Jason Brelloch and William Gimson

Overview

1. Introduction
2. History
3. Specifications
    a. Structure
    b. Communication
    c. Datatypes
4. Command Overview
5. Advanced Capabilities
6. Advantages
7. Disadvantages
8. Twitter Example using Redis

Introduction

Redis is an open source, BSD licensed, key-value store. It is often called a data structure server because it can store data as strings, lists, sets, sorted sets, and hashes.

Redis uses an "in-memory" dataset to maintain it's speed, and periodically dumps the dataset to disk to maintain persistence.

History

Redis was initially developed in 2009 by Salvatore Sanfilippo to improve the performance of a web analytics tool created by his startup company.

In March 2010 VMWare hired Salvatore to work full time on Redis. Since then, Redis has grown quite a bit in popularity and is used by many large tech companies.

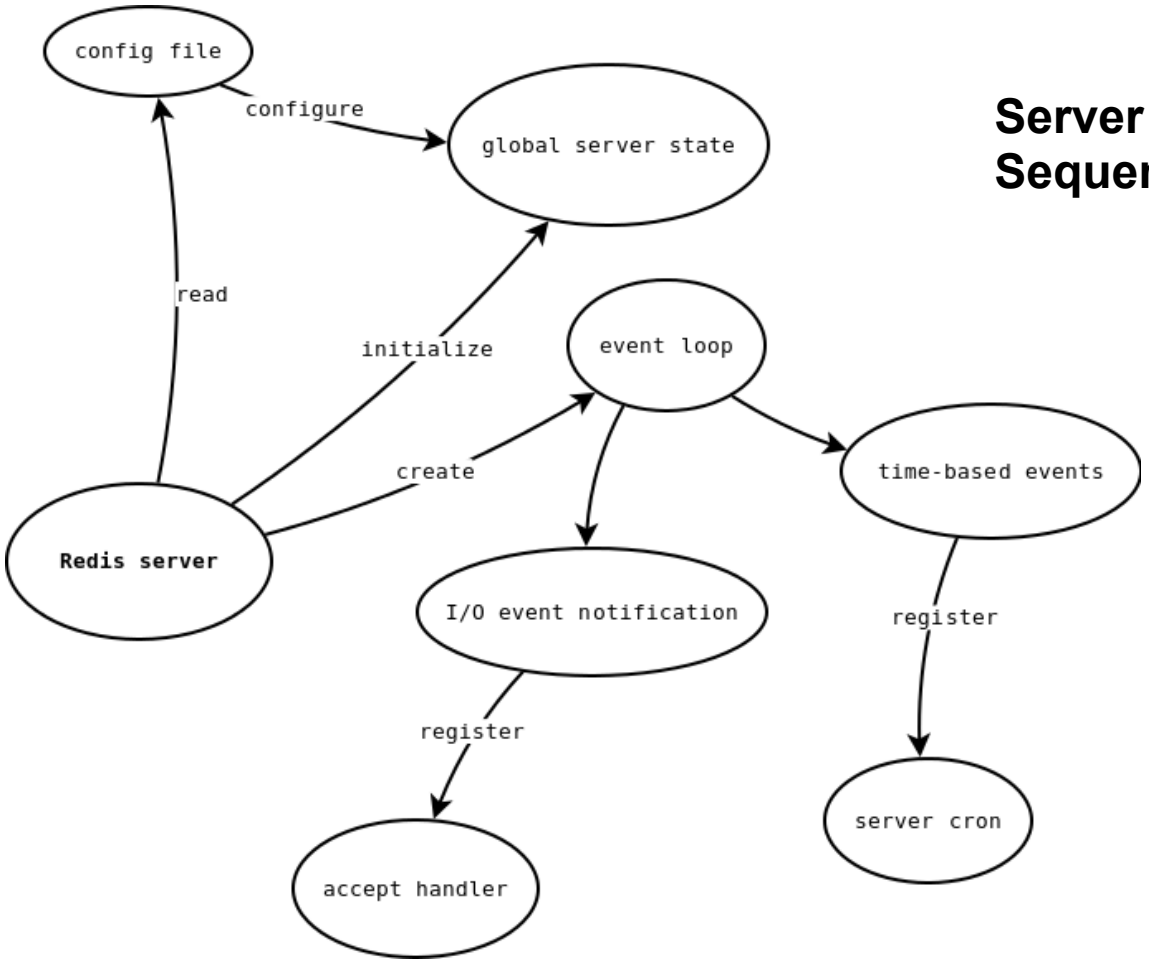
Users of Redis

craigslist

guardian.co.uk

digg

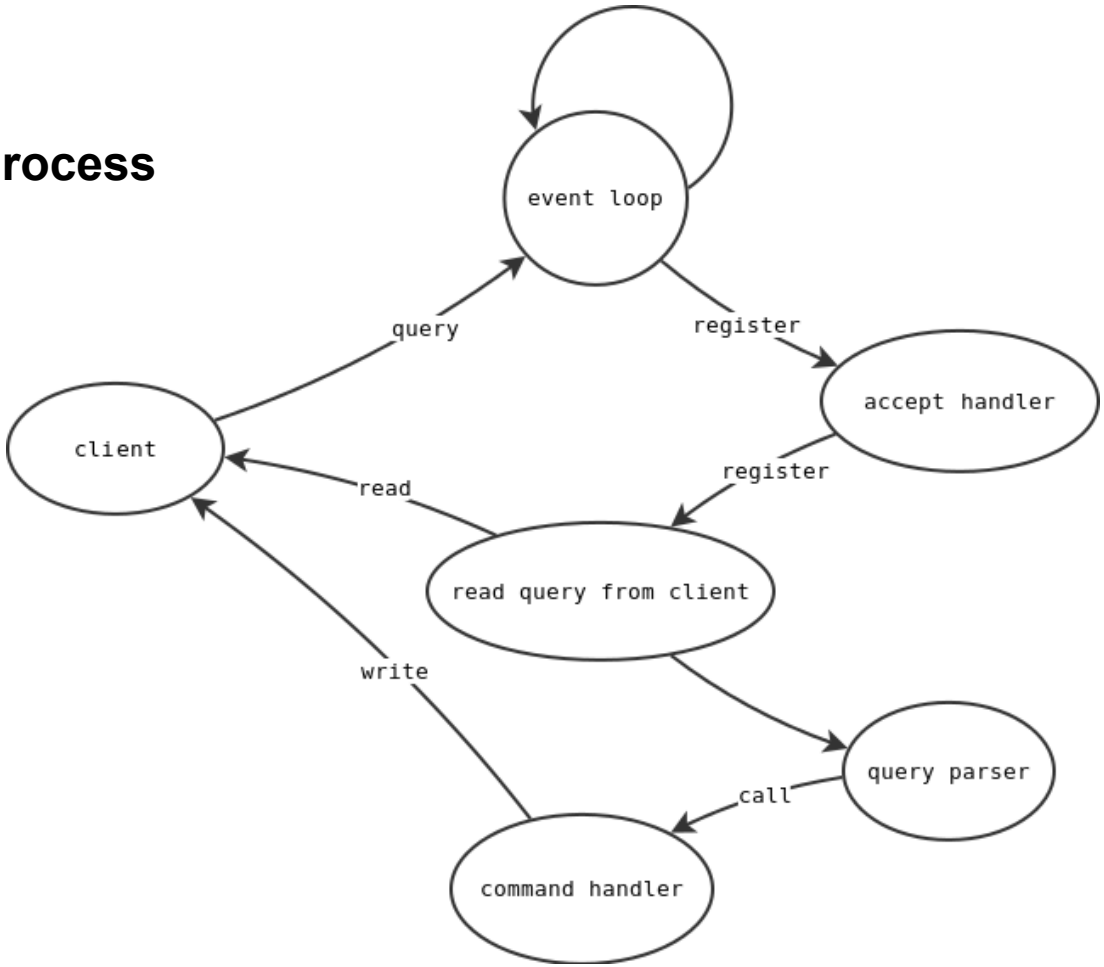
Structure



Server Startup Sequence

Structure

Query Process



Communication

Basic structure:

*<number of arguments> CR LF

$<number of bytes of argument 1> CR LF

<argument data> CR LF

...

