tuning RDS PostgreSQL

## Stop over-provisioning

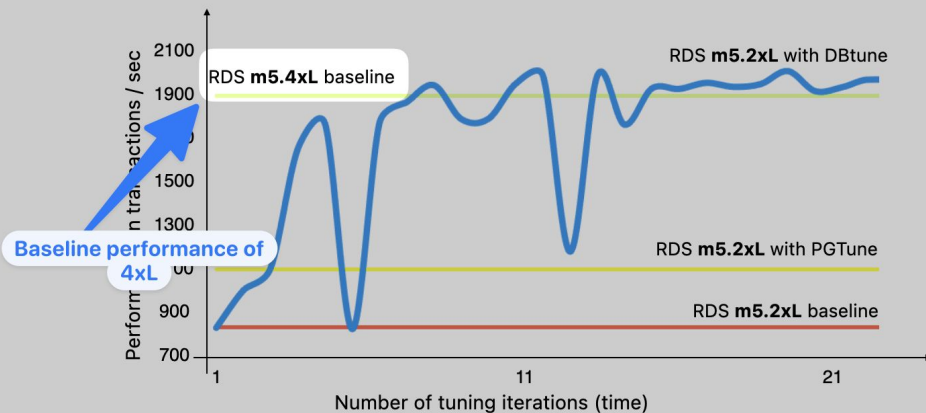


**TPCC benchmark** on a md.2/4xL instances.

**TL;dr:**  
DBtune doubles the performance of PostgreSQL Amazon RDS

# Auto-tuning RDS PostgreSQL

## Stop over-provisioning

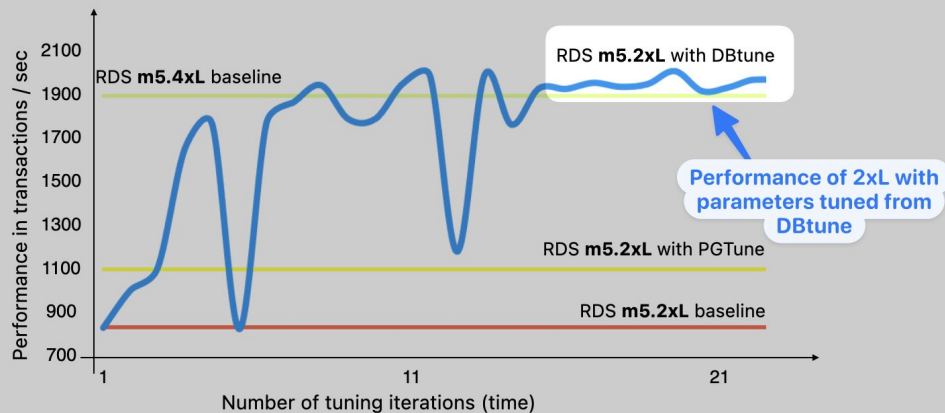


**TPCC benchmark** on a md.2/4xL instances.

**Tl;dr:**  
DBtune doubles the performance of PostgreSQL Amazon RDS

# Auto-tuning RDS PostgreSQL

## Stop over-provisioning



**TPCC benchmark** on a md.2/4xL instances.

**TL;dr:**  
DBtune doubles the performance of PostgreSQL Amazon RDS

# Auto-tuning RDS PostgreSQL

## Stop over-provisioning



Hardware				Cost / Year		
AWS RDS Instance Type	Cores	RAM	IOPS	Instance	EBS	Total
db.m5.4xlarge	8	64 GBs	4000	12 475 US\$	4 800 US\$	17 275 US\$
db.m5.2xlarge	4	32 GBs	2000	6 237 US\$	2 400 US\$	8 637 US\$

# DBtune v2

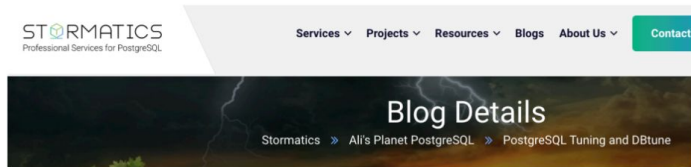


## New features

- 1. Always on agents**
- 2. Better user interface experience for DBAs**
- 3. Programmatic API for integrations**



