

Postgres Scaling Opportunities

BRUCE MOMJIAN



Configuring Postgres for heavy workloads can take many forms. This talk explores available Postgres scaling options.

<https://momjian.us/presentations>



Creative Commons Attribution License

Last updated: November 2025

Scaling

Database scaling is the ability to increase database throughput by utilizing additional resources such as I/O, memory, CPU, or additional computers.

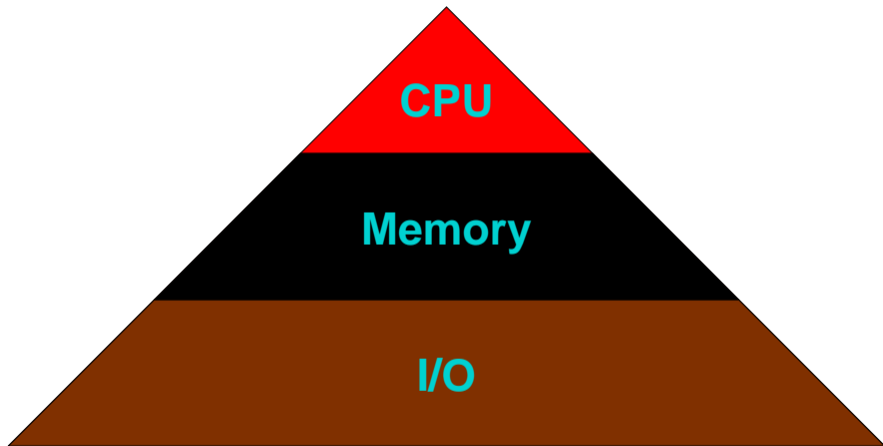
However, the high concurrency and write requirements of database servers make scaling a challenge. Sometimes scaling is only possible with multiple sessions, while other options require data model adjustments or server configuration changes.

Outline

Postgres scaling opportunities:

1. Scaling introduction
2. Multi-session
3. Single-session
4. Multi-host

1. Scaling Introduction: Vertical Scaling



https://momjian.us/main/presentations/performance.html#hw_selection

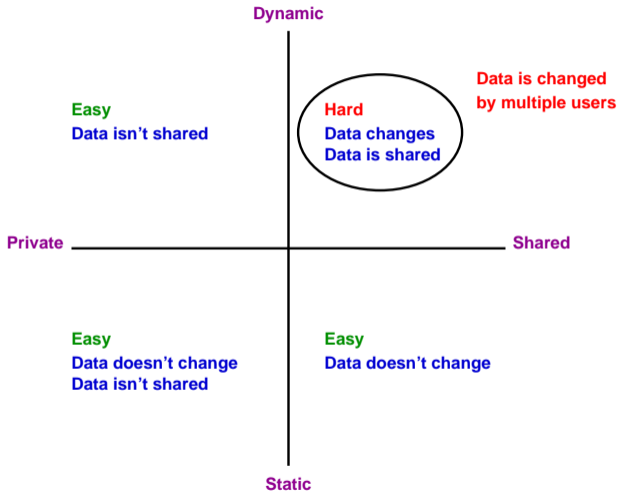
Examples

Vertical scaling examples

- More and faster CPUs
- More memory
- More and faster storage
- Power loss protection (https://momjian.us/main/blogs/pgblog/2023.html#October_23_2023)

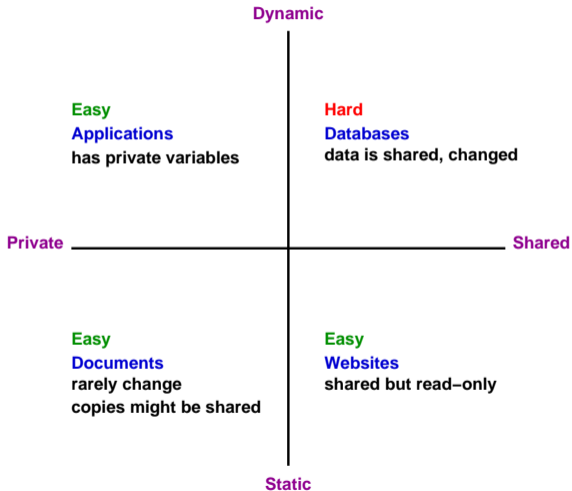
Horizontal scaling involves adding servers.

