SQLite DB Stored in a Redis Hash

In a recent post I explained how a relational database could be backed by a key-value store by virtue of B-Trees. This sounded great in theory, but I wanted to see that it actually works. And so last night I wrote a commit to Thredis, which does exactly that.

If you’re not familiar with Thredis - it’s something I hacked together last Christmas. Thredis started out as threaded Redis, but eventually evolved into SQLite + Redis. Thredis uses a separate file to save SQLite data. But with this patch it’s no longer necessary - a SQLite DB is entirely stored in a Redis Hash object.

A very neat side-effect of this little hack is that it lets a SQLite database be automatically replicated using Redis replication.

I was able to code this fairly easily because SQLite provides a very nice way of implementing a custom Virtual File System (VFS).

Granted this is only proof-of-concept and not anything you should dare use anywhere near production, it’s enough to get a little taste, so let’s start an empty Thredis instance and create a SQL table:

```
$ redis-cli
redis 127.0.0.1:6379> sql "create table test (a int, b text)"
(integer) 0
redis 127.0.0.1:6379> sql "insert into test values (1, 'hello')"
(integer) 1
redis 127.0.0.1:6379> sql "select * from test"
1) 1) 1) "a"
      2) "int"
   2) 1) "b"
      2) "text"
2) 1) (integer) 1
   2) "hello"
redis 127.0.0.1:6379>
```

Now let’s start a slave on a different port and fire up another redis-client to connect to it. (This means `slaveof` is set to localhost:6379 and `slave-read-only` is set to false, I won’t bore you with a paste of the config here).

```
$ redis-cli -p 6380
redis 127.0.0.1:6380> sql "select * from test"
1) 1) 1) "a"
      2) "int"
   2) 1) "b"
      2) "text"
2) 1) (integer) 1
   2) "hello"
redis 127.0.0.1:6380>
```

Here you go - the DB’s replicated!

You can also see what SQLite data looks like in Redis (not terribly exciting):

```
redis 127.0.0.1:6379> hlen _sql:redis_db
(integer) 2
redis 127.0.0.1:6379> hget _sql:redis_db 0
"SQLite format 3\x00 \x00\x01\x01\x00@  \x00\x00\x00\x02\x00\x00\x00\x02\x00\x00 ...
```

Another potential benefit to this approach is that with not too much more tinkering the database could be backed by Redis Cluster, which would give you a fully-functional horizontally-scalable clustered in-memory SQL database. Of course, only the *store* would be distributed, not the query *processing*. So this would be no match to Impala and the like which can process queries in a distributed fasion, but still, it’s pretty cool for some 300 lines of code, n’est-ce pas?

Posted by Gregory Trubetskoy May 29th, 2013