

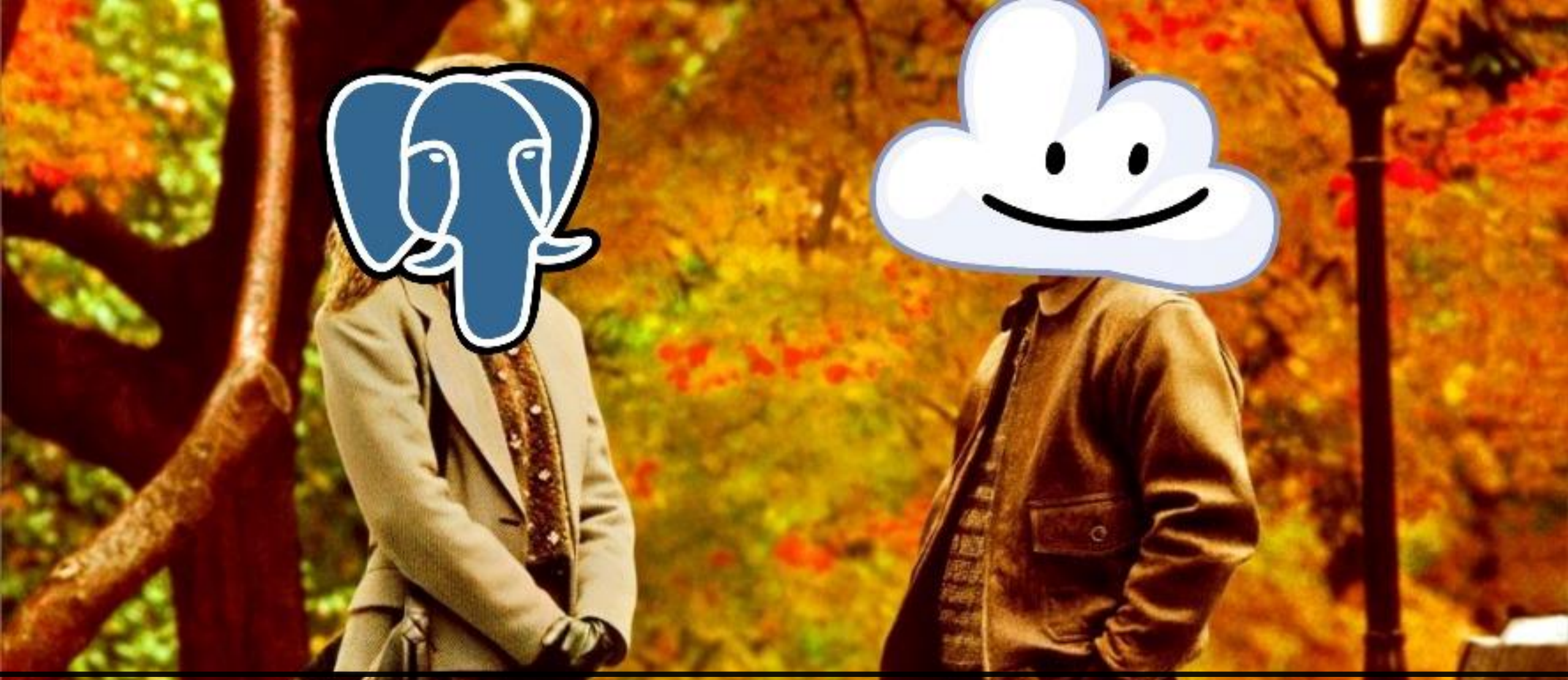


Mayur – DB Specialist@Veeam

When Autovacuum

Met FinOps: A

Cloud Romance

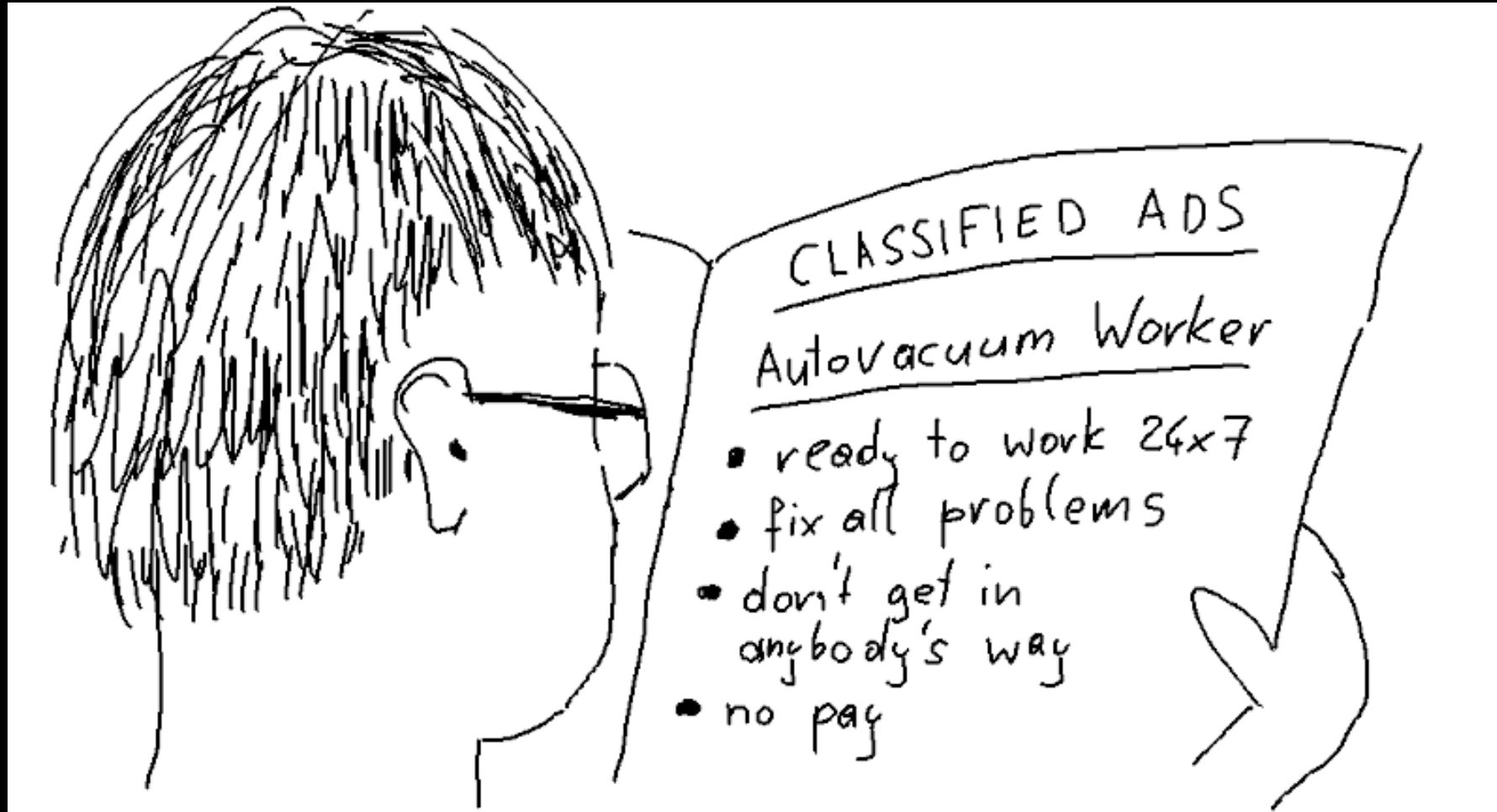


Romance? It only happens in the Movies.

Typical DBA/DEV Complaints

- "Autovacuum used to fly on bare-metal, but in the cloud, it feels like it's dragging forever!"
- "I've doubled the `max_autovacuum_workers`, yet dead tuples just keep stacking up"
- "Every time Autovacuum kicks in on that big table, our application queries start timing out!"

AutoVacuum : The most hard-working employee in your company.

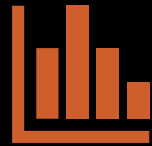


Opportunities you should pass by

AutoVacuum



Removes dead
tuples.



Updates planner
statistics.



Updates visibility
map.



Prevents txid
wraparound failures.

Important Parameters

- Autovacuum throughput is constrained by the `autovacuum_vacuum_cost_limit`, `autovacuum_vacuum_cost_delay` and further limited by host restrictions on Cloud.
- `autovacuum_vacuum_cost_limit` gets divided among number of workers specified by `autovacuum_max_workers`.
- `Autovacuum_work_mem` controls the amount of memory used by each worker.

