@

Geospatial Indexing at Scale

The ~~10~~ 15 Million QPS Redis Architecture Powering Lyft

Daniel Hochman, Engineer

redisconf17

Agenda

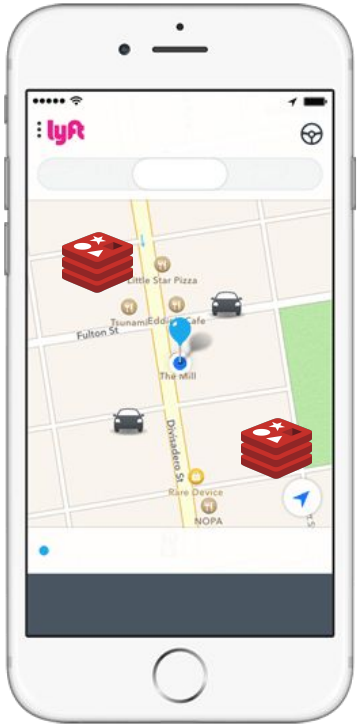
Case Study: scaling a geospatial index

- Original Lyft architecture
- Migrating to Redis
- Iterating on data model

Redis on the Lyft platform

- Service integration
- Operations and monitoring
- Capacity planning
- Open source work and roadmap

Q&A



Lyft backend in 2012

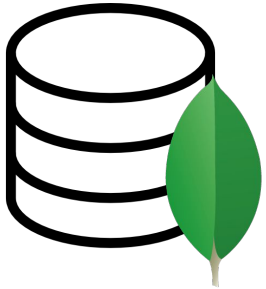
5 sec loop

```
PUT /v1/locations
  {
    "lat": 37.61,
    "lng": -122.38
  }
```

```
200 OK
  {
    "drivers": [ ... ],
    "primetime": 0,
    "eta": 30,
    ...
  }
```

API Monolith

Location stored on user record



Index on users collection

(driver_mode, region)

Monolithic architecture issues

```
drivers_in_region = db.users.find(
    driver_mode: {$eq: True},
    region: {$eq: "SFO"},
    ride_id: {$eq: None}
)

eligible_drivers = sort_and_filter_by_distance(
    drivers_in_region, radius=.5
)
dispatch(eligible_drivers[0])
```

⚠️ Global write lock, not shardable, difficult refactoring, region boundary issues

Unwritten architecture rule at Lyft

Horizontally scalable

and highly available

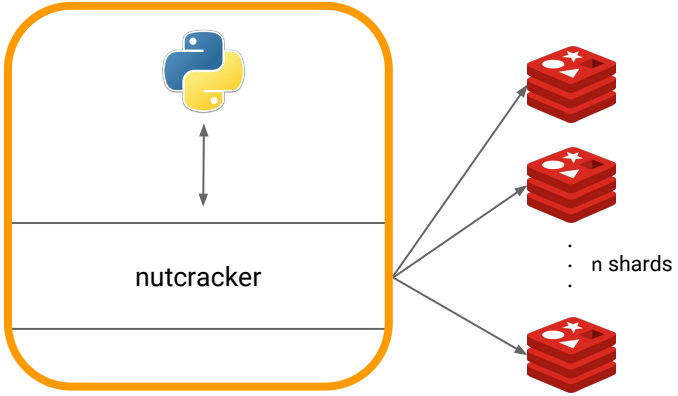
from day zero

Nutcracker overview

twitter / **twemproxy**

A fast, light-weight proxy for memcached and redis

```
locations_cluster:
  listen: locations.sock
  distribution: ketama
  hash: md5
  eject_hosts: true
  failure_limit: 3
  servers:
    - 10.0.0.1:6379:1
    - 10.0.0.2:6379:1
    ...
    - 10.0.0.255:6379:1
```



Ketama provides consistent hashing!

Lose a node, only lose 1/n data

Pipelining

```
PIPELINE (
  HGETALL foo
  SET hello world
  INCR lyfts
)
```

```
md5(foo)   % 3 = 2
md5(hello) % 3 = 0
md5(lyfts) % 3 = 2
```

nutcracker

```
0: SET hello world
```

0

```
2: PIPELINE (
     HGETALL foo
     INCR lyfts
   )
```

1

2

```
return ordered_results
```

```
RESPONSE (
  (k1, v1, k2, v2)
  OK
  12121986
)
```

1. hash the keys
2. send concurrent requests to backends
3. concatenate and return results

Parity data model

```
location = json.dumps({'lat': 23.2, 'lng': -122.3, 'ride_id': None})

with nutcracker.pipeline() as pipeline:
    pipeline.set(user_id, location)                    # string per user
    if driver_mode is True:
        pipeline.hset(region_name, user_id, location)  # hash per region
```

⚠️ Fetching inactive drivers when doing `HGETALL`. Network and serialization overhead. The obvious fix? Expiration.

Hash expiration

Implement Expire on hash #167

antirez commented on Jan 16, 2012 Owner

Hi, this will not be implemented by original design.

