The Future of Postgres Sharding

BRUCE MOMJIAN

EDB

This presentation will cover the advantages of sharding and future Postgres sharding implementation requirements.

https://momjian.us/presentations

Last updated: June 2024

Outline

1. Scaling
2. Vertical scaling options
3. Non-sharding horizontal scaling
4. Existing sharding options
5. Built-in sharding accomplishments
6. Future sharding requirements

1. 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.
Postgres Scaling Opportunities https://momjian.us/main/presentations/preformance.html#scaling

2. Vertical Scaling

Vertical scaling can improve performance on a single server by:

- Increasing I/O with
  - faster storage
  - tablespaces on storage devices
  - striping (RAID 0) across storage devices
  - Moving WAL to separate storage
- Adding memory to reduce read I/O requirements
- Adding more and faster CPUs

3. Non-Sharding Horizontal Scaling

Non-sharding horizontal scaling options include:

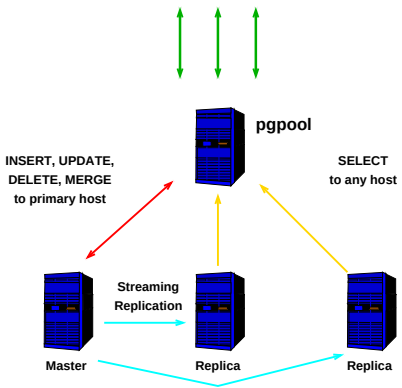
- Read scaling using Pgpool and streaming replication
- CPU/memory scaling with asynchronous multi-master

The entire data set is stored on each server.

http://en.wikipedia.org/wiki/Hype_cycle

Pgpool II With Streaming Replication

Streaming replication avoids the problem of non-deterministic queries producing different results on different hosts.



Why Use Sharding?

- Only sharding can reduce I/O, by splitting data across servers
- Sharding benefits are only possible with a shardable workload
- The shard key should be one that evenly spreads the data
- Changing the sharding layout can cause downtime
- Additional hosts reduce reliability; additional standby servers might be required

Typical Sharding Criteria

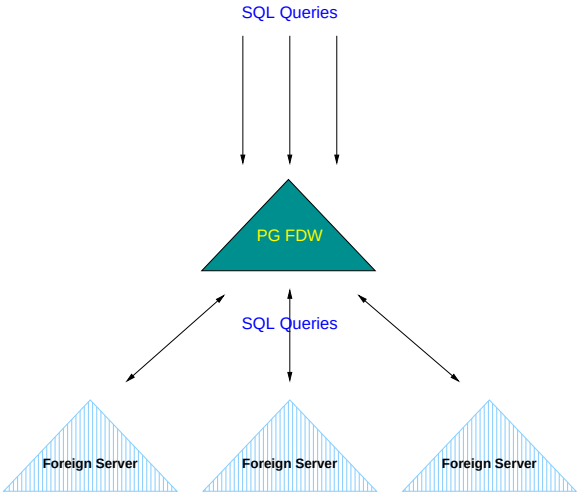
- List
- Range
- Hash

4. Existing Sharding Solutions

- Application-based sharding
- PL/Proxy
- Postgres-XC/XL
- Citus
- Hadoop

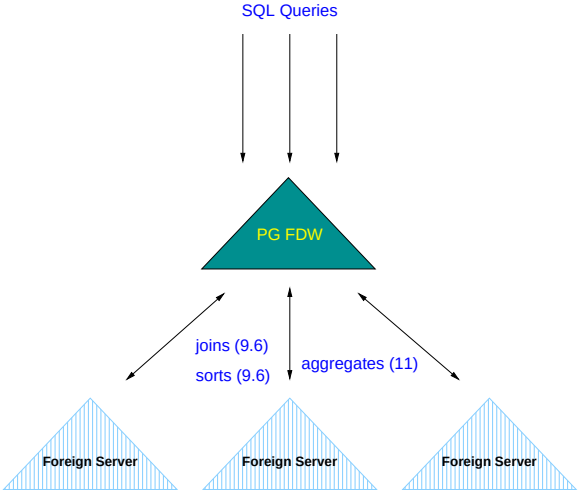
The data set is sharded (striped) across servers.