How much io?

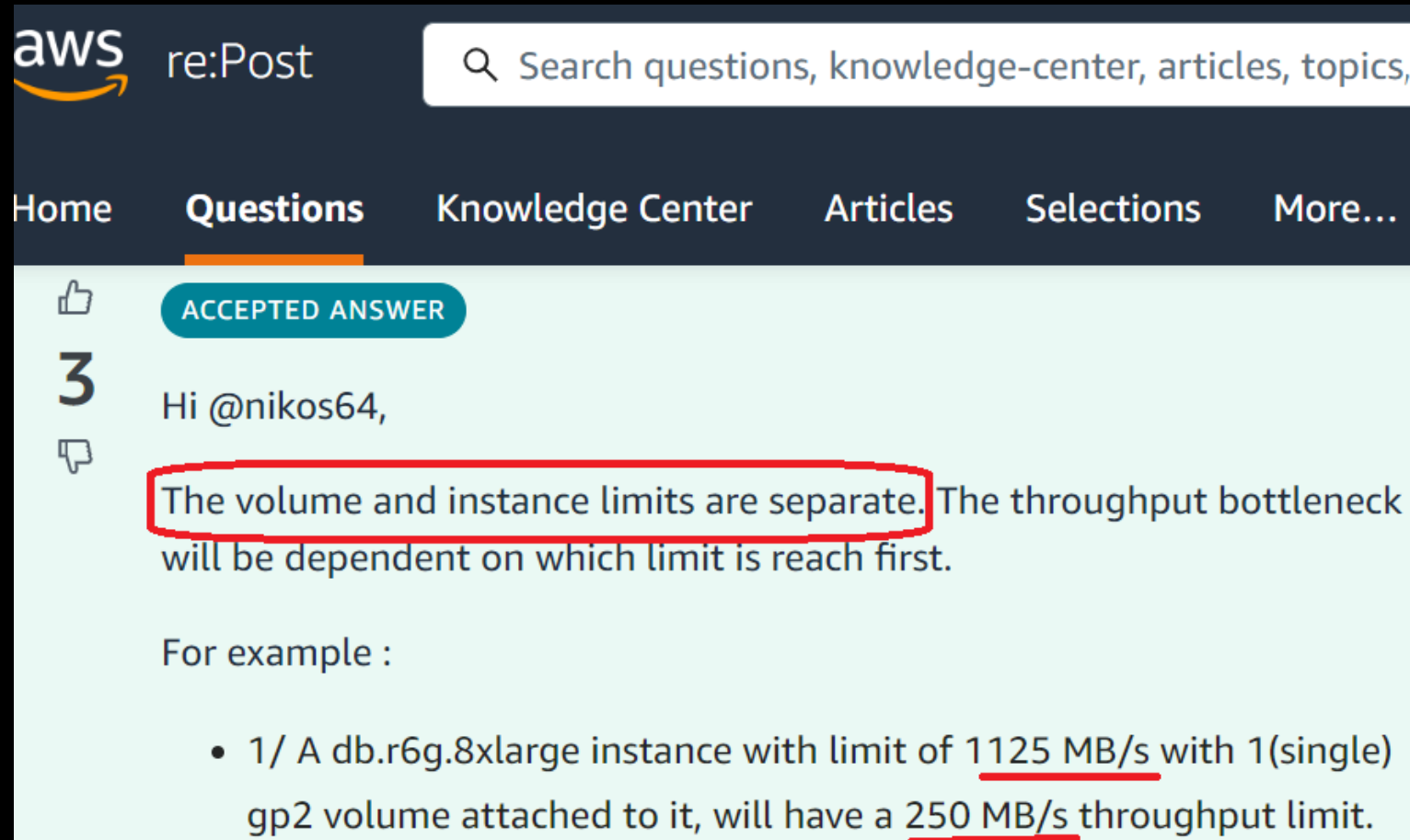
#At most, Autovacuum can do IO as shown below

Max Autovacuum throughput =
 $(1000/\text{autovacuum_vacuum_cost_delay}) * (\text{autovacuum_vacuum_cost_limit}/\text{vacuum_cost_page_hit}) * 8$
KB (default block_size)

#For PG17 default settings:

