Real-time machine learning with Redis, Apache Spark, TensorFlow, and more
Agenda

- Intro to Redis and Redis Labs - 10 min
- Using Neural Redis and Redis-ML for Model Serving - why and how - 15 min
- Exploring the MovieLens Dataset and classification using Spark-ML and Redis-ML - 10 min
- QA
Redis Labs – Home of Redis

The commercial company behind Open Source Redis

Provider of the Redis Enterprise (Redis®) technology, platform and products

Founded in 2011
HQ in Mountain View CA, R&D center in Tel-Aviv IL
Redis Labs Products

**SERVICES**

- **Redis® Cloud**
  - Fully managed Redis® service on hosted servers within AWS, MS Azure, GCP, IBM Softlayer, Heroku, CF & OpenShift

- **Redis® Cloud Private**
  - Fully managed Redis® service in VPCs within AWS, MS Azure, GCP & IBM Softlayer

**SOFTWARE**

- **Redis® Pack**
  - Downloadable Redis® software for any enterprise datacenter or cloud environment

- **Redis® Pack Managed**
  - Fully managed Redis® Pack in private data centers

RAM & Flash
Mature and Stable Technology & Products

250K+
DATABASES RUN OVER 3 YEARS

600+
NEW DATABASES CREATED EVERY DAY

1,000+
CLOUD NODE FAILURE AND OUTAGES EVENTS SURVIVED WITH NO DATA LOSS

100 +
MAN-YEARS OF ENTERPRISE REDIS TECHNOLOGY DEVELOPMENT

50 +
DEDICATED REDIS ENGINEERS

13
GRANTED AND PENDING PATENTS
Fast Growing Business

Available since mid-2013
6,900+ enterprise customers

Available since early-2015
200+ enterprise customers

60,000+ customer accounts
# Significant New Customer Wins

<table>
<thead>
<tr>
<th>Banks</th>
<th>Financial Services</th>
<th>Retail/E-commerce</th>
<th>Social</th>
<th>Media</th>
<th>Advertising</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMERICAN EXPRESS</td>
<td>VISA</td>
<td>jet</td>
<td>twitter</td>
<td>msn</td>
<td>HAVAS</td>
</tr>
<tr>
<td>CRÉDIT AGRICOLE</td>
<td>intuit</td>
<td>Starbucks</td>
<td>Atlassian</td>
<td>The Motley Fool</td>
<td>REVMOB</td>
</tr>
<tr>
<td>fiserv.</td>
<td>Groupon</td>
<td>twitch</td>
<td>bleacher report</td>
<td>Outbrain</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Technology</th>
<th>Communications</th>
<th>Business Services</th>
<th>Travel</th>
<th>Gaming</th>
<th>Education</th>
</tr>
</thead>
<tbody>
<tr>
<td>CISCO</td>
<td>COMCAST</td>
<td>Ariba</td>
<td>Expedia</td>
<td>EA</td>
<td>its Learning</td>
</tr>
<tr>
<td>DELL</td>
<td>vodafone</td>
<td>PROSPERWORKS</td>
<td>Hotel Tonight</td>
<td>Playtika</td>
<td>HealthStream</td>
</tr>
<tr>
<td>verizon</td>
<td>RingCentral</td>
<td>tripadvisor</td>
<td>Roblox</td>
<td>Edgenuity</td>
<td></td>
</tr>
</tbody>
</table>
Redis Tops Database Popularity Rankings

- **sumologic**: #1 database technology on AWS
- **stackoverflow**: The most loved database
- **node**: #1 database used by Node.js developers
- **stackoverflow**: #1 database in Top Paying Technologies
- **stackshare**: #1 NoSQL among Top 10 Data Stores
- **DATADOG**: #1 database on Docker
- **G2 CROWD**: #1 NoSQL in User Satisfaction
- **DB-Engines**: #1 in growth among top 3 NoSQL databases
- **UpScored**: #1 database in skill demand
Redis Main Differentiations

Performance

Simplicity
(through Data Structures)

Extensibility
(through Redis Modules)
A Quick Recap of Redis

Strings / Bitmaps / BitFields

Hash Tables (objects!)

Linked Lists

Sets

Sorted Sets

Geo Sets

HyperLogLog

"I'm a Plain Text String!"

