RedisGears – Redis in memory data processing

JUNE 2019 | PIETER CAILLIAU
About me

• Produced in Belgium

• (instanceof) SE @ TomTom
• Consultant @ neo4j
• Solution Architect @ Redis Labs
• Product Manager @ Redis Labs

• @cailliaup
Agenda

1. What is Redis and Redis Enterprise
2. Stream Processing with RedisGears
3. RedisGears as a Multimodel Engine
Redis is Fast ...
... Extremely Fast
## DB-Engines Ranking

<table>
<thead>
<tr>
<th>Rank</th>
<th>DBMS</th>
<th>Score Sep 2018</th>
<th>Score Aug 2018</th>
<th>Score Sep 2017</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Oracle</td>
<td>1309.12</td>
<td>-2.91</td>
<td>-49.97</td>
</tr>
<tr>
<td>3.</td>
<td>Microsoft SQL Server</td>
<td>1051.28</td>
<td>-21.37</td>
<td>-161.26</td>
</tr>
<tr>
<td>4.</td>
<td>PostgreSQL</td>
<td>406.43</td>
<td>-11.07</td>
<td>+34.07</td>
</tr>
<tr>
<td>5.</td>
<td>MongoDB</td>
<td>358.79</td>
<td>+7.81</td>
<td>+26.06</td>
</tr>
<tr>
<td>6.</td>
<td>DB2</td>
<td>181.06</td>
<td>-0.78</td>
<td>-17.28</td>
</tr>
<tr>
<td>7.</td>
<td>Elasticsearch</td>
<td>142.61</td>
<td>+4.49</td>
<td>+22.61</td>
</tr>
<tr>
<td>8.</td>
<td>Redis</td>
<td>140.94</td>
<td>+2.37</td>
<td>+20.54</td>
</tr>
<tr>
<td>9.</td>
<td>Microsoft Access</td>
<td>133.39</td>
<td>+4.30</td>
<td>+4.58</td>
</tr>
<tr>
<td>10.</td>
<td>Cassandra</td>
<td>119.55</td>
<td>-0.02</td>
<td>-6.65</td>
</tr>
</tbody>
</table>

345 systems in ranking, September 2018
And you’ve been using it already
Redis is Extensively and Diversely Used

**Uses Redis for:**
- **Timeline, following**
  - *Scope: 10-20 TB*

**Uses Redis for:**
- **Local/site/global caching**
  - *Scope: 10-20 TB*

**Uses Redis for:**
- **Repository router**
  - *Scope: 10+ TB*

**Uses Redis for:**
- **Geo search, user profiles**
  - *Scope: 40 TB*
Redis Top Differentiators

1. Performance
   NoSQL Benchmark

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

3. Extensibility
   Redis Modules
   - Lists
   - Hashes
   - Bitmaps
   - Strings
   - Sets
   - Sorted Sets
   - Geospatial Indexes
   - Hyperloglog
# Redis Speed differentiators

<table>
<thead>
<tr>
<th>OPTIMIZED ARCHITECTURE</th>
<th>ADVANCED PROCESSING</th>
<th>EFFICIENT OPERATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓ Written in C</td>
<td>✓ Most commands are executed with $O(1)$ complexity</td>
<td>✓ Easy to parse networking protocol</td>
</tr>
<tr>
<td>✓ Served entirely from memory</td>
<td>✓ Access to discrete elements within objects</td>
<td>✓ Pipelining for reduced network overhead</td>
</tr>
<tr>
<td>✓ Single-threaded, lock free</td>
<td>✓ Reduced bandwidth/overhead requirements</td>
<td>✓ Connection pooling</td>
</tr>
</tbody>
</table>
Redis Speed differentiators

<table>
<thead>
<tr>
<th>OPTIMIZED ARCHITECTURE</th>
<th>ADVANCED PROCESSING</th>
<th>EFFICIENT OPERATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓ Written in C</td>
<td>✓ Most commands are executed with O(1) complexity</td>
<td>✓ Easy to parse networking protocol</td>
</tr>
<tr>
<td>✓ Served entirely from memory</td>
<td>✓ Access to discrete elements within objects</td>
<td>✓ Pipelining for reduced network overhead</td>
</tr>
<tr>
<td>✓ Single-threaded, lock free</td>
<td>✓ Reduced bandwidth/overhead requirements</td>
<td>✓ Connection pooling</td>
</tr>
</tbody>
</table>
Modules Extend Redis Infinitely