Max Autovacuum throughput = $(1000/2) * (200/1) * 8$ KB =
800MB per second

Instance and Storage both limit throughput



The screenshot shows the AWS re:Post interface. At the top left is the AWS logo and the text 're:Post'. To the right is a search bar with the placeholder text 'Search questions, knowledge-center, articles, topics, ...'. Below the search bar is a navigation menu with 'Home', 'Questions', 'Knowledge Center', 'Articles', 'Selections', and 'More...'. The 'Questions' tab is selected and highlighted with an orange underline. Below the navigation menu, there is a thumbs-up icon, a teal badge that says 'ACCEPTED ANSWER', and a large number '3'. To the right of the '3' is the text 'Hi @nikos64,'. Below this is a red-bordered box containing the text 'The volume and instance limits are separate.' followed by 'The throughput bottleneck will be dependent on which limit is reach first.' Below the red-bordered box is the text 'For example :'. At the bottom, there is a bulleted list with one item: '1/ A db.r6g.8xlarge instance with limit of 1125 MB/s with 1(single) gp2 volume attached to it, will have a 250 MB/s throughput limit.'

aws re:Post

Search questions, knowledge-center, articles, topics, ...

Home Questions Knowledge Center Articles Selections More...

Hi @nikos64,

The volume and instance limits are separate. The throughput bottleneck will be dependent on which limit is reach first.

For example :

- 1/ A db.r6g.8xlarge instance with limit of 1125 MB/s with 1(single) gp2 volume attached to it, will have a 250 MB/s throughput limit.

Instance throughput limit

Standard instance class : General purpose standard instances of latest graviton series offer better throughput upto 2.5GBps.
Gbps/Mbps here is in Bits hence division by 8 to get GB/s or MB/s.

Model	Core Count	vCPU*	Memory (GiB)	Storage	Dedicated EBS Bandwidth (Gbps)
db.m7g.large	-	2	8	EBS-Only	<u>Up to 10</u>
db.m7g.xlarge	-	4	16	EBS-Only	Up to 10
db.m7g.2xlarge	-	8	32	EBS-Only	Up to 10
db.m7g.4xlarge	-	16	64	EBS-Only	Up to 10
db.m7g.8xlarge	-	32	128	EBS-Only	10
db.m7g.12xlarge	-	48	192	EBS-Only	15
db.m7g.16xlarge	-	64	256	EBS-Only	20

Instance throughput limit

"Up to" is a very vague term. I found another piece of the puzzle in AWS document.

Instance type	Baseline / Maximum bandwidth (Mbps)	Baseline / Maximum throughput (MB/s, 128 KiB I/O)	Baseline / Maximum IOPS (16 KiB I/O)	NVMe	EBS optimization ²
m7g.medium ¹	315.00 / 10000.00	39.38 / 1250.00	2500.00 / 40000.00	✓	default
m7g.large ¹	630.00 / 10000.00	78.75 / 1250.00	3600.00 / 40000.00	✓	default
m7g.xlarge ¹	1250.00 / 10000.00	156.25 / 1250.00	6000.00 / 40000.00	✓	default
m7g.2xlarge ¹	2500.00 / 10000.00	312.50 / 1250.00	12000.00 / 40000.00	✓	default
m7g.4xlarge ¹	5000.00 /	625.00 / 1250.00	20000.00 /	✓	default

Note

¹ These instances can support maximum performance for 30 minutes at least once every 24 hours, after which they revert to their baseline performance. Other instances can sustain the maximum performance indefinitely. If your workload requires sustained maximum performance for longer than 30 minutes, use one of these instances.

