Redis

"Little Server of Awesome"

2010 Dvir Volk

Software Architect, Do@
dvir@doat.com http://doat.com

What is redis

- Memcache-ish in-memory key/value store
- But it's also persistent!
- And it also has very cool value types:
  - lists
  - sets
  - sorted sets
  - hash tables (==object store!)

- Open source; very helpful and friendly community. Development is very active and responsive to requests.
- Sponsored by VMWare
- Used in the real world: github, craigslist, engineyard, ...
- Used heavily in do@ as a front-end database and for text classification.

Key Features and Cool Stuff

- All data is in memory (almost)
- All data is eventually persistent (But can be immediately persistent)
- Handles huge workloads easily:
  - ~150K reads/sec
  - ~150K writes/sec
  - O(1) behavior
- Ideal for write-heavy workloads
- Support for atomic operations
- Queries can be batched and executed atomically as transactions.
- Has pub/sub functionality
- Tons of client libraries for all major languages
- Single threaded, uses aync. IO

A little benchmark: 1M reads/ 1M writes

This is on my core i7 desktop PC:

- SET: 157903.05 requests per second
- GET: 154966.98 requests per second
- INCR: 144550.44 requests per second
- LPUSH: 156985.88 requests per second
- LPOP: 159085.44 requests per second
- SADD: 164018.03 requests per second
- SPOP: 161267.20 requests per second

So, how fast can an RDBMS push values to a list? :)

GREAT SUCCESS!

Scaling it up

- Master-slave replication out of the box
- Slaves can be made masters on the fly
- Currently does not support "real" clustered mode....
- ... But Redis-Cluster the next big feature in the works
- You can manually shard it client side
- Single threaded - run num_cores/2 instances on the same machine

Persistence

- All data is synchronized to disk - eventually or immediately
- Pick your risk level Vs. performance

- Data is either dumped in a forked process, or written as a append-only change-log (AOF)
- Append-only mode supports transactional disk writes so you can lose no data (cost: 99% speed loss)
- AOF files get huge, but redis can rebuild and minimize them.on the fly.
- You can save the state explicitly, background or blocking

- Default configuration:
  - Save after 900 sec (15 min) if at least 1 key changed
  - Save after 300 sec (5 min) if at least 10 keys changed
  - Save after 60 sec if at least 10000 keys changed

Virtual Memory

- If your database is too big - redis can handle swapping on its own.
- Keys remain in memory and least used values are swapped to disk.
- Swapping IO happens in separate threads
- Think about SSD drives!

Show me the features!

Now let's see the key featurs:

- Get/Set/Incr - strings/numbers
- Lists
- Sets
- Sorted Sets
- Hash Tables
- PubSub
- SORT

We'll use redis-cli for the examples.
Some of the output has been modified for readability.

The basics...

Get/Sets - nothing fancy. Keys are strings, anything goes but not spaces.

```
redis> SET foo "bar"
OK
redis> GET foo
"bar"
```

You can atomically increment numbers

```
redis> SET bar 337
OK
redis> INCRBY bar 1000
(integer) 1337
```

Getting multiple values at once

```
redis> MGET foo bar
1. "bar"
2. "1337"
```

Keys are lazily expired

```
redis> EXPIRE foo 1
(integer) 1
redis> GET foo
(nil)
```

Be careful with EXPIRE - re-setting a value without re-expiring it will remove the expiration!

Atomic Operations

GETSET puts a different value inside a key, retriving the old one

```
redis> SET foo bar
OK
redis> GETSET foo baz
"bar"
redis> GET foo
"baz"
```

SETNX sets a value only if it does not exist

```
redis> SETNX foo bar
*OK*
redis> SETNX foo baz
*FAILS*
```

SETNX + Timestamp => Named Locks! w00t!

```
redis> SETNX myLock <current_time>
OK
redis> SETNX myLock <new_time>
*FAILS*
```

Note that If the locking client crashes that might cause some problems, but it can be solved easily.

