REDIS

More than a KV store

Łukasz Dziedzia
@PyMalta 5.12.2017

HELLO!

My name is Łukasz Dziedzia

Software Development Manager @xcaliber_io

@ldziedzia

Why I talk about Redis?

- ✘ 90%+ developers I met use Redis
- ✘ Most of them uses Redis just as a caching backend
- ✘ Redis offers much more than that
- ✘ I got convinced to Redis on a “battlefield”
  - Redis is one of main components in XCaliber Gaming Platform
  - Responsible for many critical operations
  - 25k+ ops / sec
  - Minimal number of incidents

Goal of this presentation

- ✘ Introduce Redis
- ✘ Use cases (Gaming Platform context)
- ✘ Share lessons learned

Agenda

1. Redis overview
2. Data types
3. Programming with Redis
4. Administration (basics)
5. Lessons learned

Redis Overview

What actually is this Redis thing?

About Redis

REmote DIctionary Server – data structures store

Open source project https://github.com/antirez (BSD license)

Created by Salvatore Sanfilippo (@antirez)

Started in 2009

Key Features

- ✘ Blazingly fast
- ✘ KV store
- ✘ Rich data types (strings, hashes, lists, sets, geo, bitmaps and more)
- ✘ Transactions
- ✘ Built-in replication
- ✘ Different levels of persistence
- ✘ Messaging
- ✘ High availability
- ✘ Clusterization

How to use it?

```
$ wget http://download.redis.io/releases/redis-4.0.2.tar.gz
$ tar xzf redis-4.0.2.tar.gz
$ cd redis-4.0.2
$ make
$ src/redis-server
$ src/redis-cli
```

http://try.redis.io/
https://pypi.python.org/pypi/redis

How Fast is Redis?

Various benchmarks available, just to give an idea:

```
lukasz@XPLL017:~$ redis-benchmark -q -n 100000 -P 16
PING_INLINE: 1190476.25 requests per second
PING_BULK: 1666666.75 requests per second
SET: 943396.25 requests per second
GET: 1204819.38 requests per second
```

Intel(R) Core(TM) i5-5200U CPU @ 2.20GHz
4 cores, 16GB RAM

https://redis.io/topics/benchmarks

Why Redis is fast?

In-memory

Single threaded

Written in C

Redis Keyspace

16 databases (schemas) on one Redis server (see lessons learned)

- ✘ Manipulate keys: SET, GET, DEL, etc.
- ✘ Inspect keys with OBJECT and TYPE
- ✘ Expire keys with EXPIRE, EXPIREAT, PEXPIRE, etc.
- ✘ Navigate using: KEYS, SCAN, RANDOMKEY

Redis Data types

Strings, Hashes, Lists, Sets, Sorted Sets, Spatial, HyperLogLogs, Bitmaps

Strings

- ✘ Basic data type
- ✘ Binary safe
- ✘ Up to 512MB
- ✘ Support for operations on ints and floats (eg. INCRBY)

Use case: Cache. Set values under keys with TTL.

Strings – API example

Python

```
player_name = "Łukasz"
print(player_name)
player_age = 32
player_age += 1
len(player_name)
player_name[0:3]
```

Redis

```
SET player_name Łukasz
GET player_name
SET player_age 32
INCRBY player_age 1
STRLEN player_name
GETRANGE player_name 0 2
```

SETEX cache:key1 60 value

other commands: MSET, MGET, SETRANGE, INCR, DECR, INCRBYFLOAT, and more

Hashes

- ✘ Maps string keys to string values
- ✘ Good for representing “objects” (eg. user instance)
- ✘ “Small” hashes are stored very efficiently
- ✘ Up to 2^32-1 keys

