CS165 - Milestone 2

Lev Kruglyak

October 2022

1 Introduction

In this milestone, we improve the speed of our scans by adding support for batching operators together and distributing these across multiple cores. Some of the challenges we face are concurrency, avoiding locking and blocking other queries, and fully understanding the performance implications of our improvements to make sure that they don't slow down queries in some cases.

We also would like to investigate how much of a theoretical speedup is possible for queries as the number of cores increases, as well as the maximum possible number of concurrent queries before losing performance improvements.

2 Problems Tackled

Let's examine some of the core problems faced in the implementation of this milestone.

- **Batching:** We don't execute any commands between the batch operators, and when we send the batch execute operation, we execute all the operators concurrently. This involves figuring out optimizations for executing these operators dynamically.
- **Concurrency:** If we have multiple cores running queries, we need to synchronize things carefully to ensure that we have no race conditions in the code.
- **Vectorization:** To limit the memory footprint of our code, and to improve memory allocation performance, we should vectorize our operations, meaning queries are performed on small chunks of columns at a time.

3 Technical Description

Here we'll explain how we solved these various problems.

3.1 Batching:

We add an "operator queue" to the `ClientContext` struct, which we fill up with operators as we receive them following a batch start command. Once we get a batch execute command, we send the queue to a separate scheduler which decides how to split up the list into parallelizable components.

3.2 Concurrency:

In order to keep things scalable and concurrent, we create an initial thread pool of workers (we found twice the number of cores to be a reasonable size) and send vectorized batches to them. We will not use any locks, since no two threads will need to access the same data, and all the execution will be done from a single main thread which also performs all the parsing and client IO. This should provide a linear speedup in the number of cores as desired.

3.3 Vectorization:

We need some way of breaking up commands into constant size chunks in order to parallelize and to keep the footprint small. Since we are not fetching anything from disk at runtime, we can limit our vectorization size to some small number of RAM pages to avoid unnecessary data movement. (Say blocks of size 4096, but this can be experimented with) Then, we run our sequentially on this data.

4 Challenges

We still have not implemented this milestone yet, so many challenges will still appear for example the implementation specifics of the various above described problems. I predict that combining vectorization with the batch optimizer will be tricky, since I haven't fully thought this out yet.