

# Wisconsin Benchmark Pt 2: Benchmarking Postgres

Logan Ballard

05/15/2019

## 1. Background

Due to its ubiquity and my own familiarity with the system, I have chosen to benchmark postgres against itself. Postgres is a very flexible and well-supported database system which has a lot of community and enterprise backing behind it. It will behoove me to learn much more of its ins and outs if I wish to use it professionally, which seems very likely given how widespread its adoption is.

In addition to business reasons for choosing postgres, it is useful because it has a large amount of configurability. This ability allows for near-infinite tweaking of performance to maximize different attributes of the database. It also can be run locally without much setup, which allows for rapid iteration on various ideas. Through tools like *pgAdmin* and integration with familiar programming languages like *python3* with the *psycopg2* library, I was able to interface easily with the database and tune performance to my liking. This allows for more in-depth and meaningful exploration of the actual data and concepts without extended ramp-up to the system.

## 2. The System

I examined a few parameters that are involved in tuning postgres performance. They came in two flavors: Memory Parameters and Query Planner Parameters.

### (a) Memory Parameters

- i. *work\_mem* - change performance of hash joins and sorts
- ii. *temp\_buffers* - ????

### (b) Query Planning Parameters

- i. *enable\_hashjoin* - Will want to find queries where a hash join is preferable and attempt to show performance of them improving or degrading.
- ii. *enable\_mergejoin* - Will want to find queries where a merge join is preferable and attempt to show performance of them improving or degrading. Maybe create a hierarchy of join types?
- iii. *enable\_sort* - All sorting steps will be discouraged. I would expect this to degrade performance significantly in ORDER BY and GROUP BY as well as sort-merge join queries.