UXPADC Conference Highlights

- Golden Ratio
- Boiling Chickens
- Impactful Images / Visual Simplicity

  Obama vs Romney (Take 1)

  Obama vs Romney (Take 2)
- Responsive Web Design

  Boston Globe

Redis in 20 Minutes

Jason Carey
Asymmetrik

The W's

What An open source, networked, in-memory, advanced key-value store written in ANSI C with optional durability

When Released in 2009 and currently sponsored by VMware

Who Salvatore Sanfilippo (aka antirez)

Where and many, many others

Why It fills a gap in the storage solution space

http://www.redis.io

Redis Manifesto

- A DSL for Abstract Data Types
- Memory storage is #1
- Fundamental structures for fundamental API
- Code is like a poem
- Against complexity
- Two levels of API: local vs. distributed
- Optimize for joy

http://antirez.com/post/redis-manifesto.html

Redis Tutorial

http://try.redis-db.com

Redis ASTs

- Strings
- Lists
- Sets
- Hashes
- Sorted Sets

Redis Strings

- Binary safe
- Max length: 512 MB
- Spotlighting operations
  - Atomic Counters (incr, decr, incrby)
  - Atomic Check-then-Set (setnx)
  - Random Access (getrange, setrange, getbit, setbit)

Redis Lists

- List of strings, sorted by insertion order
- Max size: 2^32-1
- Spotlighting operations (l*)
  - Insertion (lpush, rpush, lset)
  - Deletion (lpop, rpop, blpop, brpop, lrem)
  - Atomic (rpoplpush, lpushx, rpushx)
  - Length (llen, ltrim)

Commands start with the letter "l" for list

Redis Sets

- Unordered collection of distinct strings
- Max size: 2^32-1
- Spotlighting operations (s*)
  - Set (sdiff, sinter, sunion)
  - Existence (sismember)
  - Atomic (sdiffstore, sunionstore)

Redis Hashes

- Associations of string fields to string values
- Max size: 2^32-1
- Spotlighting operations (h*)
    - Insertion (hset, hmset)
    - Retrieval (hget, hmget)
    - Atomic Counters (hincrby, hincrbyfloat)
    - Atomic Check-then-Set (hsetnx)

Redis Sorted Sets

- Similar to sets, but each member is associated with a numeric score and sorted in increasing order
- Max size: 2^32-1
- Spotlighting operations (z*)
    - Range (zrange, rangebyscore)
    - Attribute (zrank, zscore)

Beyond the Basics

- Transactions
- Pipelining
- Scripting

Transactions

Allow a group of commands to be executed in a single step

- Commands executed sequentially and no other client's request will be served in the middle of the group
- All or none of the commands are processed
- Execution continues in the presence of command failures

Example: Increment 'foo' and 'bar' atomically

```
> MULTI          // start transaction
OK
> INCR foo       // command queued on server
QUEUED
> INCR bar       // another command queued on server
QUEUED
> EXEC           // transaction committed
1) (integer) 1
2) (integer) 1
```