5. Built-in Sharding Accomplishments: Sharding Using Foreign Data Wrappers (FDW)



https://wiki.postgresql.org/wiki/Built-in_Sharding

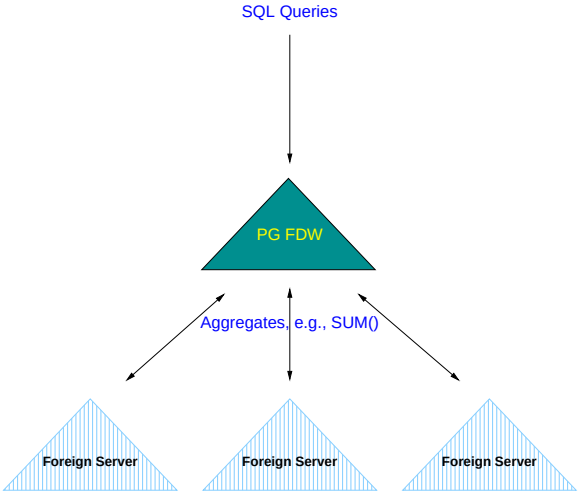
FDW Sort/Join/Aggregate Pushdown



Advantages of FDW Sort/Join/Aggregate Pushdown

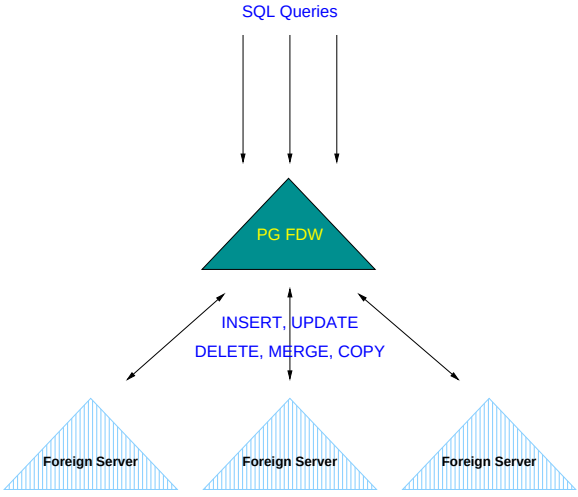
- Sort pushdown reduces CPU and memory overhead on the coordinator
- Join pushdown reduces coordinator join overhead, and reduces the number of rows transferred
- Aggregate pushdown causes summarized values to be passed back from the shards
- WHERE clause restrictions are also pushed down

Aggregate Pushdown in Postgres 11

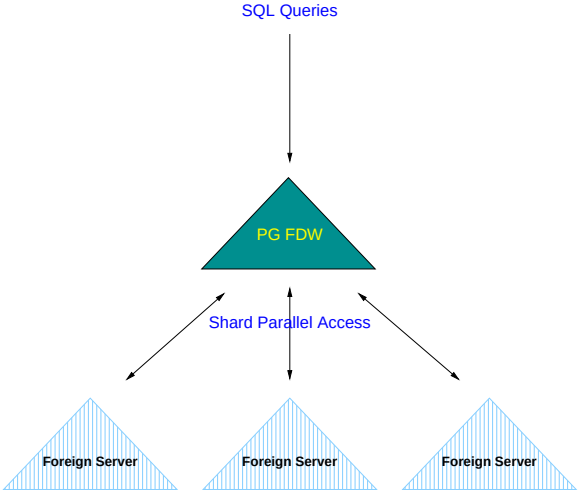


Unfortunately, aggregates are currently evaluated one partition at a time, i.e., serially.

FDW DML Pushdown in Postgres 9.6 & 11



Parallel Shard Access in Postgres 14

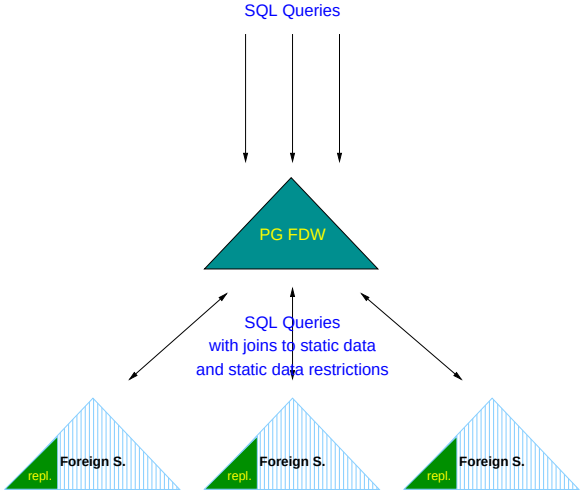


Advantages of Parallel Shard Access

- Ideal for queries that must run on every shard, e.g.,
  - restrictions on static tables
  - queries with no sharded-key reference
  - queries with multiple shared-key references
- Parallel aggregation across shards

https://www.youtube.com/watch?v=apvzyPwDN5g

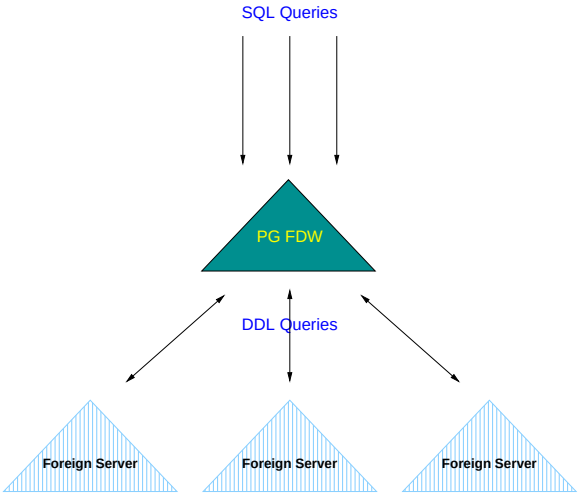
6. Future Sharding Requirements: Joins With Replicated Tables