# Some nice readings for DBtune



February 14, 2024 By Muhammad Ali Ali's Planet PostgreSQL, Blog

## PostgreSQL Tuning and DBTune

Parameter tuning in PostgreSQL involves the adjustment of various configuration settings inside `postgresql.conf` file which dictates how the database operates. These parameters affect many aspects of the database's operation which includes memory allocation, query planning, connection handling and disk I/O operations. Proper tuning ensures that PostgreSQL runs efficiently, making full use of the available hardware resources

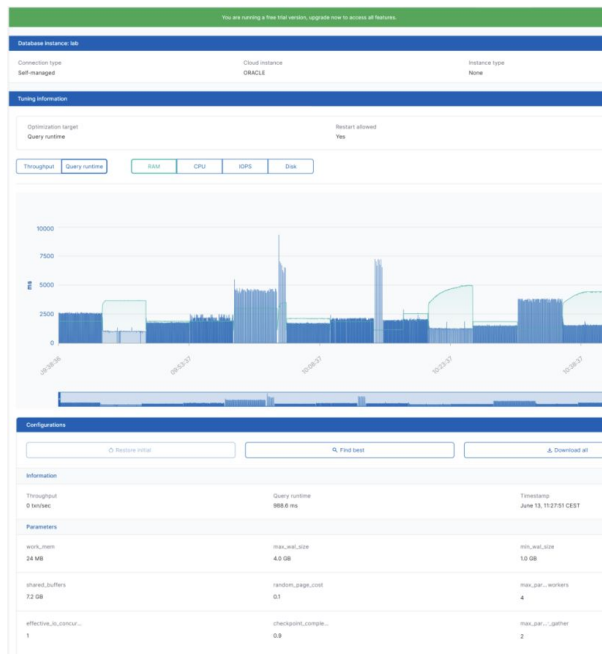
- Across all tests cases DBtune delivered improvement in performance up to 13.6x
- Compared to manual tuning DBtune achieved 2.2x speedup

Blog: <https://stormatics.tech/alis-planet-postgresql/postgresql-tuning-and-dbtune>

# Some nice readings for DBtune



## Independent technical analysis by Franck Pachot



- DBtune proposes to optimize parameters you wouldn't think of
- Using DBtune doesn't mean that the DBA goes on vacation — DBtune does the boring stuff
- Humans prefer stable configuration, AI is more aggressive

Blog: <https://www.linkedin.com/pulse/testing-dbtune-showing-postgresql-double-buffering-some-franck-pachot-voyhe>

# Thank you 🙏

Try DBtune for free ✨

# Auto-tuning RDS

## Stop over-provisioning



### Safe tuning in production environments

System guardrails to avoid unsafe configurations

- ✓ **Constrained optimization**  
Parameters have safe upper / lower limits in place
- ✓ **Memory monitoring guardrail**  
Real-time system memory monitoring to revert from potentially unsafe configurations  
E.g. configuration that uses too much RAM — Triggered at 90% of RAM
- ✓ **Early exit condition**  
Optimization space may result in configuration with worse performance than default  
This triggers early exit from existing configuration and move to next iteration

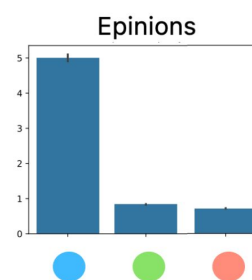
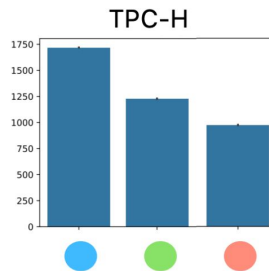
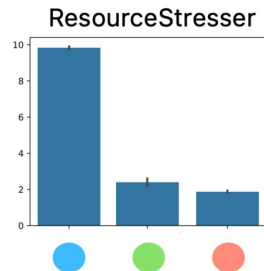
# Auto-tuning RDS

## Stop over-provisioning



Performance downside of non-restart (reload-only) strategy

Average query runtime



- Default PostgreSQL configuration & no autotuning
- *shared\_buffers*=25% & reload-only & autotuning
- Restarts allowed & autotuning