Transactions

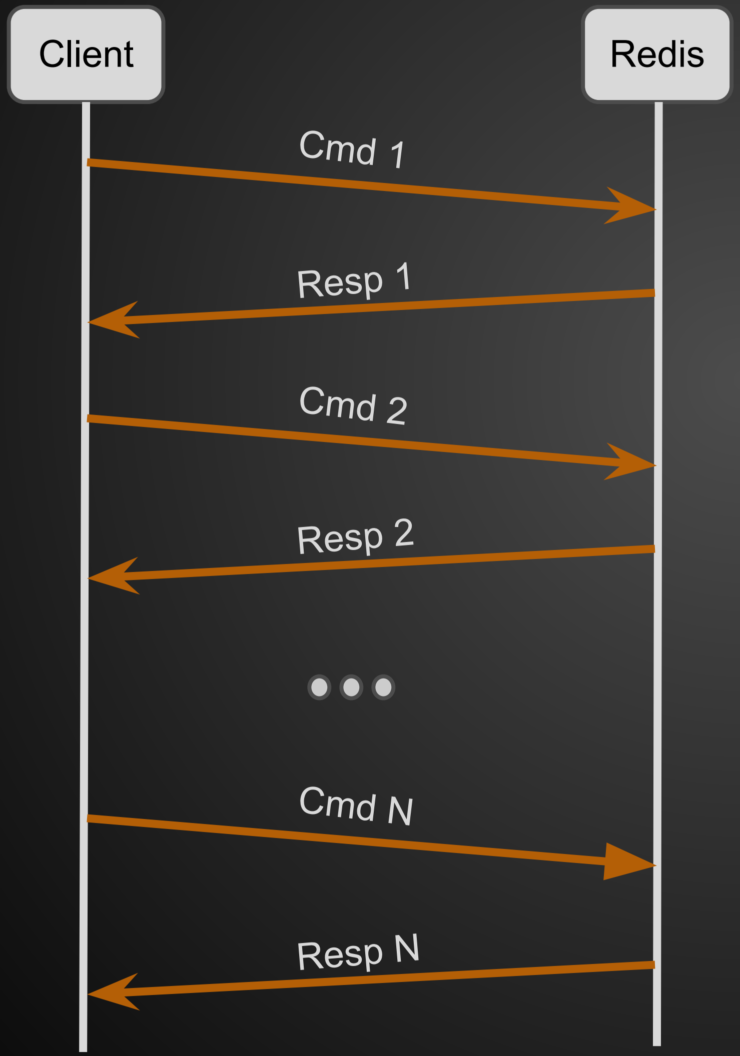
Optimistic locking using check-and-set

- Keys can be "watched" for changes against them
- If at least one watched key is modified before commit of transaction, then transaction is aborted
- All keys are unwatched after transaction is committed or aborted

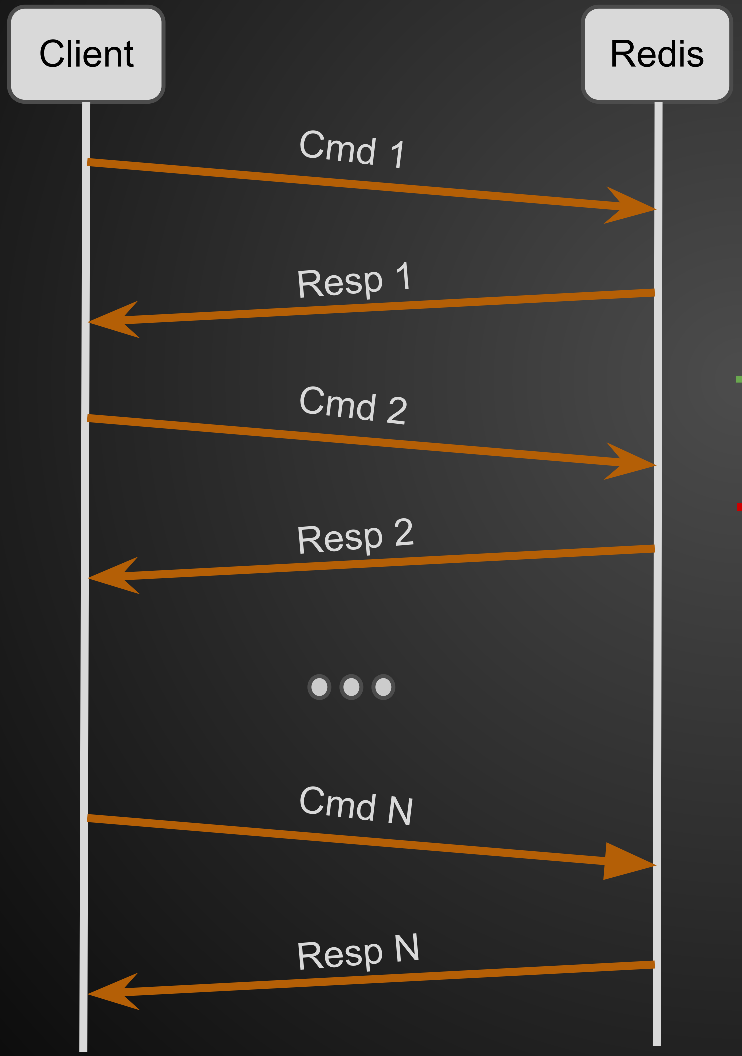
Example: Increment 'mykey' by 1

```
WATCH mykey          // watches 'mykey' for changes
val = GET mykey      // store value of key in client-side variable
val = val + 1        // modify variable's value
MULTI                // start transaction
SET mykey val        // update key
EXEC                 // commit conditioned on key not modified
```

