Project1 StandaloneKV

In this project, you will build a standalone key/value storage gRPC service with the support of the column family. Standalone means only a single node, not a distributed system. Column family (it will abbreviate to CF below) is a term like key namespace, namely the values of the same key in different column families is not the same. You can simply regard multiple column families as separate mini databases. It's used to support the transaction model in the project4, you will know why TinyKV needs the support of CF then.

The service supports four basic operations: Put/Delete/Get/Scan. It maintains a simple database of key/value pairs. Keys and values are strings. `Put` replaces the value for a particular key for the specified CF in the database, `Delete` deletes the key's value for the specified CF, `Get` fetches the current value for a key for the specified CF, and `Scan` fetches the current value for a series of keys for the specified CF.

The project can be broken down into 2 steps, including:

1. Implement a standalone storage engine.
2. Implement raw key/value service handlers.

The Code

The `gRPC` server is initialized in `kv/main.go` and it contains a `tinykv.Server` which provides a `gRPC` service named `TinyKv`. It was defined by protocol-buffer in `proto/proto/tinykvpb.proto`, and the detail of rpc requests and responses are defined in `proto/proto/kvrpcpb.proto`.

Generally, you don't need to change the proto files because all necessary fields have been defined for you. But if you still need to change, you can modify the proto file and run `make proto` to update related generated go code in `proto/pkg/xxx/xxx.pb.go`.

In addition, `Server` depends on a `Storage`, an interface you need to implement for the standalone storage engine located in `kv/storage/standalone_storage/standalone_storage.go`. Once the interface `Storage` is implemented in `StandaloneStorage`, you could implement the raw key/value service for the `Server` with it.

Implement standalone storage engine

The first mission is implementing a wrapper of badger key/value API. The service of gRPC server depends on an `Storage` which is defined in `kv/storage/storage.go`. In this context, the standalone storage engine is just a wrapper of badger key/value API which is provided by two methods:

```
type Storage interface {
    // Other stuffs
    Write(ctx *kvrpcpb.Context, batch []Modify) error
    Reader(ctx *kvrpcpb.Context) (StorageReader, error)
}
```

`Write` should provide a way that applies a series of modifications to the inner state which is, in this situation, a badger instance.

`Reader` should return a `StorageReader` that supports key/value's point get and scan operations on a snapshot.

And you don't need to consider the `kvrpcpb.Context` now, it's used in the following projects.

> Hints:
>
> - You should use badger.Txn to implement the `Reader` function because the transaction handler provided by badger could provide a consistent snapshot of the keys and values.
> - Badger doesn't give support for column families. engine_util package (`kv/util/engine_util`) simulates column families by adding a prefix to keys. For example, a key `key` that belongs to a specific column family `cf` is stored as `${cf}_${key}`. It wraps `badger` to provide operations with CFs, and also offers many useful helper functions. So you should do all read/write operations through `engine_util` provided methods. Please read `util/engine_util/doc.go` to learn more.
> - TinyKV uses a fork of the original version of `badger` with some fix, so just use `github.com/Connor1996/badger` instead of `github.com/dgraph-io/badger`.
> - Don't forget to call `Discard()` for badger.Txn and close all iterators before discarding.

Implement service handlers

The final step of this project is to use the implemented storage engine to build raw key/value service handlers including RawGet/ RawScan/ RawPut/ RawDelete. The handler is already defined for you, you only need to fill up the implementation in `kv/server/raw_api.go`. Once done, remember to run `make project1` to pass the test suite.