{ A: "foo", B: "bar", C: "baz" }

[ A → B → C → D → E ]

{ A, B, C, D, E }

{ A: 0.1, B: 0.3, C: 100, D: 1337 }

{ A: (51.5, 0.12), B: (32.1, 34.7) }

00110101 11001110 10101010
What's Redis Modules

- Any C/C++ program can now run on Redis
- Modules are a native way to extend Redis for new use cases and functions
- Use existing or add new data-structures
- Enjoy simplicity, infinite scalability and high availability while keeping the native speed of Redis
- Can be created by anyone
<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neural Redis</td>
<td>Simple Neural Network Native to Redis</td>
</tr>
<tr>
<td>ReJSON</td>
<td>JSON Engine on Redis. Pre-released</td>
</tr>
<tr>
<td>Rate Limiter</td>
<td>Based on Generic Cell Rate Algorithm (GCRA)</td>
</tr>
<tr>
<td>Redis-ML</td>
<td>Machine Learning Model Serving</td>
</tr>
<tr>
<td>Time Series</td>
<td>Time series values aggregation in Redis</td>
</tr>
<tr>
<td>Crypto Engine Wrapper</td>
<td>Secure way to store data in Redis via encrypt/decrypt with various Themis primitives</td>
</tr>
<tr>
<td>Graph</td>
<td>Graph database on Redis based on Cypher language</td>
</tr>
<tr>
<td>RediSearch</td>
<td>Full Text Search Engine in Redis</td>
</tr>
<tr>
<td>Secondary Index/RQL</td>
<td>Indexing + SQL -like syntax for querying indexes. Pre-released</td>
</tr>
</tbody>
</table>
Machine Learning/Deep Learning Modules

**NEURAL REDIS**

- Simple neural network as a native data type for Redis
- Training and classification/regression in one place with simple, intuitive APIs
- Very fast and accurate for real time non-visual ML use cases

**REDIS-ML**

- Models can be stored, retrieved and updated natively with Redis-ML
- Accelerates complex ML models by x100
- Unified and simplified ML serving operation
- Currently available with Spark ML
- Other ML/DL platforms coming soon
Neural Redis
• Developed by Salvatore
• Training is done inside Redis
• Online continuous training process
• Builds Fully Connected NNs
The Simple Machine Learning Lifecycle

Train and Serve with Neural-Redis

Train and serve model from Redis

Serving Client

Redis-ML

Client App

Client App

Client App
>NR.CREATE net REGRESSOR 2 3 -> 1 NORMALIZE DATASET 50 TEST 10
>NR.OBSERVE net 4 5 -> 9
>NR.OBSERVE net 1 1 -> 2
....
>NR.OBSERVE net 5 6 -> 11

>NR.TRAIN net AUTOSTOP
>NR.RUN net 1 1
1) "2.1406970024108887"
>NR.RUN net 3 5
1) "8.3342075347900391"

(1) Training
(2) Creating a model
(3) Serving the model
Typical Machine Learning Lifecycle with Spark-ML

Data is loaded into Spark → Spark Training → File System → Custom Server → Client App

- Model is saved in files
- Model is loaded to your custom app
- Serving Client
ML Models Serving Challenges

- Models are becoming bigger and more complex
- Can be challenging to deploy
- Do not scale well, speed and size
- Reliable services are hard to do
- Can be very expensive
The New Machine Learning Lifecycle

Train with Spark-ML, Serve with Redis-ML

- Data loaded into Spark
- Model is saved in Redis-ML

Any Training Platform

Serving Client

Client App

Client App

Client App
Redis-ML – ML Serving Engine

- Store training output as “hot model”
- Perform evaluation directly in Redis
- Easily integrate existing C/C++ ML libs
- Can be tuned on-the-fly
- Enjoy the performance, scalability and HA of Redis
Redis-ML

ML Models

- Tree Ensembles
- Linear Regression
- Logistic Regression
- Matrix + Vector Operations
- More to come...
Random Forest Model

• A collection of decision trees
• Supports classification & regression
• Splitter Node can be:
  ◦ Categorical (e.g. day == “Sunday”)
  ◦ Numerical (e.g. age < 43)
• Decision is taken by the majority of decision trees
Titanic Survival Predictor on a Decision Tree

Sex = Male?

YES
Age < 9.5?

NO
Survived

*Died

*Sibps > 2.5?

*Died

Survived

*Sibps = siblings + spouses
Titanic Survival Predictor on a Random Forest

**Tree #1**
- Sex = Male?
  - YES: Age < 9.5?
    - YES: *Sibps > 2.5? (Died)
    - NO: Died
  - NO: Survived

**Tree #2**
- Country = US?
  - YES: State = CA?
    - NO: Height > 1.60m? (Died)
    - YES: I.Q < 100? (Died)
  - NO: Survived

**Tree #3**
- Weight < 80kg?
  - YES: Eye color = blue? (Survived)
  - NO: Died
Would John Survive The Titanic

• John’s features:
  \{male, 34, married + 2, US, CA, 1.78m, 78kg, 110iq, blue eyes\}

• Tree#1 – Survived

• Tree#2 – Failed

• Tree#3 – Survived

• Random forest decision - Survived
Real World Challenge

- Ad serving company
- Need to serve 20,000 ads/sec @ 50msec data-center latency
- Runs 1k campaigns → 1K random forest
- Each forest has 15K trees
- On average each tree has 7 levels (depth)
## Large/Accurate Models are Expensive to Serve!