Pipelining

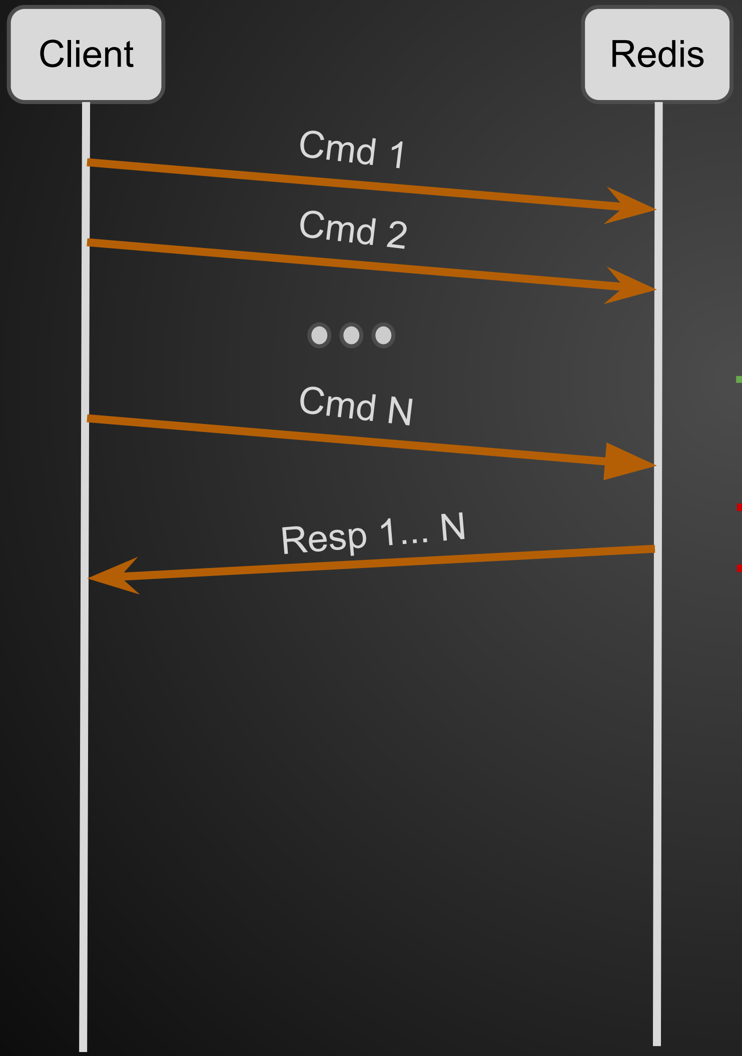


Pipelining



+ Can react to individual response
- Pay RTT cost N times

Pipelining



+ Pay RTT cost 1 time

- Cannot react to individual response
- Server buffers client responses

Scripting

- Redis 2.6 embeds a Lua interpreter
    - Lua (www.lua.org) is a dynamically-typed language with a single native data structure the table (associative array)
    - Small footprint around 200K for interpreter and 200K for library
- Advantage over transactions because data can be read and written atomically with minimal latency
- Scripts can be cached for reduced bandwidth
- Redis puts constraints on contents of script, but not an issue in my experience

Example Lua Script

```
local remainder = ARGV[1]       -- capture arguments
local values = {}               -- holds retrieved values  (even or odd)

for keyIdx, key in ipairs(KEYS) do
  local value = redis.call('get', key)
  if value % 2 == remainder then
    table.insert(values, value)
  else
    redis.call('set', key, value - 1)
  end
  redis.log(redis.LOG_NOTICE, "Key  " .. key .. ", Value " .. value)
end

return values
```

Additional Topics

- Publish/Subscribe
- Persistence
- Security
- Replication
- High Availability
- Sharding
- Clustering

Summary

- Redis is a blazingly fast key-value store, but requires data to fit into memory
- The ASTs and atomic operations enable complex data scenarios to be handled
- Scripting enables bandwidth efficient RW of keys
- Redis has as a lot of potential, but some features may not be ready for production