$<number of bytes of argument N> CR LF

<argument data> CR LF

Example:

*3
$3
SET
$5
mykey
$7
myvalue

"*3\r\n$3\r\nSET\r\n$5\r\nmykey\r\n$7\r\nmyvalue\r\n"

Datatypes - Strings

Strings are the most basic type of Redis value, and are binary safe, meaning that any type of value, such as a JPEG image or serialized object, may be stored as a String

Strings in Redis are extremely flexible, and allow programmers to...

- Use Strings as atomic counters using commands in the INCR family: INCR, DECR, INCRBY.
- Append to strings with the APPEND command.
- Use Strings as a random access vectors with GETRANGE and SETRANGE.
- Encode a lot of data in little space, or create a Redis backed Bloom Filter using GETBIT and SETBIT.

In our application key/value pairs of Strings are used to register users as well as maintain data about users..

```
// Register the user
$userid = $r->incr("global:nextUserId");
$r->set("username:$username:id",$userid);
$r->set("uid:$userid:username",$username);
 $r->set("uid:$userid:password",$password);
```

Datatypes - Lists

Redis Lists are simply lists of Strings, sorted into insertion order.

New Strings can be pushed to the head or the tail of a List.

```
LPUSH mylist a   # now the list is "a"
LPUSH mylist b   # now the list is "b","a"
RPUSH mylist c   # now the list is "b","a","c" (RPUSH was used this time)
```

Insertion and deletion of elements of a List is O(n) in the middle, but approaches constant time at either end.

Fine grained control of list subsets is achieved through the LRANGE operator, as well as trimming with LTRIM

In our application we used Lists to manage user posts.

```
foreach($followers as $fid) {
        $r->lpush("uid:$fid:posts",$postId);
}
# Push the post on the timeline, and trim the timeline to the
# newest 1000 elements.
$r->lpush("global:timeline",$postId);
$r->ltrim("global:timeline",0,1000);
```

Datatypes - Lists

LRANGE Example -

```
redis>  RPUSH mylist "one"
(integer) 1
redis>  RPUSH mylist "two"
(integer) 2
redis>  RPUSH mylist "three"
(integer) 3
redis>  LRANGE mylist 0 0
1) "one"
redis>  LRANGE mylist -3 2
1) "one"
2) "two"
3) "three"
redis>  LRANGE mylist -100 100
1) "one"
2) "two"
3) "three"
redis>  LRANGE mylist 5 10
(empty list or set)
redis>
```

Datatypes - Sets

In Redis Sets are unordered collections of Strings

Addition, removal and testing for member existence are handled in constant time

Multiples of the same value, as in logical sets, are not allowed. This means that the check 'add if not already a member' is never required

Union, intersection and difference of sets are supported

In our application we use Sets and Set operations to maintain members and followers of members. This is the natural datatype for such operations since no two users may have the same user name.

```
// Add us to the set of followers for the uid we chose to follow
$r->sadd("uid:".$uid.":followers", $User['id']);
// Add the uid of the person we chose to follow to our set of followers
$r->sadd("uid:".$User['id'].":following", $uid);

// Remove us from the set of followers of this uid
$r->srem("uid:".$uid.":followers", $User['id']);
// Remove this uid from the set of people we follow
$r->srem("uid:".$User['id'].":following", $uid);
```

Datatypes - Sorted Sets

Exactly like regular Redis sets, except that every entry is associated with a 'score'

The set is ordered from smallest to largest score

While members are unique, set scores are not and may be repeated

Addition, removal and updating are very fast in a sorted set

Datatypes - Hashes

Redis Hashes are maps between string fields and string values, so they are the perfect data type to represent objects.

```
redis>  HMSET myhash field1 "Hello" field2 "World"
OK
redis>  HGET myhash field1
"Hello"
redis>  HGET myhash field2
"World"
redis>
```

Redis hashes can store 232-1 pairs and are extremely data efficient. This means a relatively small Redis database could potentially contain millions of individual objects.