Horizontal Database Scaling: Why so Hard?

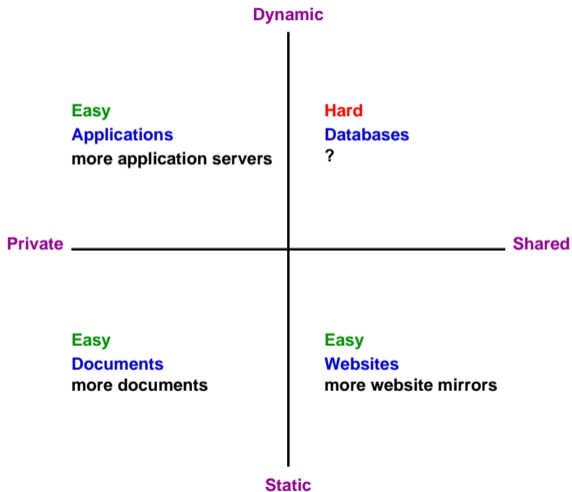


https://momjian.us/main/blogs/pgblog/2021.html#June_16_2021

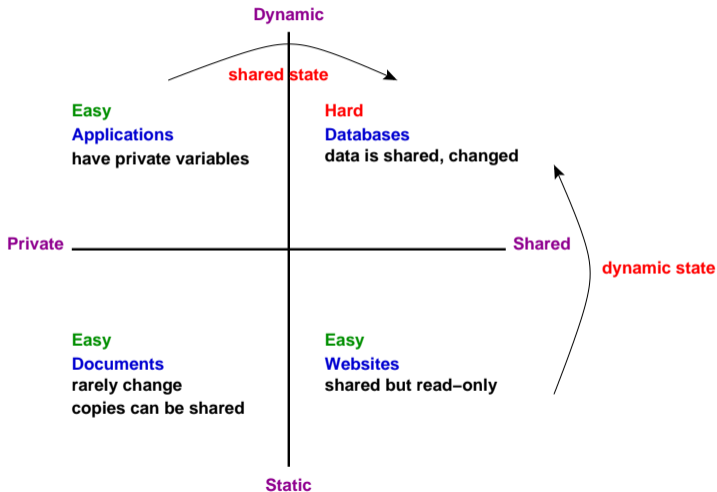
Example of Horizontal Scaling



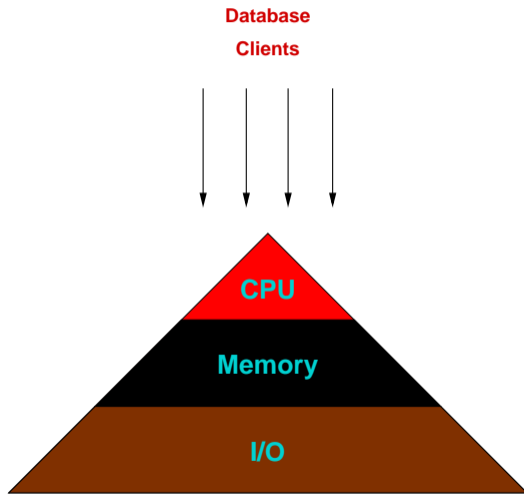
Easy Quadrants Can Be Copied



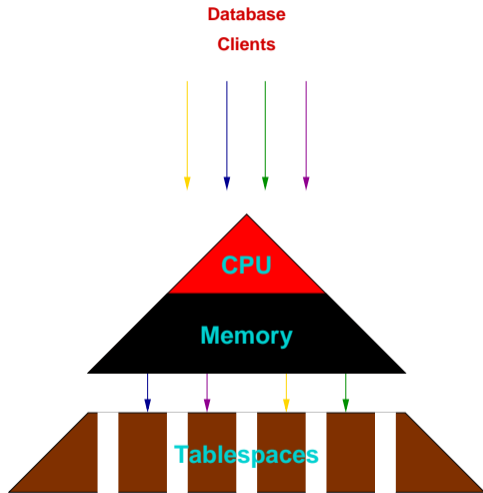
Easy Quadrants Use Databases



2. Multi-Session



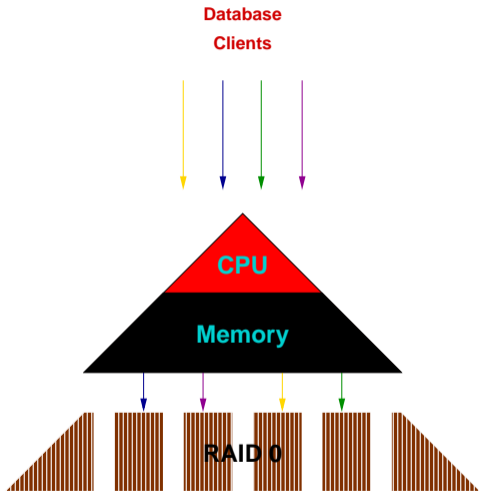
I/O Spreading Using Tablespaces