Implementing Joins With Replicated Tables

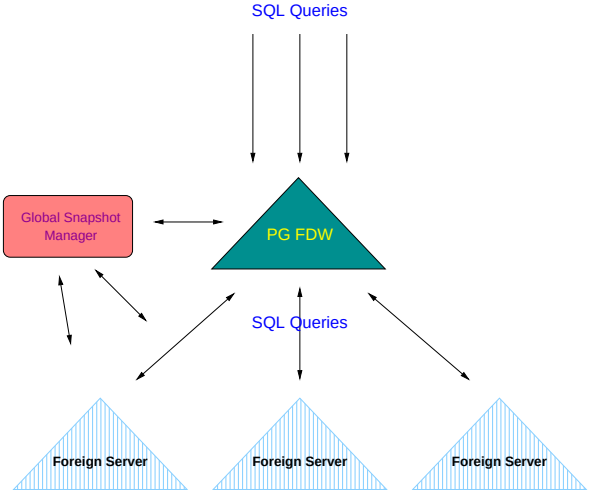
Joins with replicated tables allow join pushdown where the query restriction is on the replicated (lookup) table and not on the sharded column. Tables can be replicated to shards using logical replication. The optimizer must be able to adjust join pushdown based on which tables are replicated on the shards.

Shard Management



Shard management will be added to the existing partitioning syntax, which was added in Postgres 10.

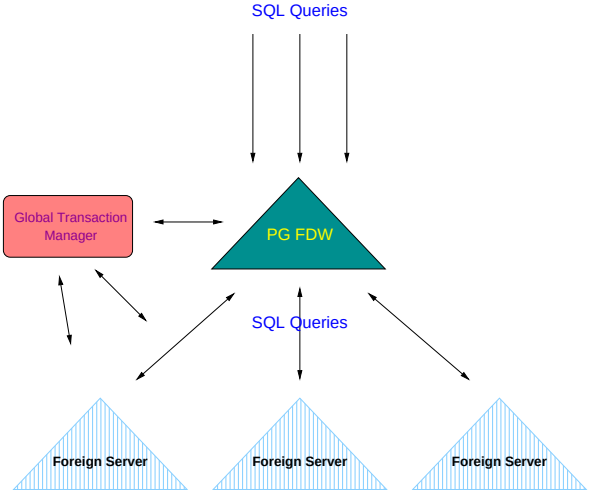
Global Snapshot Manager



Implementing a Global Snapshot Manager

- We already support sharing snapshots among clients with `pg_export_snapshot()`
- We already support exporting snapshots to other servers with the GUC `hot_standby_feedback`

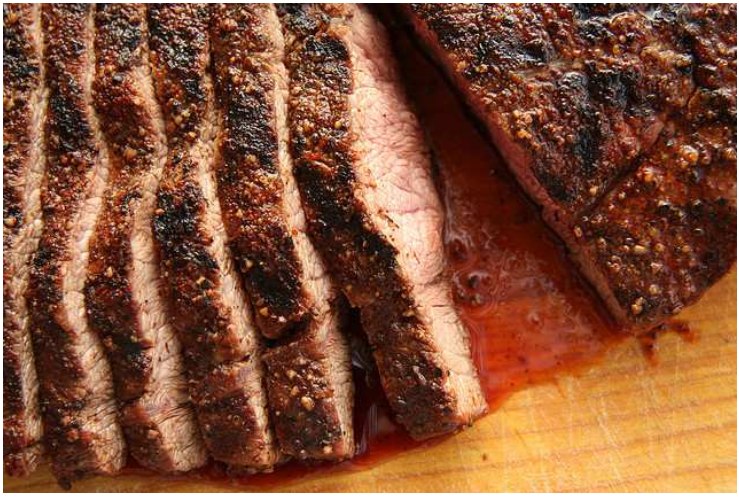
Global Transaction Manager



Implementing a Global Transaction Manager

- Can use prepared transactions (two-phase commit)
- Transaction manager can be internal or external
- Can use an industry-standard protocol like XA

Conclusion



https://momjian.us/presentations

https://www.flickr.com/photos/anotherpintplease/