Datatypes - Hashes

Benefits as reported by Instagram:

- Initially used basic key-value pair to store a media ID and a user ID (1 million pairs took about 70 MB)
- In order to convert to hashes the developers decided to make "buckets" of ID's. Each bucket or hash contained 1000 ID's. (1 million pairs was now contained in about 16 MB)
- Even with this reduction in size, search time maintained constant O(1)

http://instagram-engineering.tumblr.com/post/12202313862/storing-hundreds-of-millions-of-simple-key-value-pairs

Command Reference

Keys

DEL key [key ...]
DUMP key
EXISTS key
EXPIRE key seconds
EXPIREAT key timestamp
KEYS pattern
MIGRATE host port key destination-db
MOVE key db
OBJECT subcommand [arguments]
PERSIST key
PEXPIRE key milliseconds
PEXPIREAT key milliseconds-timestamp
PTTL key
RANDOMKEY
RENAME key newkey
RENAMENX key newkey
RESTORE key ttl serialized-value
SORT key [**BY** pattern]
TTL key
TYPE key

Strings

APPEND key value
BITCOUNT key [start] [end]
BITOP operation destkey key [key ...]
DECR key
DECRBY key decrement
GET key
GETBIT key offset
GETRANGE key start end
GETSET key value
INCR key
INCRBY key increment
INCRBYFLOAT key increment
MGET key [key ...]
MSET key value [key value ...]
MSETNX key value [key value ...]
PSETEX key milliseconds value
SET key value [**EX** seconds]
SETBIT key offset value
SETEX key seconds value
SETNX key value
SETRANGE key offset value
STRLEN key

Hashes

HDEL key field [field ...]
HEXISTS key field
HGET key field
HGETALL key
HINCRBY key field increment
HINCRBYFLOAT key field increment
HKEYS key
HLEN key
HMGET key field [field ...]
HMSET key field value [field value ...]
HSET key field value
HSETNX key field value
HVALS key

Command Reference

Lists

BLPOP key [key ...]
BRPOP key [key ...]
BRPOPLPUSH source destination timeout
LINDEX key index
LINSERT key **BEFORE|AFTER** pivot value
LLEN key
LPOP key
LPUSH key value [value ...]
LPUSHX key value
LRANGE key start stop
LREM key count value
LSET key index value
LTRIM key start stop
RPOP key
RPOPLPUSH source destination
RPUSH key value [value ...]
RPUSHX key value

Sets

SADD key member [member ...]
SCARD key
SDIFF key [key ...]
SDIFFSTORE destination key [key ...]
SINTER key [key ...]
SINTERSTORE destination key [key ...]
SISMEMBER key member
SMEMBERS key
SMOVE source destination member
SPOP key
SRANDMEMBER key [count]
SREM key member [member ...]
SUNION key [key ...]
SUNIONSTORE destination key [key ...]

Sorted Sets

ZADD key score member [score member ...]
ZCARD key
ZCOUNT key min max
ZINCRBY key increment member
ZINTERSTORE destination numkeys key [key ...] [**WEIGHTS** weight [weight ...]]
ZRANGE key start stop [**WITHSCORES**]
ZRANGEBYSCORE key min max [**WITHSCORES**] [**LIMIT** offset count]
ZRANK key member
ZREM key member [member ...]
ZREMRANGEBYRANK key start stop
ZREMRANGEBYSCORE key min max
ZREVRANGE key start stop [**WITHSCORES**]
ZREVRANGEBYSCORE key max min [**WITHSCORES**] [**LIMIT** offset count]
ZREVRANK key member
ZSCORE key member
ZUNIONSTORE destination numkeys key [key ...]

Advanced Capabilities

- Transactions
  - Redis transactions are groups of command that are batched together to be executed together (MULTI ... EXEC)
  - They have two very important guarantees.
    i. The commands will be executed in order and no other client will be able to execute commands in between
    ii. Either all commands will execute successfully or none at all will
  - There is also a WATCH command that will watch a variable and ensure that it remains unchanged until all operations have been successfully executed
    i. WATCH key1
    myVal = GET key1
    myVal = myVal + 1
    MULTI
    SET key1 myVal
    EXEC

Advanced Capabilities

- Pub/Sub
  - A built in messaging system for Redis
  - Users can subscribe to a "channel" to retrieve data that is published to that channel by other users
  - SUBSCRIBE channelName
    PUBLISH channelName thingToPublish
- Scripting
  - Redis supports execution of stored Lua scripts

Advantages

- Performance
- Atomicity
- Advanced Datatypes
- Support (documentation/clients)

Performance

- Redis can handle a large number of requests very quickly
- Benchmarks were performed using one Redis server and 50 simulated clients
- Results:
  - >100k successful sets/second
  - >80k successful gets/second

Atomicity

Redis' backend is designed using mostly single threaded programming. As a result all primitive commands are atomic in nature.