Requires tables & indexes to be spread across tablespaces

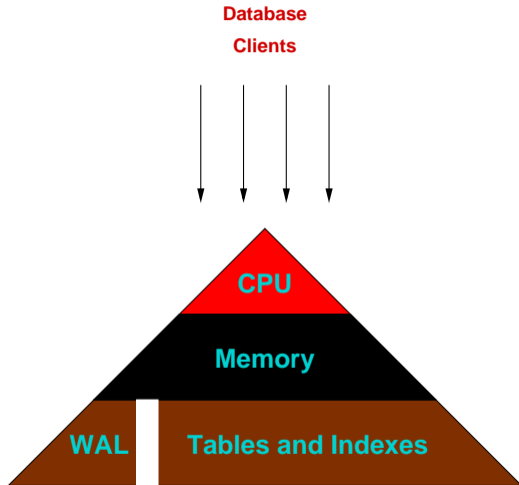
Tablespaces should be on different storage devices

I/O Spreading Using RAID 0



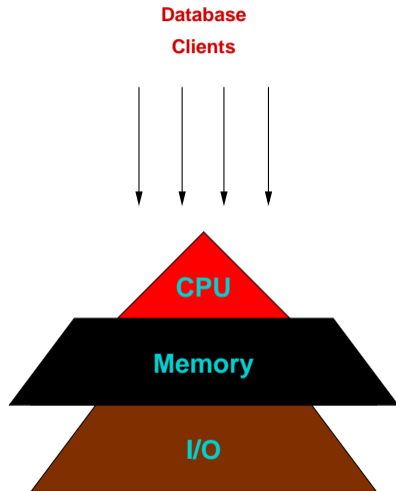
Auto-hashed by storage block number

Write Spreading Using WAL Relocation



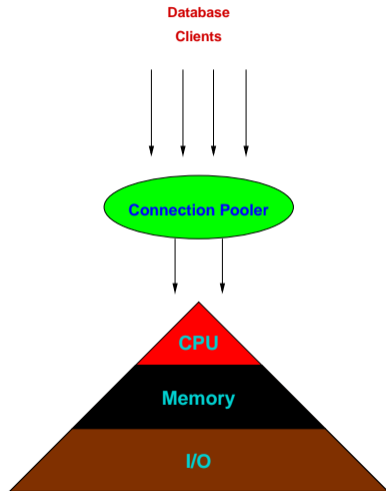
Separates WAL writes from table & index I/O

Read Reduction via Increased Memory



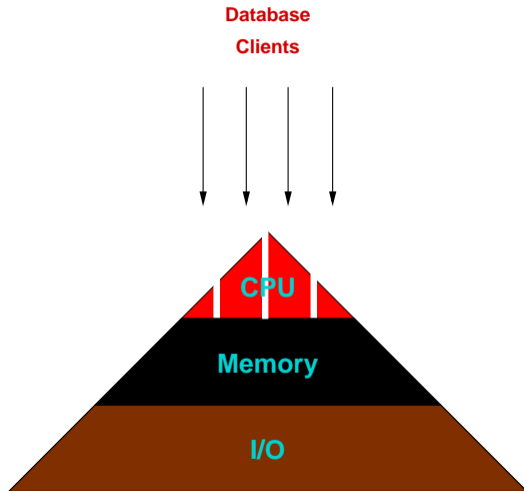
Additional memory caching reduces read requirements

