REDIS

KEY-VALUE STORE

JUSTIN MCBRIDE

BASICS

- “Data-structure server”, or key-value store
- Not a database replacement
- All keys are strings
- All [final] values are strings
- Dataset is loaded into memory
- Bindings in all popular languages

KEY-VALUE STORE USE CASES

- Because it’s memory-oriented, real-time data is the best use
- Session storage
- State database
- Statistics
- Caching

USERS

OPTIONAL FEATURES

- Persistence
- Replication
- Clustering
- Server-side scripting

KEY FACTS

- String keys, containing any information (even blank)
- Keys shouldn’t be too verbose, nor unreadably short
- Maximum key size is 512 MB
- object-type:id

VALUE FACTS

- All values are strings
- The string can contain any information (e.g. a JPEG encoded image)
- Each value can be at most 512 MB

STORABLE TYPES

- Strings
- Hashes
- Lists
- Sets (and sorted sets)
- Bitmaps
- Hyperloglogs

SOME SPECIAL COMMANDS

- Incr
- Decr
- Incrby
- Decrby

```
127.0.0.1:6379> set x "1"
OK
127.0.0.1:6379> get x
"1"
127.0.0.1:6379> incr x
(integer) 2
127.0.0.1:6379> get x
"2"
127.0.0.1:6379>
```

CLOUD PROVIDERS

- AWS
- Morpheus
- RedisToGo
- Redis Cloud
- ObjectRocket (by Rackspace)

SET COMMAND

- SET [key] [value]

```
127.0.0.1:6379> set x 1
OK
127.0.0.1:6379> mset y 2 z 3
OK
127.0.0.1:6379>
```

- MSET [key] [value] [key] [value] …

GET COMMAND

- GET [key] [value]

```
127.0.0.1:6379> get x
"1"
127.0.0.1:6379> mget y z
1) "2"
2) "3"
127.0.0.1:6379>
```

- MGET [key] [value] [key] [value] …

REDIS EXPIRES

> set key some-value

OK

> expire key 5

(integer) 1

> get key (immediately)

"some-value"

> get key (after some time)

(nil)

ADVANCED SET

- SET key value [EX seconds] [PX milliseconds] [NX|XX]

- EX – expires
- PX – expires
- NX – only set if the key does NOT exist
- XX – only set if the key DOES exist

DATA STRUCTURES

STRINGS

- The basic type
- Any information allowed
- Blank strings are valid values
- Values are type-checked and casted automatically

HASHES

- Essentially a dictionary
- Useful to represent objects

```
127.0.0.1:6379> hmset student:1000 firstname "justin" lastname "mcbride"
OK
127.0.0.1:6379> hget student:1000 firstname
"justin"
127.0.0.1:6379> hgetall student:1000
1) "firstname"
2) "justin"
3) "lastname"
4) "mcbride"
127.0.0.1:6379>
```

HASH COMMANDS

- HSET [key] [field] value
- HGET [key] [field]
- HDEL [key] [field] [field …]
- HEXISTS [key] [field]
- HVALS [key]

```
127.0.0.1:6379> hkeys student:1000
1) "firstname"
2) "lastname"
127.0.0.1:6379> hvals student:1000
1) "justin"
2) "mcbride"
127.0.0.1:6379>
```

LISTS

- Implemented via linked lists
- Adding to the head or tail is constant time
- When fast access to the middle is required, use Sorted Sets
- No length limits

LIST OPERATIONS

- Can left (front) or right (rear) push
- Can left or right pop
- Pushing operations return number of elements
- Retrieve a range of elements from the list
- Truncate the list

LIST COMMANDS

- LPUSH [key] [value] [value…]
- LPOP [key]
- RPUSH [key] [value] [value…]
- RPOP [key]
- LRANGE [key] [start] [end]
- LTRIM [key] [start] [end]

BLOCKING LIST OPS

- List operations can be blocking
- Useful for producer-consumer pattern
- BRPOP / BLPOP [key] [timeout]
- First-come, first-serve
- Returns ‘nil’ after timeout
- 0 timeout is infinite wait

SETS

- Unordered collection
- Elements can be returned in any order
- Unique, non-repeating elements
- Can check existence and membership
- Intersection / Union / Difference between sets

SET COMMANDS

- SADD [key] [value] [value…]
- SMEMBERS [key]
- SISMEMBER [key] [value]
- SINTER [key] [key] [key…]
- SPOP [key]
- SCARD [key]

SORTED SETS

- Mix between a set and a hash
- Unique, non-repeating elements
- Each element has a ‘score’ associated with it
- When looking up, score has priority, then lexicographical score

SORTED SET COMMANDS

- ZADD [key] [score] [member] [[score] [member] …]
- ZRANGE [key] [start] [stop]
- ZRANGEBYSCORE [score] [start] [stop]
- ZLEXCOUNT [key] [min] [max]

BITMAPS

- Not a datatype, but operations performed on strings
- Up to 2^32 bits (512 MB)
- Constant time setting/getting
- BITOP
  - AND
  - OR
  - XOR
- BITCOUNT
- BITOS

HYPERLOGLOGS

- Essentially a set that doesn’t contain items
- Maintains a count of unique elements “added” to it
- Constant memory (~12KB worst case)

REDIS FEATURES

PERSISTENCE

- Entire dataset in memory
- Two persistence methods:
  - Snapshotting
  - Append-only file (journal)
- Defaults to two second write-outs

REPLICATION

- Master-slave
- Slaves can also be masters
- Read-only slaves are generally recommended

CLUSTERING

- Currently in beta
- Automatic partioning of the key space
- Hot resharding
- Heartbeat and failure detection planned
- Slave to Master promotion

LUA SCRIPTING

- It’s possible, and easy.
- http://redis.io/commands/eval

NOTES

MEMORY USAGE

- Empty instance -- ~1MB
- 1 Million small keys -> string value pairs -- ~100 MB
- 1 Million keys -> Hash with 5 fields -- ~200 MB

PERFORMANCE

- Single-threaded
- RAM speed or network speed limiting
- 32 bit is recommended, as 64-bit uses more memory
- Recommended to pair with on-disk database

BACKGROUND

- Open source software
- Written in 79% C, 15% Tcl
- https://github.com/antirez/redis
- Sponsored by Pivotal Software
- Means REmote DIctionary Server

DOCUMENTATION

- Data Types: http://redis.io/topics/data-types-intro
- All Redis commands: http://redis.io/commands
- Full documentation: http://redis.io/documentation
- Redis Online Tutorial: http://try.redis.io/
- Language clients: http://redis.io/clients