Use case: storing player’s data

Hashes – API example

Python	Redis
player_1 = {"name": "Łukasz"}	HSET player:1 name Łukasz
player_1["name"]	HGET player:1 name
"name" in player_1	HEXISTS player:1 name
player_1.pop("name")	HDEL player:1 name
player_1.update(age=32, country="Poland")	HMSET player:1 age 32 country Poland
player_1.keys()	HKEYS player:1
player_1.values()	HVALS player:1
player.items()	HGETALL player:1

other commands: HINCRBY, HINCRBYFLOAT, HLEN, HSTRLEN, HSCAN, HSETNX, HMGET

Lists

- ✘ Linked list
- ✘ Fast append, slower lookup by index
- ✘ Sequences of strings stored in insertion order

Use case: store latest winners

Lists – API example

LPUSH winners '{"user": "user:1", "amount": 100}'

LTRIM winners 0 4 # 5 elements

...

LRANGE winners 0 –1
1) "{\"user\": \"user:7\", \"amount\": 700}"
2) "{\"user\": \"user:6\", \"amount\": 600}"
3) "{\"user\": \"user:5\", \"amount\": 500}"
4) "{\"user\": \"user:4\", \"amount\": 400}"
5) "{\"user\": \"user:3\", \"amount\": 300}"

LLEN winners
5

other commands: LINSERT, LREM, LPOP, RPOP, RPUSH, RPOPLPUSH, and more

Sets

- ✘ Unordered collections of strings
- ✘ Guarantees uniqueness
- ✘ Add, remove, check in O(1)

Use case: simple segmentation for fraud detection

Sets – API example

```
SADD players_deposited user:1 user:2 user:3
(integer) 3

SISMEMBER players_deposited user:2
(integer) 1

SCARD players_deposited
(integer) 3

SADD players_withdrew user:1 user:4
(integer) 2

SUNION players_deposited players_withdrew
1) "user:1"
2) "user:3"
3) "user:4"
4) "user:2"

SDIFFSTORE potential_bonus_abusers players_withdrew players_deposited
(integer) 1

SMEMBERS potential_bonus_abusers
1) "user:4"
```

Other commands:
SINTER, SDIFF, SINTERSTORE,
SUNIONSTORE, SMOVE, SPOP, SREM

Sorted Sets

- ✘ Ranked collections of strings
- ✘ Every item has an assigned float score
- ✘ Ordering happens on insertion, not request
- ✘ Like sets, they guarantee uniqueness

Use case: Leaderboard based on winnings.

Sorted Sets – API example

```
ZADD top_winners INCR 100 user:1
"100"

ZADD top_winners INCR 200 user:2
"200"

ZADD top_winners INCR 300 user:3
"300"

ZREVRANGE top_winners 0 -1 WITHSCORES
1) "user:3"
2) "300"
3) "user:2"
4) "200"
5) "user:1"
6) "100"

ZADD top_winners INCR 500 user:1
"600"

ZREVRANK top_winners user:1
(integer) 0
```

Other commands:
ZRANK, ZINCR, ZRANGE, ZREM, ZCARD, ZSCORE, ZSCAN, ZRANGEBYLEX, ZINTERSTORE, ZUNIONSTORE, ZDIFFSTORE, and more...

HyperLogLog

- ✘ Probabilistic data structure
- ✘ Estimates unique elements count
- ✘ Uses up to 12KB memory
- ✘ < 1% error