This make's application level code much easier to program because there is no need to worry about command order when it hits the server.

This also makes Redis' internals are much simpler and since it is open source, much easier to modify for individual situations.

Datatypes

Strings, Lists, and Sets are fundamental data types that can be used to store a wide variety of data models.

Also all strings in Redis are binary safe, meaning objects can be serialized and stored directly in the database.

Support

- Redis has clients in 28 different languages
- Due to Redis being both open-source and supported by VMWare, the documentation is quite extensive
- All primitive commands also have the time complexity in big O notation written in the documentation, which can be very helpful when planning algorithms and figuring out query performance

Disadvantages

- Memory
  - Partitioning
  - Hashes
  - Bit/Byte Operations
- Persistence
  - Replication
- Security

Memory

Redis is an "in-memory" dataset. This means that as the dataset expands it will take up more and more memory. Problems:

- Memory on servers is expensive
- 32 bit Redis implementations are limited to 4GB of total memory
- 64 bit Redis implementations have larger data sets because each key and value have to take up at least 64 bits - more wasted space

How do we solve these problems?

Partitioning

Partitioning splits the dataset across multiple Redis instances allowing the dataset to utilize the sum total of the memory, processing power, and network connections. The data can be split in several different ways.

- range partitioning - each key is mapped to a specific Redis instance
- hash partitioning - a hash function is used to generate a number for each key

It can also be accessed in several different ways.

- client side - the client determines where to send the data
- proxy assisted - a proxy server determines where to send the data
- query routing - each Redis instance takes any data and routes it to the correct instance

Other Optimizations

- Hashes
  - Data in hashes is stored much more efficiently than in regular groups of keys and values
  - Example - If there are several users, each with names/addresses/emails/etc., each user can be a hash containing all that data
- Bit/Byte level operations
  - Redis has support for querying individual bits and bytes of at a specific key
  - GETRANGE/SETRANGE can be used manipulate bytes of data, GETBIT/SETBIT are for bits
  - 100 million users gender data can be stored in 12MB of memory using this approach

Persistence

Reddis maintains persistence in two possible ways.

1. RDB persistence - Every so often Redis will spawn another process that will begin writing the database to disc. Once that write is complete, the newly written database will replace the old one on the disc.
2. AOF persistence - Redis logs every command that is done to the server, and then after a server restart it replays them to get to the same point

It is also possible to do a combination of the two.

Persistence - Problems

1. RDB persistence - Any operations done in between a write to disc and a crash/restart will be lost. Write operations can also be cpu intensive if there is a big dataset and the cpu power is lacking. The write operations also take up memory which can be a problem
2. AOF persistence - the AOF files tend to get pretty large, and can also slow down redis because the writing is not passed off to a new thread

There are some projects to combine the two into one unified persistence plan, but this is not supported by core Redis.

Replication

Replication is done by duplicating the database across multiple Redis instances. The instances are set up with a 1 Master, n Slaves relationship.

Advantages:

- A lot less memory and cpu required
- Data requests can be distributed to slaves (Read-only)
- If the Master instance goes down, a slave can automatically become the new Master

Disadvantages:

- Cost

Security

Redis is designed to be accessed by trusted clients in a trusted environment.

- generally not a good idea to expose it directly to the internet
- no SSL support
- no encryption
- only very basic authentication support (default turned off)

Redis Clients

Redis has clients for the following languages:

ActionScript	C	C#	C++	Clojure	Common Lisp	D
Dart	emacs lisp	Erlang	Fancy	Go	Haskell	haXe
Io	Java	Lua	Node.js	Objective-C	Perl	PHP
Pure Data	Python	Ruby	Scala	Scheme	Smalltalk	Tcl

Most languages have several clients. We will show you an example Redis project using the PHP client called Predis.

A Word About Predis

Predis is a popular PHP library for Redis, which makes development using these technologies together easier by providing 'syntactic sugar' for most Redis operations. An Example...

```
foreach($followers as $fid) {
        $r->lpush("uid:$fid:posts",$postId);
}
```

Twitter Example

We created an application inspired by twitter in which users can post and view messages, as well as follow other members, viewing their posts

The End

Questions?