<table>
<thead>
<tr>
<th>Item</th>
<th>Calculation</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random forest ops/sec</td>
<td>20K (ads/sec) x 1K (forests) x 15K(trees) x 7 x 0.5 (levels)</td>
<td>1.05 trillion</td>
</tr>
<tr>
<td>Max ops/sec on the strongest AWS instance vcore</td>
<td>2.6Ghz x 0.9 (OS overhead) x 0.1 (10 lines of code per ops) x 0.1 (Java overhead)</td>
<td>23.4 million</td>
</tr>
<tr>
<td># of vcores needed</td>
<td>1.1 trillion / 23.4 million</td>
<td>44,872</td>
</tr>
<tr>
<td># of c4.8xlarge instances needed</td>
<td>44,872 / 36</td>
<td>1,247</td>
</tr>
<tr>
<td>Total cost reserved instances</td>
<td>1,247 x 9213</td>
<td>~$11.5M/yr</td>
</tr>
</tbody>
</table>
Ads Model Serving: Homegrown vs. Redis\textsuperscript{e} + ML

Cut computing infrastructure by 97%

Homegrown

1,247 x c4.8xlarge

Redis\textsuperscript{e}

35 x c4.8xlarge
Redis ML with Spark ML

40x Faster
Classification Time Over Spark

Random Forest; 1,000 forests @ 15,000 trees
Other Challenges

• How do you manage multiple model types
  (Random Forest, Gradient Boosted Trees, Logistic Regression, etc.)

• How do you manage multiple versions of each model

• How do you upgrade a model across so many machines

• What if the training and serving apps are written in different languages

• Etc.
Real World Example: Movie Recommendation System
Demo overview

1. GroupLens
2. Database
3. Spark Training
4. Redis-ML
Step 1: Transform

GroupLens → Database → Spark Training → Redis-ML
Step2: Train and Load to Redis

```scala
val rf = new RandomForestClassifier().setFeatureSubsetStrategy("auto").setLabelCol("indexedLabel").setFeaturesCol("indexedFeatures").setNumTrees(500)

// Train model. This also runs the indexers.
val model = pipeline.fit(trainingData)

val rfModel = model.stages(2).asInstanceOf[RandomForestClassificationModel]

// Load the model to redis
val f = new Forest(rfModel.trees)
f.loadToRedis("movie-10", "127.0.0.1")
```
Redis time: 0.635129ms, res=3
Spark time: 46.657662ms, res=3.0

-----------------------------
Redis time: 0.644444ms, res=3
Spark time: 49.028983ms, res=3.0

-----------------------------
Classification averages:
redis: 0.9401250000000001 ms
spark: 58.01970206666667 ms
ratio: 61.71488053893542
diffs: 0.0
Step 3: Execute in Redis

- GroupLens
- Redis
- Spark-ML
- Training
- Redis-ML
- +
- Client App
>>> import redis

>>> config = {
    "host": "localhost",
    "port": 6379
}

>>> r = redis.StrictRedis(**config)

>>> user_profile = r.get("user_shay_profile")

>>> print(user_profile)
12:1.0,13:1.0,14:3.0,15:1.0,17:1.0,18:1.0,19:1.0,20:1.0,23:1.0,24:5.0,1.0,115:1.0,116:2.0,117:2.0,119:1.0,120:4.0,121:2.0,122:2.0,
        .......
1360:1.0,1361:1.0,1362:1.0, 1701:6.0,1799:435.0,1801:0.2,1802:0.11,1803:0.04 ,1812:0.04,1813:0.07,1814:0.24,1815:0.09,1816:0.32,1817:0.06

>>> r.execute_command("ML.FOREST.RUN", "movie-10", user_profile)
'3'
Redis CLI Example

>keys *
1) "movie-10"
2) "user_shay_profile"

>ML.FOREST.RUN movie-10

12:1.0,13:1.0,,332:3.0,333:1.0,334:1.0,335:2.0,336:1.0,357:2.0,358:1.0,359:1.0,
362:1.0,367:1.0,368:3.0,369:2.0,404:4.0,405:1.0,406:2.0,407:1.0,408:1.0,409:1.0
........
,410:3.0,411:2.0,412:2.0,423:1.0,454:1.0,455:1.0,456:1.0,457:3.0,458:1.0,459:1.0
0,470:1"
"3"
>
• Train with Spark, Serve with Redis
• 97% resource cost serving
• Simplify ML lifecycle

Redis\textsuperscript{e} (Cloud or Pack):
– Scaling, HA, Performance
– PAYG – cost optimized
– Ease of use
– Supported by the teams who created Redis

Data loaded into Spark

Model is saved in Redis-ML

Serving Client
Resources

- Redis-ML: https://github.com/RedisLabsModules/redis-ml
- Spark-Redis-ML: https://github.com/RedisLabs/spark-redis-ml
- Neural Redis: https://github.com/antirez/neural-redis
Q&A