Use case: store estimated total number of all game spins

HyperLogLog – API example

PFADD all_spins spin:1

(integer) 1

PFADD all_spins spin:2

(integer) 1

PFCOUNT all_spins

(integer) 2

PFADD all_spins spin:1

(integer) 0

PFCOUNT all_spins

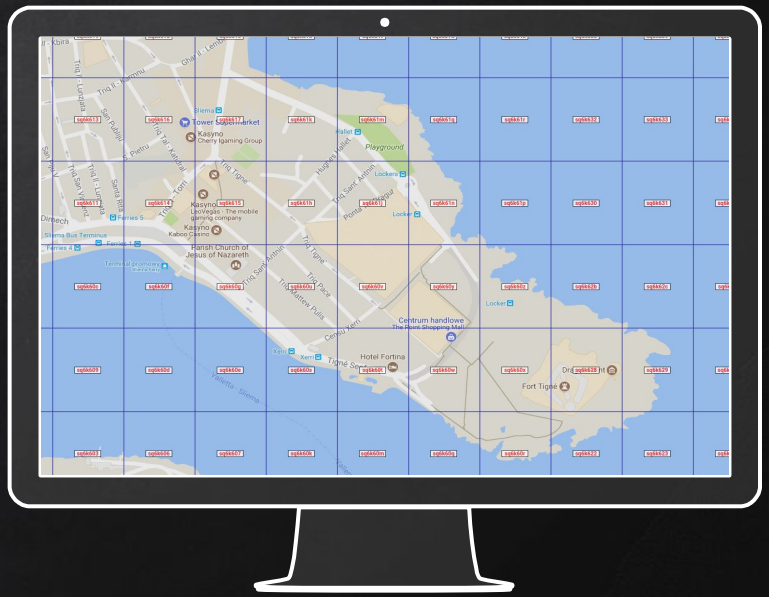
(integer) 2

other commands: PFMERGE

Geospatial

- ✘ Basic geospatial lookups
- ✘ Implemented based on Sorted Sets
- ✘ Based on Geohash (https://en.wikipedia.org/wiki/Geohash)

Use case: players nearby



Geospatial – API example

```
GEOADD map 19.0584 49.8224 Bielsko-Biała
(integer) 1
GEOHASH map Bielsko-Biała
1) "u2ve3tvusb0"
GEOADD map 19.0238 50.2649 Katowice
(integer) 1
GEODIST map Katowice Bielsko-Biała  km
"49.2798"
GEORADIUSBYMEMBER map Katowice 50 km WITHDIST
1) 1) "Katowice"
   2) "0.0000"
2) 1) "Bielsko-Bia\xc5\x82a"
   2) "49.2798"
```

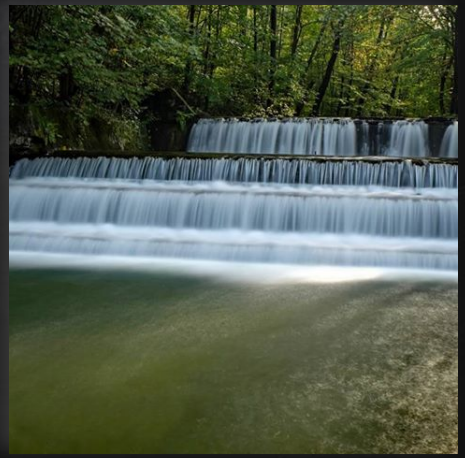
Other commands:
GEORADIUS, GEOPOS

Streams

- ✘ Sequence of data elements
- ✘ Represent infinite, "moving" (continuous) data
- ✘ Provide fixed offset
- ✘ Explorable with range queries
- ✘ Allow parallel publishers and consumers (consumer groups)

- ✘ Inspired by some Kafka concepts
- ✘ Planned to be backported to 4.0 end 2017

http://antirez.com/news/114



Streams – API example (current)

```
XADD winners MAXLEN 1000 * player user:1 amount 100
1511219499656-0

XADD winners MAXLEN 1000 * player user:2 amount 250
1511219509734-0

XRANGE winners - + COUNT 1
1) 1) 1511219499656-0
   2) 1) "player"
      2) "user:1"
      3) "amount"
      4) "100"

XRANGE winners 1511219499656 1511219499657 COUNT 2
1) 1) 1511219499656-0
   2) 1) "player"
      2) "user:1"
      3) "amount"
      4) "100"
```

Streams - API example (current)

PRODUCER

```
XADD games_opened * id starburst
```

CONSUMER

```
XREAD BLOCK 0 STREAMS games_opened $
```

```
1) 1) "games_opened"
   2) 1) 1) 1511221632858-0
         2) 1) "id"
            2) "starburst
(3.00s)
```

```
XREAD BLOCK 0 STREAMS games_opened 1511221632858-0
```

Redis Modules

Not enough?