• Create your own data types and commands
• Reuse Redis’ simplicity, performance, scalability and high availability.
• Can be written in C/C++/Go/Python/Rust/Zig
• Leverage existing data structures
• Turn Redis into a Multi-Model database

https://redislabs.com/community/redis-modules-hub/
Redis Modules

- RediSearch (GA)  [redisearch.io]
- RedisBloom (GA)  [redisbloom.io]
- RedisTimeSeries  [redistimeseries.io]
- RedisJSON (GA)   [redisjson.io]
- RedisAI          [redisai.io]
- RedisGraph (GA)  [redisgraph.io]
Introducing Redis Enterprise
Redis Enterprise

DBaaS
• Available since mid 2013
• 8,500+ enterprise customers

Software
• Available since early 2015
• 300+ enterprise customers

Customers
• 6 of top Fortune 10 companies
• 3 of top 5 communications companies

• 3 of top 4 credit card issuers
• 3 of top 5 healthcare companies
Cloud Providers have different incentives

- Cloud Provider
  - Higher margin by
    - Idleness
  - Cloud Lock-in

- DBAAS Provider
  - Higher margin by better resource utilization
    - Multi-tenancy
    - Reducing RAM
    - CPU utilization
# Redis Enterprise: A Unique Primary Database

<table>
<thead>
<tr>
<th>FAST</th>
<th>RELIABLE</th>
<th>FLEXIBLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>HIGHEST PERFORMANCE, LINEAR SCALING</td>
<td>BUILT-IN HIGH PERFORMANCE SEARCH</td>
<td>MULTI-MODEL</td>
</tr>
<tr>
<td>HIGH AVAILABILITY WITH INSTANT FAILOVER</td>
<td></td>
<td>FLEXIBLE DEPLOYMENT OPTIONS (CLOUD, ON-PREM, HYBRID)</td>
</tr>
<tr>
<td>DURABILITY AT MEMORY SPEEDS</td>
<td></td>
<td>INTELLIGENT TIERED DATA ACCESS (RAM &amp; FLASH MEMORY)</td>
</tr>
<tr>
<td>ACTIVE-ACTIVE GEO DISTRIBUTION (CRDT-BASED)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Redis Enterprise Cluster

Uneven number of symmetric nodes

Node 1  Node 2  Node N (odd number)
Redis Enterprise Cluster

Single master database

Node 1  Node 2  Node N (odd number)
Redis Enterprise Cluster

An HA database

Node 1
Node 2
Node N (odd number)
Redis Enterprise Cluster
A Clustered Database
How do keys get assigned to partitions?

\[ \text{hash slot} = \text{CRC16(key)} \mod 16384 \]
How do keys get assigned to partitions?

\[
\text{hash\_slot} = \text{CRC16(key)} \mod 16384
\]

def HASH_SLOT(key):
    s = key.index "{"  
    if s
        e = key.index ""},s+1
        if e & e != s+1
            key = key[s+1..e-1]
        end
    end
    crc16(key) % 16384
end
Redis Enterprise Cluster

A Highly Available Clustered Database

Node 1

Node 2

Node N (odd number)
Redis Enterprise Node

Enterprise Layer

- Cluster Manager
- REST API
- Redis Shards

Zero latency proxy

Open Source Layer
Redis Enterprise: Shared Nothing Symmetric Architecture

Data-Path and Control/Management Path Separation

Cluster Management Path

Data Path

Node 1

Node 2

Node N (odd number)

Node Watchdog Cluster Watchdog

Redis Shards & Proxies
Microservices Architecture and Polyglot Persistence

- **Authentication**
  - Key/Value
  - Graph
  - RDBMS

- **Customers**
  - Key/Value
  - Graph
  - RDBMS

- **Fulfilment**
  - Key/Value
  - Graph
  - RDBMS

- **Session Store**
  - Key/Value
  - Graph
  - RDBMS

- **Search**
  - Key/Value
  - Graph
  - RDBMS

- **Document**
  - Key/Value
  - Graph
  - RDBMS

- **Catalog**
  - Key/Value
  - Graph
  - RDBMS

- **Fraud Detection**
  - Key/Value
  - Graph
  - RDBMS

- **Columnar**
  - Key/Value
  - Graph
  - RDBMS

- **API**

- **Microservices**
  - Document
  - RDBMS
  - Cache
The Cost of Polyglot Persistence

**Increased application complexity → Costly communication**
Application does heavy lifting in sharing data, keeping data sets in sync

**High operational burden → Higher cost of ownership**
Different databases have specialized administrative, scaling, availability requirements

**Sub-optimal Resource Usage → Higher cost**
Dedicating pods/servers for each type of database reduces deployment efficiency
Redis Enterprise: A Multi-model Database for Microservices

- Authentication
- Customers
- Catalog
- Search
- Fraud Detection
- Session Store
- Fulfilment

API

Key/Value
Graph
Key/Value
Cache
Search
Key/Value
Document
Cache

RDBMS
RDBMS
Built-in Pub-Sub / Streams for event synch across data stores
What are we missing?