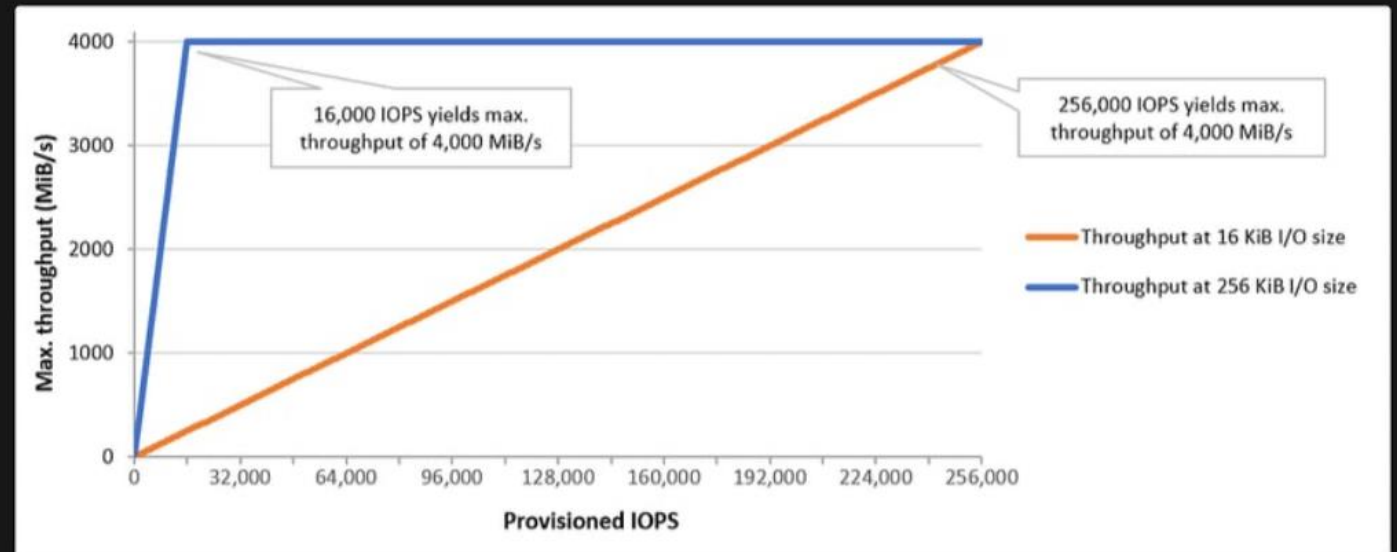
Since maximum performance is guaranteed for only 30 minutes per day, we will focus only on baseline values.

Storage throughput limit

io2 :

- Throughput scales proportionally up to 0.256 MiB/s per provisioned IOPS. Maximum throughput of 4,000 MiB/s can be achieved at 256,000 IOPS with a 16-KiB I/O size and 16,000 IOPS or higher with a 256-KiB I/O size. For DB instances not based on the AWS Nitro System, maximum throughput of 2,000 MiB/s can be achieved at 128,000 IOPS with a 16-KiB I/O size.

For io2 we can reach up to 500MB/s as we crank up iops knob.



Storage throughput limit



Instance class m7a.4xlarge (baseline is 625MB/s), Storage io2 256000 iops

1 instance (100 Utilized/Month) x 730 hours in a month = 984.0400 USD

24.04 USD

USD

Storage 100 GB in a month 1024 GB

Price 100 instances = 128.00 USD (Storage Cost)

128.00 USD x 1 instances = 25,600.00 USD (IOPS IO2 Cost)

128.00 USD

Storage 128.00 USD

holy IO

GP3 saves the day

Don't worry GP3 saves the day.

▼ Show calculations

Unit conversions

Storage amount: 1 TB x 1024 GB in a TB = 1024 GB

Pricing calculations

16,000 iops / 1,024 GB = 15.63 IOPS to GB ratio (gp3)

4,000 MiBps / 16,000 iops = 0.25 IOPS to Throughput ratio

1,024 GB per month x 0.115 USD x 1 instances = 117.76 USD (Storage Cost)

16,000 IOPS - 12000 free GP3 iops = 4,000 billable gp3 iops

4,000 MiBps - 500 MiBps free throughput = 3,500 MiBps billable throughput

4,000 IOPS x 0.02 USD = 80.00 USD

3,500 MiBps x 0.08 USD = 280.00 USD

117.76 USD + 80.00 USD + 280.00 USD = 477.76 USD

Storage pricing (monthly): 477.76 USD

Estimate summary [Info](#)