- ✘ Redis can be extended with modules
- ✘ Exposed C API
- ✘ Existing library of modules (eg. NN, ML, Search)

https://redis.io/modules

Machine Learning and Redis

Model training:

- ➔ Apache Spark
- ➔ TensorFlow
- ➔ ....

So, you have your ML model, and what now?

Redis-ML!

- ➔ Linear regression
- ➔ Logistic regression
- ➔ Decision trees
- ➔ Matrix operations

On top of Redis features!

https://github.com/RedisLabsModules/redis-ml

Programming with Redis

Additional Redis constructs and popular Redis usages

Pipelining

Fact: By default each command is sent individually

Consequence: Cost of network traffic can be significant

Solution: pipelining

On a protocol level it's implemented as sending commands separated with \r\n.

```
redis_ = redis.Redis(...)
pipe = redis_.pipeline()
pipe.set('a', 1)
pipe.set('b', 2)
pipe.execute()
```

Transactions

Redis transactions guarantees:

1. Transaction is guaranteed to be executed without interruptions from other clients.

2. All or none commands in group are executed.
   Commands are executed despite the errors!
   No rollback!

Transactions – API demo

GET x
(nil)

MULTI
OK

SET x 1
QUEUED

SCARD x
QUEUED

SET x 2
QUEUED

EXEC
1) OK
2) (error) WRONGTYPE Operation against a key holding the wrong kind of value
3) OK

GET x
"2"

Pub/Sub

"...senders of messages, called publishers, do not program the messages to be sent directly to specific receivers, called subscribers."

https://en.wikipedia.org/wiki/Publish%E2%80%93subscribe_pattern

Pub/Sub in Redis:

- ➔ Supports pattern matching
- ➔ Ignores keyspace numbers
- ➔ Client can receive single duplicated messages

Pub/Sub – API example

PRODUCER

```
PUBLISH winners:PL user:1
(integer) 1

PUBLISH winners:DE user:2
(integer) 1

PUBLISH winners:MT user:3
(integer) 1
```

CONSUMER

```
SUBSCRIBE winners:PL winners:MT

1) "message"
2) "winners:PL"
3) "user:1"

1) "message"
2) "winners:MT"
3) "user:3"

PSUBSCRIBE winners:*
```

Distributed Lock

Problem: mutually exclusive access to a shared resource in a distributed environment

Single instance solution: SET lock_name lock_id NX PX timeout_value

Redlock algorithm:

- Mutual exclusion
- Deadlock free
- Fault tolerant

Controversy!

https://redis.io/topics/distlock

http://martin.kleppmann.com/2016/02/08/how-to-do-distributed-locking.html

http://antirez.com/news/101

Lua Scripting

Redis allows to execute logic on the server

“Redis stored procedures”

Reduces amount of calls to Redis

```
HMSET player:1 real 100 bonus 200
OK

SCRIPT LOAD "return {redis.call('HGET', KEYS[1], 'real') + redis.call('HGET', KEYS[1], 'bonus')}"
"a238c6486c63d7f26e3204e1c7df0132429d38d7"

EVALSHA a238c6486c63d7f26e3204e1c7df0132429d38d7 1 player:1
1) (integer) 300
```