Scaling Connections Using a Pooler



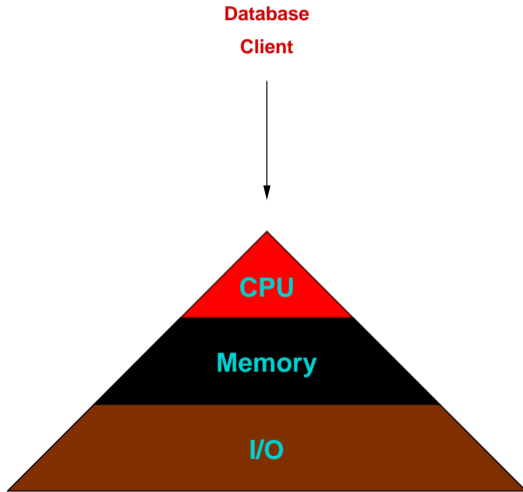
Fewer idle connections reduces resource usage

Multi-Session CPU Scaling

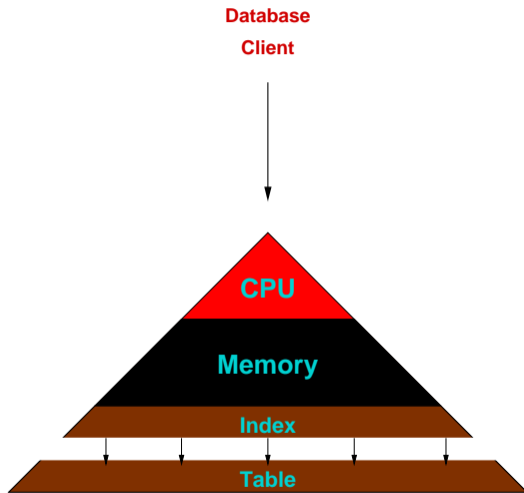


Multiple sessions spread across available CPUs

3. Single-Session



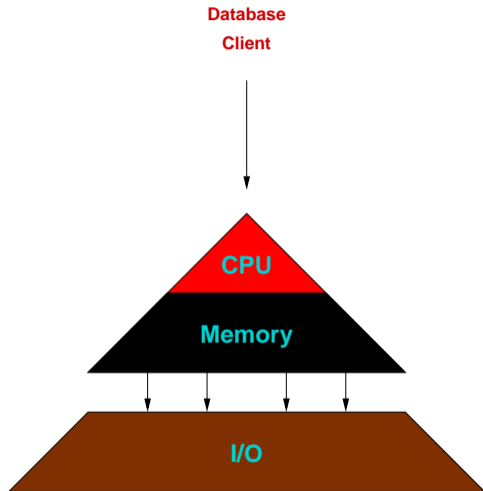
Read Parallelism Using effective and maintenance_io_concurrency



Used during bitmap heap scan

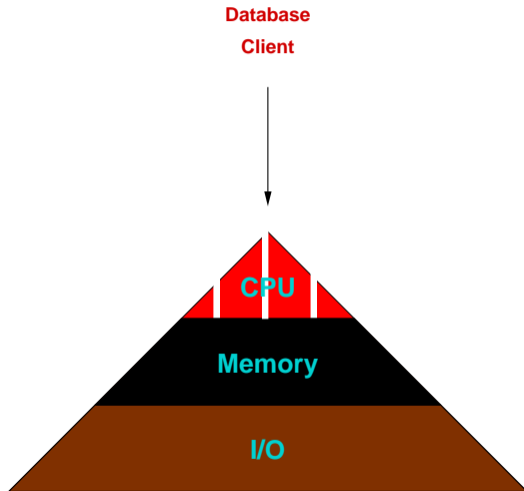
Assumes table is hashed across multiple devices

I/O Scaling via Parallelism



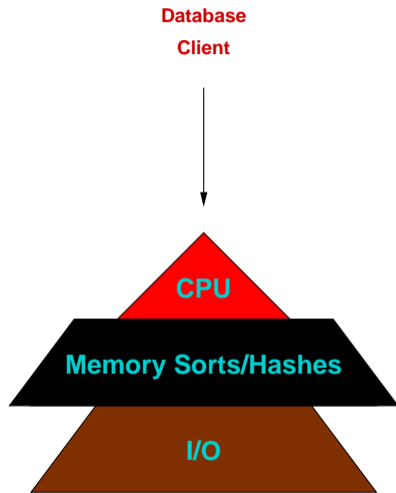
Involves parallel index, heap, partition, and tablespace access

