Redis Streams, Functions and Data Structures

Dave Nielsen
Redis Labs
Agenda

About Redis
Use Cases
Redis Streams
"Most Popular Database on AWS" – Sumo Logic 2016 Survey
Redis Top Differentiators

1. Performance
NoSQL Benchmark

2. Simplicity
Redis Data Structures
- Strings
- Sets
- Bitmaps
- Sorted Sets
- Bit field
- Geospatial Indexes
- Hashes
- Hyperloglog
- Lists
- Streams

3. Extensibility
Redis Modules

Graphs and diagrams showing performance benchmarks and data structures.
Performance: The Most Powerful Database

Highest Throughput at Lowest Latency in High Volume of Writes Scenario

Least Servers Needed to Deliver 1 Million Writes/Sec

<table>
<thead>
<tr>
<th>Application requests/sec</th>
<th>Latency in Milliseconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Couchbase</td>
<td>394.42</td>
</tr>
<tr>
<td>Cassandra</td>
<td>381.31</td>
</tr>
<tr>
<td>Datasync</td>
<td>372.31</td>
</tr>
<tr>
<td>Redis®</td>
<td>71.22</td>
</tr>
</tbody>
</table>

Application Latency (msec)

<table>
<thead>
<tr>
<th>Servers used to achieve 1M writes/sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cassandra</td>
</tr>
<tr>
<td>Couchbase</td>
</tr>
<tr>
<td>Redis®</td>
</tr>
</tbody>
</table>

ANNUAL COST

- Cassandra: $2,226,216
- Couchbase: $371,040
- Redis®: $14,832

COST COMPARED TO REDIS®

- Cassandra: 150X
- Couchbase: 25X

Benchmarks performed by Avalon Consulting Group

Benchmarks published in the Google blog
“REDIS IS FULL OF DATA STRUCTURES!”
Simplicity: Redis Data Structures – ’Legos’

- **Strings**
  "I’m a Plain Text String!"

- **Bitmaps**
  0011010101100111001010

- **Hashes**
  { A: “foo”, B: “bar”, C: “baz” }

- **Lists**
  [ A → B → C → D → E ]

- **Sets**
  { A, B, C, D, E }

- **Sorted Sets**
  { A: 0.1, B: 0.3, C: 100 }

- **Geospatial Indexes**
  { A: (51.5, 0.12), B: (32.1, 34.7) }

- **Bit field**
  {23334}{112345569}{766538}

- **Hyperloglog**
  00110101 11001110

- **Streams**
  →{id1=time1.seq1(A:"xyz", B:"cdf"),
  d2=time2.seq2(D:"abc", )}

- **Sets**
  {  A , B , C , D , E  }

- **Geospatial Indexes**
  { A: (51.5, 0.12), B: (32.1, 34.7) }

- **Hyperloglog**
  00110101 11001110

- **Streams**
  →{id1=time1.seq1(A:"xyz", B:"cdf"),
  d2=time2.seq2(D:"abc", )}

"Retrieve the e-mail address of the user with the highest bid in an auction that started on July 24th at 11:00pm PST”

ZREVRANGE 07242015_2300 0 0
Extensibility: Modules Extend Redis Functionality

- RediSearch
- Redis-ML
- Redis Graph
- ReJSON
- Rebloom
- Neural-Redis
- Redis-Cell
- Redis-TDigest
- Redis-Timerseries
- Redis-Rating
- Redis-Cuckoofilter
- Cthulhu
- Redis Snowflake
- redis-roaring
- Session Gate
- ReDe
- TopK
- countminsketch
Microservices

Click to add text
SOA vs. Microservices
Microservices at Netflix
Monolith or Microservices?
Benefits of Microservices

• Microservices are hot. It seems like everyone is using them
Benefits of Microservices

• Make it perform faster or scale better
• Extend an application’s capabilities more easily
• Add new features more quickly and easily
• Improve maintainability
• Reduce vulnerabilities
But, Microservices are Complicated

• A lot more going on that meets the eye.
Be Prepared for Success

• What to do when your app begins to hockey stick
  • Duck tape the parts when they break?
  • Do you rewrite your app with scalability in mind?
You Can Do Both with Redis & Kubernetes

• Redis became famous by solving web scale data problems
  • Remember the Twitter Fail Whale?
• Kubernetes became famous by solving hockey stick problem
  • Remember Pokemon Go?
And Scale with Redis and Microservices