Redis Administration

Brief overview on topics like persistence, security

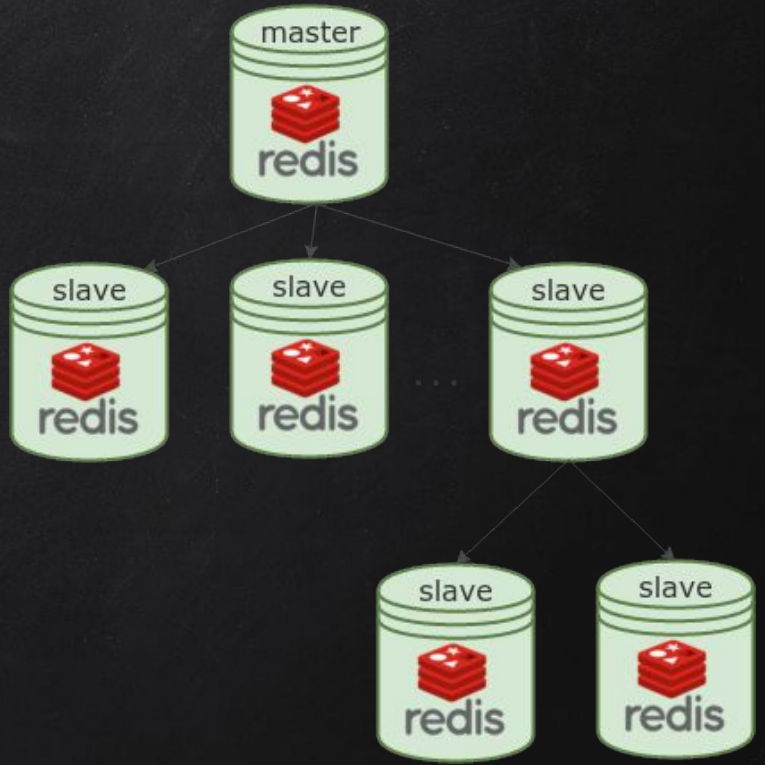
Persistence

Options:

- ✘ No persistence
- ✘ RDB
  - Point-in-time
  - Every X seconds or Y changes
- ✘ AOF
  - Append only log
- ✘ RDB + AOF

Replication

- ✘ master-slave(s)
- ✘ non blocking on master
- ✘ almost non blocking on slave
- ✘ for performance and/or safety




```
slaveof 192.168.1.1 6379
```

Security

- ✘ Designed to be used in trusted environments
- ✘ Provides simple authentication (AUTH command)
- ✘ No data encryption
- ✘ Possibility to disable commands

Consider this prior to go live!

Monitoring

Use SLOWLOG to understand slowest operations

```
SLOWLOG GET 10
```

Use MONITOR while debugging

```
MONITOR
```

Use INFO to get general stats

```
INFO server | clients | memory | persistence | stats | replication | cpu | commandstats |
keyspace | cluster
```

Lessons Learned

Don't do my mistakes, son...

Keyspace

- ✘ Use database 0
  - Ignored by Pub/Sub
  - Does not work well with cluster
- ✘ Structure your keyspace:
  - Use prefixes for keys
  - Have index of prefixes

    John123 ---> player:John123

  Why:
  - Helps with scanning
  - Minimizes "WRONGTYPE Operation against a key holding the wrong kind of value." errors.

Scanning

- ✘ Don't do KEYS
- ✘ Use SCAN:
  - SCAN
  - HSCAN
  - SSCAN
  - ZSCAN

Operations

- ✘ Mind Big O notation
  - Most of commands provide this information
- ✘ Avoid fetching big collections
- ✘ Use pipelines if possible
- ✘ Use multi-key operations (eg. HMSET)
- ✘ Treat Redis as operational DB

Data

- ✘ Use hashes to group objects:
  - HSET player:1 name Łukasz instead of SET player:1:name
- ✘ Keep hashes small
- ✘ Expire keys if possible

Monitor

- ✘ Make SLOWLOG your friend
- ✘ Analyze your apps with MONITOR
- ✘ Mind number of connected clients:
  - echo "INFO" | redis-cli | grep connected_clients

THANKS!

Questions?

You can find me at
@ldziedzia
lukasz.dziedzia@gmail.com

Links

https://redis.io/

http://oldblog.antirez.com/post/redis-persistence-demystified.html

https://dzone.com/articles/an-introduction-to-redis-ml-part-1

https://groups.google.com/forum/#!topic/redis-db/vS5wX8X4Cjg/discussion

Presentation template by SlidesCarnival