CPU Scaling via Parallelism



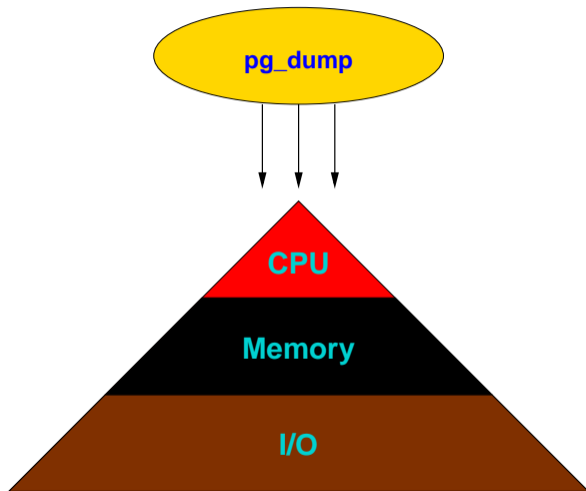
Involves parallel sorts, joins, and function execution

Sort I/O Reduction Using work_mem



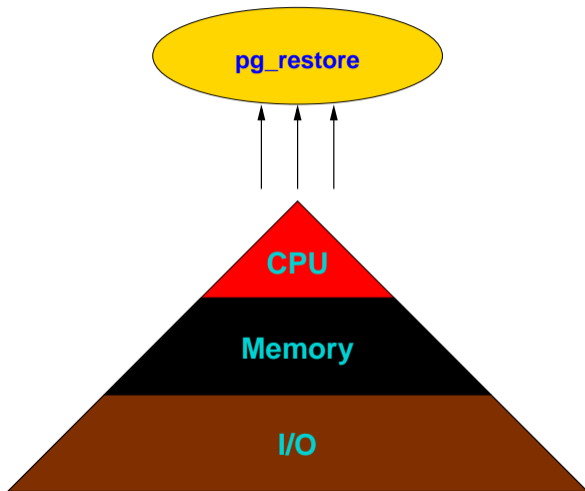
Reduces temporary result reads & writes

Logical Dump Parallelism



Dumps tables using concurrent database connections

Logical Restore Parallelism



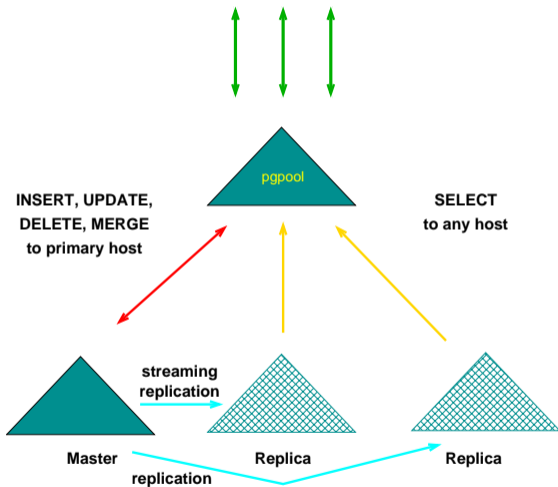
Loads tables and creates indexes using concurrent database connections

4. Multi-Host



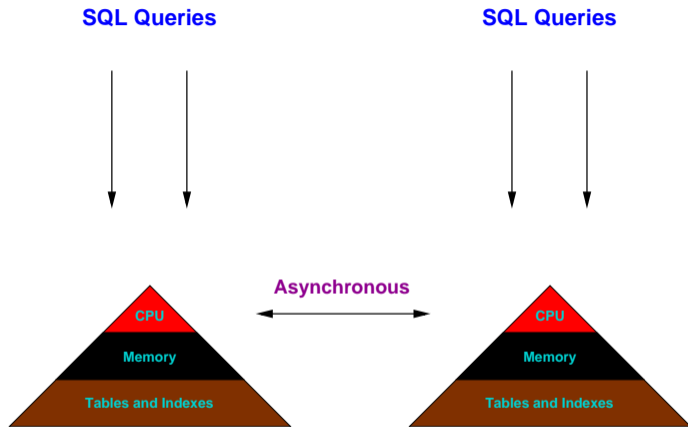
https://wiki.postgresql.org/wiki/PGECons_CR_Scaleout_2021

Read Scaling Using Pgpool & Streaming Replication



A full copy of the data exists on every node.