List operations

- Lists are basically arrays with random access. vector<*>
- You can push and pop at both sides, extract range, resize, etc.

```
redis> LPUSH foo bar
(integer) 1

redis> LPUSH foo baz
(integer) 2

redis> LRANGE foo 0 2
1. "baz"
2. "bar"

redis> LPOP foo
"baz"
```

- **BLPOP: Blocking POP - wait until a list has elements and pop them. Useful for realtime stuff.**

```
redis> BLPOP baz 10 [seconds]
..... We wait!
```

Set operations

- Sets are... well, sets of unique values w/ push, pop, etc.
- Sets can be intersected/diffed /union'ed server side.
- Can be useful as keys when building complex schemata.

```
redis> SADD foo bar
(integer) 1
redis> SADD foo baz
(integer) 1
redis> SMEMBERS foo
["baz", "bar"]

redis> SADD foo2 baz // << another set
(integer) 1
redis> SADD foo2 raz
(integer) 1

redis> SINTER foo foo2 // << only one common element
1. "baz"
redis> SUNION foo foo2 // << UNION
["raz", "bar", "baz"]
```

Sorted Sets

- Same as sets, but with score per element
- Ranked ranges, aggregation of scores on INTERSECT
- Can be used as ordered keys in complex schemata
- Think timestamps, inverted index, analytics...

```
redis> ZADD foo 1337 hax0r
(integer) 1
redis> ZADD foo 100 n00b
(integer) 1
redis> ZADD foo 500 luser
(integer) 1

redis> ZSCORE foo n00b
"100"

redis> ZINCRBY foo 2000 n00b
"2100"

redis> ZRANK foo n00b
(integer) 2
```

```
redis> ZRANGE foo 0 10
1. "luser"
2. "hax0r"
3. "n00b"

redis> ZREVRANGE foo 0 10
1. "n00b"
2. "hax0r"
3. "luser"
```

Hashes

- Hash tables as values
- Think of an object store with atomic access to object members

```
redis> HSET foo bar 1
(integer) 1
redis> HSET foo baz 2
(integer) 1
redis> HSET foo foo foo
(integer) 1

redis> HGETALL foo
{
  "bar": "1",
  "baz": "2",
  "foo": "foo"
}
```

```
redis> HINCRBY foo bar 1
(integer) 2

redis> HGET foo bar
"2"

redis> HKEYS foo
1. "bar"
2. "baz"
3. "foo"
```

PubSub - Publish/Subscribe

- Clients can subscribe to channels or patterns and receive notifications when messages are sent to channels.
- Subscribing is O(1), posting messages is O(n)
- Think chats, Comet applications: real-time analytics, twitter

```
redis> subscribe feed:joe feed:moe feed:
boe

//now we wait
....

1. "message"
2. "feed:joe"                <<<<<----------
3. "all your base are belong to me"
```

```
redis> publish feed:joe "all your base are
belong to me"
(integer) 1 //received by 1
```

SORT FTW!

- Key redis awesomeness
- Sort SETs or LISTS using external values, and join values in one go:

```
SORT key
SORT key BY pattern (e.g. sort userIds BY user:*->age)
SORT key BY pattern GET othervalue

SORT userIds BY user:*->age GET user:*->name
```

- ASC|DESC, LIMIT available, results can be stored, sorting can be numeric or alphabetic

- Keep in mind that it's blocking and redis is single threaded. Maybe put a slave aside if you have big SORTs

Example: *Very* Simple Social Feed

(using the python clien this time)

```
#let's add a couple of followers
>>> client.rpush('user:1:followers', 2)
>>> numFollowers = client.rpush('user:1:followers', 3)
>>> msgId = client.incr('messages:id') #ATOMIC FTW!

#add a message
>>> client.hmset('messages:%s' % msgId, {'text': 'hello world', 'user': 1})

#distribute to followers
>>> followers = client.lrange('user:1:followers', 0, numFollowers)

>>> pipe = client.pipeline()
>>> for f in followers:
    pipe.rpush('user:%s:feed' % f, msgId)
>>> pipe.execute()

>>> msgId = client.incr('messages:id') #increment id
#....repeat...repeat..repeat..repeat..
#now get user 2's feed
>>> client.sort(name = 'user:2:feed', get='messages:*->text')
['hello world', 'foo bar']
```

Other implementation ideas

- Real time analytics

use ZSET, SORT, INCR of values

- API Key and rate management

Very fast key lookup, rate control counters using INCR

- Real time game data

ZSETs for high scores, HASHES for online users, etc

- Database Shard Index

map key => database id. Count size with SETS

- Comet - no polling ajax

use BLPOP or pub/sub

- Inverted Index

Keep each word's occurrences in a ZSET, quick intersect

More resources

Redis' website:
http://code.google.com/p/redis/

Excellent and more detailed presentation by Simon Willison:
http://simonwillison.net/static/2010/redis-tutorial/

Much more complex twitter clone:
http://code.google.com/p/redis/wiki/TwitterAlikeExample

Full command reference:
http://code.google.com/p/redis/wiki/CommandReference

Redis based python ORM:
https://github.com/lsbardel/python-stdnet