Upfront cost

0.00 USD

Monthly cost

1,461.80 USD

Total 12 months cost

17,541.60 USD

Includes upfront cost

Min. Config for 500MB/s throughput, 1TB db

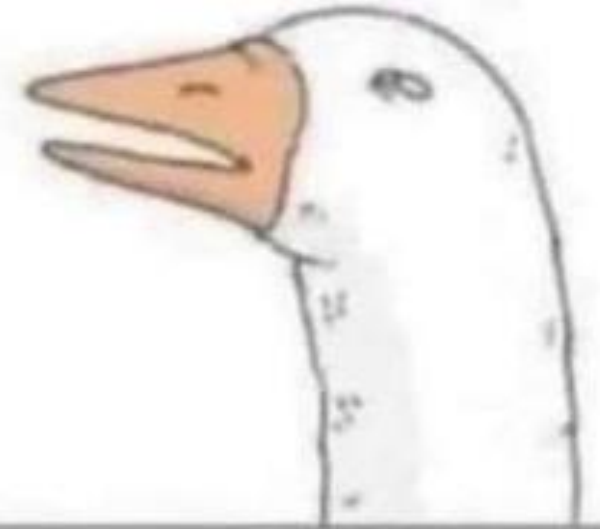
Cloud	Storage (all SSDs but taking only cost-efficient type)	Compute (Instance class)	Monthly Cost (in US-East)
AZURE (Azure Database for PostgreSQL — Flexible Server)	Premium SSD (5K iops for 500MB/s)	D16ds_v5	\$1407
AWS (Amazon RDS — Postgres)	GP3 (16K iops minimum for 500MB/s)	m7g.4xlarge	\$1462
GCP (Cloud SQL — Postgres)	Zonal extreme-pd	8 vCPUs	\$497

Why should you understand costs?



Why should
you
understand
costs?

**CLOUD
FIN-OPS EXPERT**



DBA

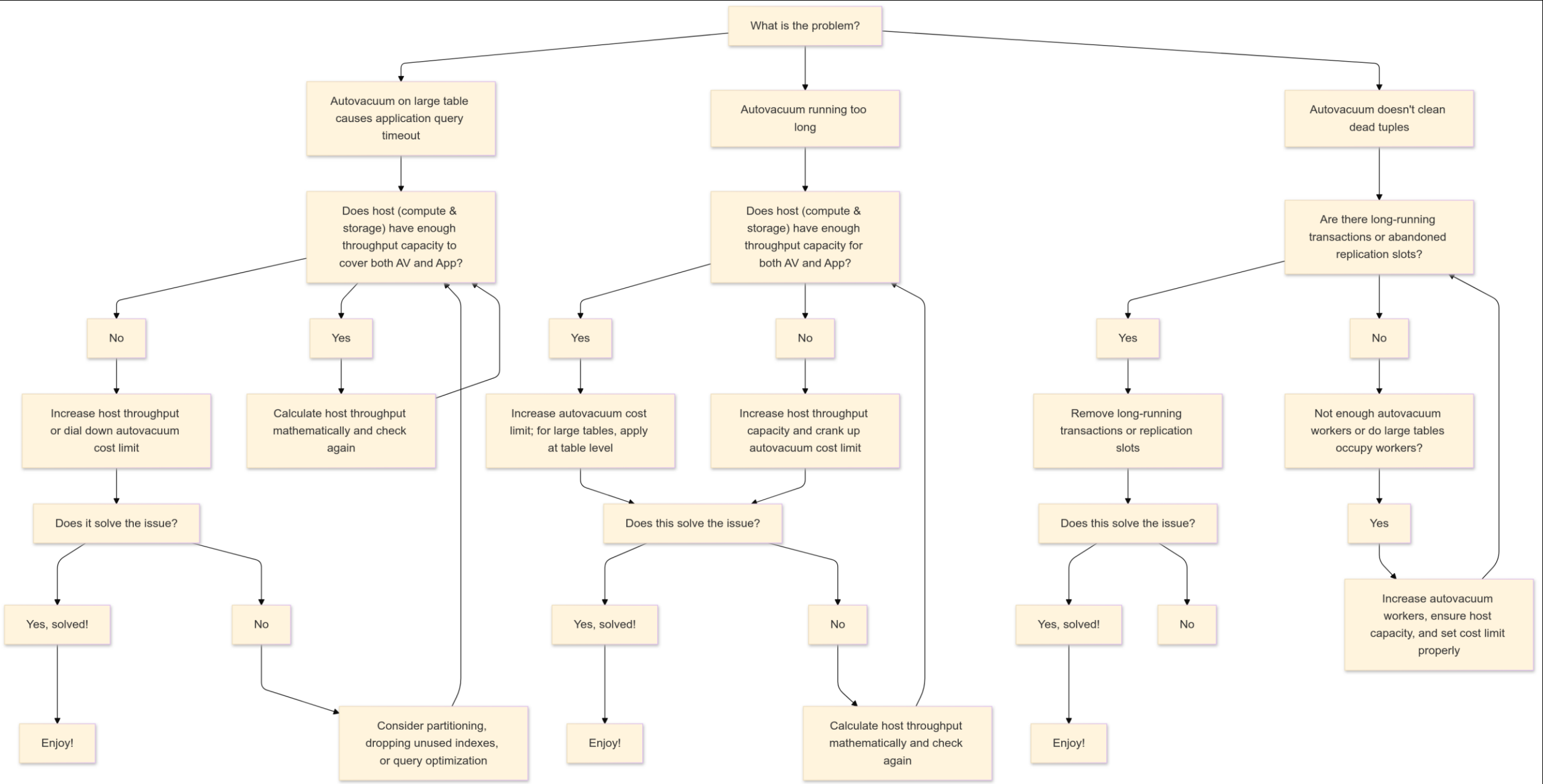
**WHY DID YOU PUT
DISKS ON IO2 DURING
CYBER BLACK FRIDAY AND
INCREASE OUR BILL BY \$23?**