• How to consume messages in this “built in message borker”
• Given a sharded database, how can I run analytical queries?

• Multi Model database
  – Single copy in core datatypes
  – Inter module communication
  – Component X doing translations between modules.
Introducing RedisGears
What is RedisGears?

RedisGears is a **Serverless** engine for **multi-model and cluster operations** in Redis, supporting both **event driven** as well as **batch operations**.
Gears infrastructure is written in C
Scripting with RedisGears

RedisGears allows to define a pipe of operations

• Returning value from one operation is passed to the next operation in the pipe
• Last operation returning the result to the user
• First operation is called ‘reader’ - responsible for providing data
  – Keys reader - read keys from Redis
  – Stream reader - read streams from Redis
  – Python reader - allow to user to write his own readers in python
Supported Operations

- Map
- FlatMap
- Filter
- Groupby + Reduce
- Aggregate
- Sort
- Limit
- ForEach
- Distinct
Using RedisGears – (Flat)Mapping
Using RedisGears - Filtering
Using RedisGears - Aggregate
Use Case #1 – Stream Processing

Gears has a streaming API to allow to trigger gears execution on events.

- Redis Stream events - Trigger an execution whenever a new data enters a stream
- Redis Keys events - Trigger an operation whenever a key is touched

Every sec

Redis Streams

RedisTimeSeries
Use Case #2 – a MultiModel Engine

Because of RedisGears’ flexibility (it's actually running python) you can achieve internal module integration with it:

– Read from hashes and index in RediSearch/RedisGraph
– Read RedisJSON data and pass to RedisTimeSeries
– …
Recipe #1 – even triggering

Build a gear that creates maintains a set of all keys within redis

```python
# create the builder
definition = GearsBuilder()
# filter events on key:'all_keys'
definition.filter(lambda x: x['key'] != 'all_keys')
# add the keys to 'all_keys' set
definition.map(lambda x: execute('sadd', 'all_keys', x['key']))
# register the execution on key space notification
definition.register()
```
Recipe #2 – map reducing

Build a **gear** that counts how often a genre is used within a set of movies

```python
# create the pipe builder. KeysOnlyReader is a performance improvement only piping the keys.
builder = GearsBuilder('KeysOnlyReader')

# get from each hash the genres field
builder.map(lambda x: execute('hget', x, 'genres'))

# filter those who do not have genres
builder.filter(lambda x: x is not None)

# split genres by comma
builder.flatmap(lambda x: x.split(',','))

# count for each genre the number of times it appears
builder.countby()

# start the execution
builder.run('movie:*')
```
Build a gear that consumes a stream and updates keys accordingly

```python
# create the builder with a StreamReader
builder = GearsBuilder('StreamReader')

# extract each field value pair from the message and increase the pipe granularity
builder.flatmap(lambda x: [(a[0], a[1]) for a in x.items()])

# filter out the streamId itself
builder.filter(lambda x: x[0] != 'streamId')

# make sure the gears data lives in the correct shard
builder.repartition(lambda x: x[0])

# apply each field value pair to a key
builder.foreach(lambda x: execute('set', x[0], x[1]))

# register on new messages on the stream 'inputStream'
builder.register('inputStream')
```
Example Trigger Explained
Example Trigger Explained
Example Trigger Explained
Example Trigger Explained - Flatmap
Example Trigger Explained - Repartition
Example Trigger Explained - `executeCommand`
Demo
Demo Setup

XADD all → XADD cats → Resized image → Classification → RedisAI

RedisGears

XREAD all → XREAD cats

Mobilenet (v2)

54
Challenge?

- [https://github.com/RedisGears/AnimalRecognitionDemo](https://github.com/RedisGears/AnimalRecognitionDemo)
- #redisfoundmycat
Redis Modules

- RediSearch (GA)  
  redisearch.io
- RedisBloom (GA)  
  redisbloom.io
- RedisTimeSeries  
  redistimeseries.io
- RedisJSON (GA)  
  redisjson.io
- RedisAI  
  redisai.io
- RedisGraph (GA)  
  redisgraph.io
Thank you!
pieter@redislabs.com