• In many cases, Monolith is the right way to start
• Smaller apps and small teams don’t need the overhead and unnecessary complexity of Microservices Architecture
• But when its time to scale, use Redis and Microservices
Use Cases

Click to add text
Use Cases

Top 4
• Cache
• Session Store
• Metering
• Fast Data Ingest

More:
• Primary Database
• Real-time Analytics
• Messaging
• Recommendations
• High-speed Transactions
• Search – RediSearch
• Geo Spatial Indexing
• Many more …
1. Redis as a Cache

When to use
- Frequent reads, infrequent writes
- Data is shared between user sessions

Examples:
- Pictures, documents, videos, statements, reports, etc.
2. Redis as a Session Store

When to use
Session based apps with frequent reads and writes
Data is isolated between sessions

Examples:
e-Commerce, gaming, social applications, etc.
In a simple world

Internet ↔ Server ↔ Database
Good problems

Traffic Grows...

Internet  Server  Database

Struggles
Good solution

Internet → Server

Session storage on the server

Server → Database

performance restored
More good problems

Internet

Server

Session storage on the server

Database

Struggling
Problematic Solutions

Internet

Server

Database

Load balanced

Session storage on the server
Multiple Servers + On-server Sessions?

Server #1 – Hello Robin!
Multiple Servers + On-server Sessions?

Server #3 – Hello ????

Robin

Server

Database
Better solution

Internet

Server

Database

Load balanced

Redis

Session Storage
hash key: usersession:1

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>userid</td>
<td>8754</td>
</tr>
<tr>
<td>name</td>
<td>dave</td>
</tr>
<tr>
<td>ip</td>
<td>10:20:104:31</td>
</tr>
<tr>
<td>hits</td>
<td>1</td>
</tr>
<tr>
<td>lastpage</td>
<td>home</td>
</tr>
</tbody>
</table>

```
HMSET usersession:1 userid 8754 name dave ip 10:20:104:31 hits 1
HMGET usersession:1 userid name ip hits
HINCRBY usersession:1 hits 1
HSET usersession:1 lastpage "home"
HGET usersession:1 lastpage
HDEL usersession:1 lastpage
DEL usersession:1
```

Hashes store a mapping of keys to values – like a dictionary or associative array – but faster
Use Case: Rate-limiting

Limit the peak load on your legacy database by limiting the number of queries per second to the highest threshold.

How Redis helps you?

• Built-in counters
• Time-to-live
• Single-threaded architecture assures serializability
4. Redis for Fast Data Ingest

Use Cases:

• Real-time analytics
• IoT
• Log collection, time-series

How Redis helps you?

• Pub/Sub
• List
• Sorted Set
Do more with Redis

- Caching
- Session Store
- Metering
- Fast Data Ingest

- Primary Database
- Real-time Analytics
- Messaging
- Recommendations
- High-speed Transactions
- Search – RediSearch
- Geo Spatial Indexing

It’s a Swiss Army Knife for data processing
Managing Leaderboards w/ Redis Sorted Sets
Leaderboard with Sorted Sets Example

The Problem

- MANY users playing a game or collecting points
- Display real-time leaderboard.
- Who is your nearest competition
- Disk-based DB is too slow

Why Redis Rocks

- **Sorted Sets** are perfect!
- Automatically keeps list of users sorted by score
- ZADD to add/update
- ZRANGE, ZREVRANGE to get user
- ZRANK will get any users rank instantaneously
Redis Sorted Sets

ZADD game:1 10000 id:1
ZADD game:1 21000 id:2
ZADD game:1 34000 id:3
ZADD game:1 35000 id:4
ZADD game:1 44000 id:3
or
ZINCRBY game:1 10000 id:3

ZREVRANGE game:1 0 0
ZREVRANGE game:1 0 1 WITHSCORES
Redis Streams

• 1st class Redis citizens
• An abstract data type that is not unlike a log
• Designed with time series data in mind
• Provide some "Kafkaesque" messaging abilities
Why invent yet another Redis thingamajig?