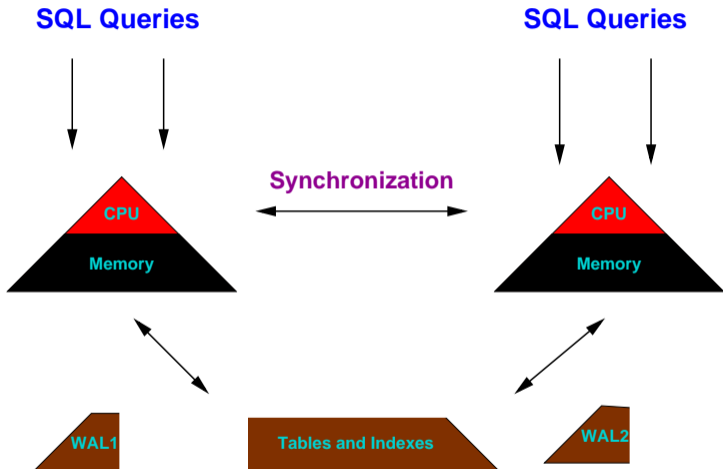
CPU/Memory Scaling With Asynchronous Multi-Master



A full copy of the data exists on every node; requires conflict resolution. The asynchronous delay allows write-load buffering.

https://momjian.us/main/blogs/pgblog/2018.html#December_24_2018

Oracle Real Application Clusters (RAC)

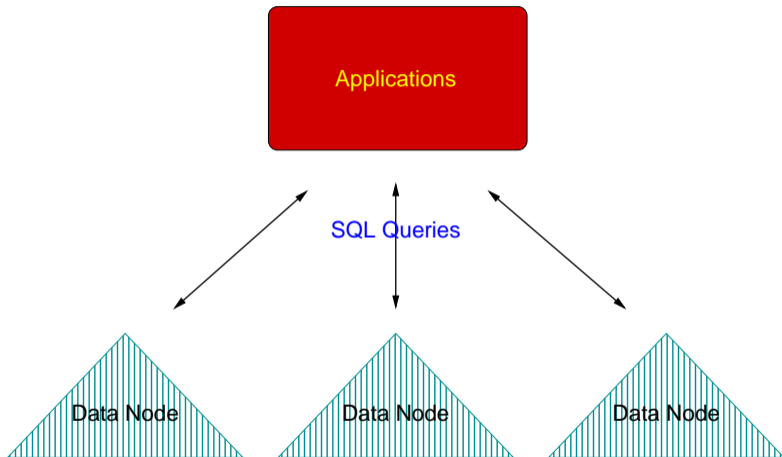


Tables and indexes on shared storage; inter-node synchronization required for cache consistency.

I/O Scaling with Sharding: Challenges

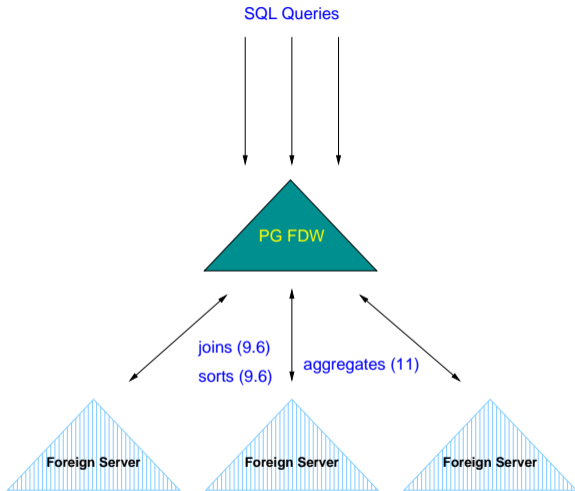
- Multi-host write queries require two-phase commit
- Multi-host visibility snapshots are not supported
- Sharding benefits are only possible with a shardable workload
- Changing the sharding layout can cause downtime
- Additional hosts reduce reliability; additional standby servers might be required

Application-Based Sharding



Applications send queries based on the sharding layout.

Scaling Using Foreign Data Wrappers



Requires rows to be hashed by key

Conclusion



<https://momjian.us/presentations>

<https://www.flickr.com/photos/87179607@N06/>