Juggling Cost, Autovacuum Efficiency, and Application Performance



AV Tuning Flow : A picture is worth a thousand words



The Low Hanging Fruit

$$\text{autovacuum_work_mem} = \text{LEAST} \left(\frac{\text{Total Memory} - (\text{shared_buffers} + \text{session_memory} + \text{reserve_memory})}{\text{autovacuum_workers}}, 1 \text{ GB} \right)$$

Example:

Total Memory = 64 GB

shared_buffers = 16 GB (25% of Total Memory)

work_mem = 32 MB

active_sessions = 100

idle_sessions = 500 (assume poorly configured connection pool for more realistic calculations)

idle_session_memory = 10 MB per idle connection

autovacuum_workers = 6

session_memory = (active_sessions * work_mem * hash_mem_multiplier) + (idle_sessions * idle_session_memory)

Reserve OS Memory = Reserve 20% of Total Memory

autovacuum_work_mem = LEAST(3.9667GB, 1GB) = 1GB

Reason for autovacuum_work_mem 1GB restriction (Pre-PG17)

postgres / src / include / utils / memutils.h

Code

Blame

209 lines (179 loc) · 6.99 KB

```
38     * compute twice an allocation's size without overflow.
39     */
40     #define MaxAllocSize      ((Size) 0x3fffffff) /* 1 gigabyte - 1 */
41
```

Budget constraints = Think creatively

- Consider partitioning problematic tables, Size of data and indexes reduces.
- Detect and drop unused indexes (since PG vacuums all indexes).
- Minimize long-running transactions (lower wasteful vacuum runs).
- If it still impacts application performance, you may need to dial back cost limit and increase cost delay.
- Always implement an Early Warning System for TXID wraparound (AWS offers [a detailed guide](#) on this).
- Last but not the least, upgrade to PG17 for improved vacuuming.

Adapt to the nature of the beast.



- Thousands of DB's (in cloud) => Start with less aggressive settings for AV.
- Implement early warning system for txid wraparound exhaustion.
- Set `log_autovacuum_min_duration` and do log mining, create automated alert for tables appearing repeatedly.
- Increase cost limit gradually on need basis.
- If just a few PET DBs then take scientific approach of calculating optimal value for all AV settings.

A man wearing a brown baseball cap, a plaid shirt, and blue overalls stands in a field. His face is obscured by a large, blue, stylized elephant head with white outlines.

John Lennon was
a Postgres DBA

*As soon as you're born, they
make you feel small*

*By giving you no time instead of
it all*

*'Til the pain is so big you feel
nothing at all*

*A working class hero is
something to be*

*A working class hero is
something to be*

IT AIN'T MUCH BUT IT'S HONEST

References:

<https://www.percona.com/blog/tuning-autovacuum-in-postgresql-and-autovacuum-internals/>

<https://calculator.aws/#/>

<https://azure.microsoft.com/en-us/pricing/details/postgresql/flexible-server>

<https://cloud.google.com/sql/docs/postgres/pricing>



Thank you

DBA

AUTOVACUUM

Database Comedy Blog



Mayuresh B.

Database Specialist @ Veeam
Software | Databases, RDBMS, ...