Necessity is the mother of invention
There ain't no such thing as a free lunch
The existing (i.e. lists, sorted sets, PubSub) isn't "good enough" for things like:
- Log-like data patterns
- At-least-once messaging with fan-out
And listpacks, radix trees & reading Kafka :)

In-Memory Computing SUMMIT NORTH AMERICA 2014
The Log is hardly a new thing

A storage abstraction that is:
• Append-only, can be truncated
• A sequence of records ordered by time

A Logical Log is:
• Based on a logical offset, i.e. time (vs. bytes)
• Therefore time range queries
• Made up of in-memory data structures, naturally
Logging streams of semi-structured data

A data stream is a sequence of elements. Consider:

• Real time sensor readings, e.g. particle colliders
• IoT, e.g. the irrigation of avocado groves
• User activity in an application

...  
• Messages in distributed systems
A side note about Distributed Systems

“A distributed system in which components located on networked computers communicate and coordinate their actions by passing messages” – Distributed Computing, Wikipedia

Includes: client-server, 3/n-tier, peer to peer, SOA, micro- & nanoservices, FaaS & serverless...
An observation

There are only two hard problems in distributed systems:

2. Exactly-once delivery
1. Guaranteed order of messages
2. Exactly-once delivery

- Mathias Verraes, on Twitter
Fact #1: you can choose one and only one:
• *At-most-once* delivery, i.e. "shoot and forget"
• *At-least-once* delivery, i.e. explicit ack

Fact #2: *exactly-once* delivery doesn't exist

Observation: order is usually important (duh)
This isn’t exactly a new challenge

Consider the non-exhaustive list at taskqueues.com

• 17 message brokers, including: Apache Kafka, NATS, RabbitMQ and Redis

• 17 queue solutions, including: Celery, Kue, Laravel, Sidekiq, Resque and RQ <- all these use Redis as their backend btw ;)

And that’s without considering protocol-based etc.
So again, why "reinvent hot water"?

Redis (in general and) Streams (in particular) are:

• Everywhere, from the IoT's edge to the cloud
• Blazing fast, massive throughput
• Usable from all(most) languages and platforms (IoT microcontrollers included)

Note: apropos IoT, they are great async buffers
Redis Streams “formalism”

A stream is a sequence of entries (records). It:
• Is "sharded" by key ("topic")
• Has 1+ producers
• Has 0+ consumers
• Can provide *at-most-* or *at-least-once* semantics
• Enables stream processing/real time pipelines (as opposed to batch)
A picture of a stream
Entries in the Stream

Every entry has a unique ID that is its logical offset. The ID is in following format:

<epoch-milliseconds>-<sequence>

Note: each ID part is a 64-bit unsigned integer

An entry also has one or more ordered field-value pairs, allowing for total abstraction (the empty string is a valid field name, good for time series).
Adding Entries

# Adding entries
redis> XADD <key> <* | id>
    [MAXLEN [~] <n>]
    <field> <value> [...]
    <epoch-milliseconds>-<sequence>

# Stream length
redis> XLEN <key>
(integer) <stream-length>
# Iterating

redis> X[REV]RANGE <key> <start> <stop> [COUNT <n>]

1) 1) <entry-id>
2) 1) <field1>
   2) <value1>
3) ...
Blocking Read

```bash
# [Blocking] read
redis> XREAD [BLOCK <milliseconds>] STREAMS <key> [...] <start> [...]

1) 1) <entry-id>
  2) 1) <field1>
     2) <value1>
     3) ...
```
# And the usual Redis goodness, e.g. TX
redis> MULTI
...
# Or server-side processing
redis> EVAL "return 'Lua Rocks!'" 0
...
# Or your own custom module
redis> MODULE LOAD <your-module-here>
OK
The problem with scaling consumers

A consumer of a stream gets all entries in order, and will eventually become a bottleneck.

Possible workarounds:

• Add a "type" field to each record - that's dumb
• Shard the stream to multiple keys - meh
• Have the consumer dispatch entries as jobs in queues … GOTO 10
Consumer Groups

"... allow multiple consumers to cooperate in processing messages arriving in a stream, so that each consumer in a given group takes a subset of the messages."

Shifts the complexity of recovering from consumer failures and group management to the Redis server.
We are here :)  

- Groups are named and are explicitly (!) created: 
  
  \texttt{XGROUP CREATE temps agg $}  

- Consumers are also named, and each gets only a subset of the stream:  
  
  \texttt{XREAD-GROUP GROUP agg CONSUMER escher-01 STREAMS temps >}  

- XACK/NOACK in XREAD, XCLAIM, XPENDING
Redis Streams status

• Expected to be GA within a month or so (est. Oct 2018)
Try it yourself

From your browser: https://try.redis.io
Or download it: https://redis.io/download
Or clone it: https://github.com/antirez/redis
Or dockerize it: docker run -it redis
Or try Redis Enterprise by https://redislabs.com
Questions

Dave Nielsen

dave@redislabs.com
@davenielsen