The reasoning is ... complex and is a lot biased by personal feelings, preference and sensibility ... ;)

Closing. Thanks for reporting.

antirez closed this on Jan 16, 2012

Expiring a hash

```
- pipeline.hset(region_name, user_id, location)  # hash per region
```

```
  # hash per region per 30 seconds
  bucket = now_seconds() - (now_seconds() % 30)
  hash_key = '{}_{}'.format(region_name, bucket)
  pipeline.hset(hash_key, user_id, location)
  pipeline.expire(hash_key, 15)
```

12:00:00	12:00:30	12:01:00	12:01:30	...

⚠️ `HGETALL` current bucket plus next and previous to handle clock drift and boundary condition, merge in process.

Growing pains

Region is a poor hash key (hot shards)

Bulk expiration blocks Redis for longer than expected

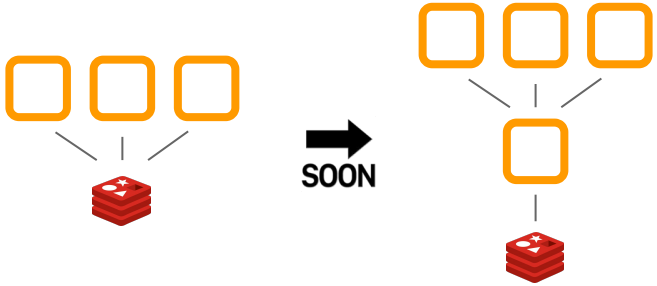
Redis used for inter-service communication with new dispatch system

Let's fix it!

Proper service communication

Redis is used for inter-service communication

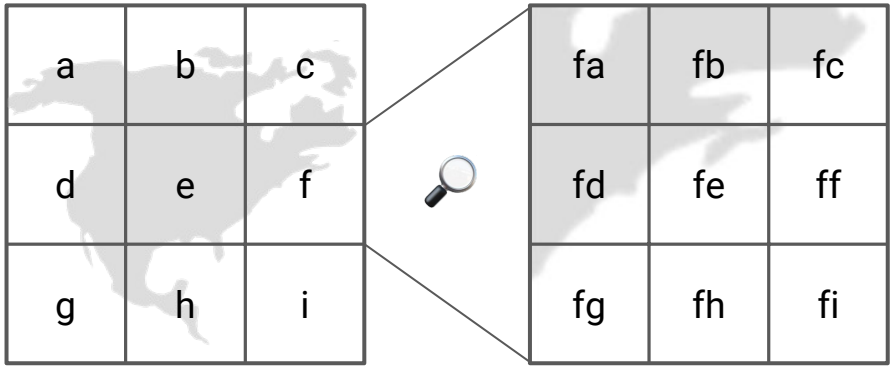
- Replicate existing queries and writes in new service
- Replace function calls that query or write to Redis with calls to new service
- With contract in place between existing services and new service, refactor data model
- Migrate relevant business logic



Geohashing

Region is a poor hash key

Geohashing is an algorithm that provides arbitrary precision with gradual precision degradation




```
>>> compute_geohash(lat=37.7852, lng=-122.4044, level=9)
9q8yywefd
```

Google open-sourced an alternative geohashing algorithm, S2

Data model with geohashing

```
loc = {'lat': 23.2, 'lng': -122.3, 'ride_id': None}
geohash = compute_geohash(loc['lat'], loc['lng'], level=5)

with nutcracker.pipeline() as pipeline:
    # string per user
    pipeline.set(user_id, json.dumps(loc))

    # sorted set per geohash with last timestamp
    if driver_mode is True:
        pipeline.zset(geohash, user_id, now_seconds())
        pipeline.zremrangebyscore(geohash, -inf, now_seconds() - 30)  # expire!
```

⚠️ Sorted set tells you where a driver might be. Use string as source of truth.

On query, look in neighboring geohashes based on desired radius.

Why not GEO?

Stable release in May 2016 with Redis 3.2.0

Point-in-radius, position of key

Uses geohashing and a sorted set under-the-hood

No expiration or sharding

No metadata storage

Great for prototyping

Much more data model complexity behind the scenes

- Additional metadata
- Writing to multiple indices to lower cost of high frequency queries
- Balancing scattered gets and hot shards with geohash level

```
GEOADD
GEODIST
GEOHASH
GEOPOS
GEORADIUS
GEORADIUSBYMEMBER
```

Redis on the Lyft platform

Cluster creation

Creating a new cluster is a self-service process that takes less than one engineer hour.

2015: 1 cluster of 3 instances

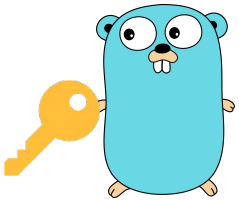
2017: 50 clusters with a total of 750 instances

Internal libraries

Golang and Python are the two officially supported backend languages at Lyft

Python features

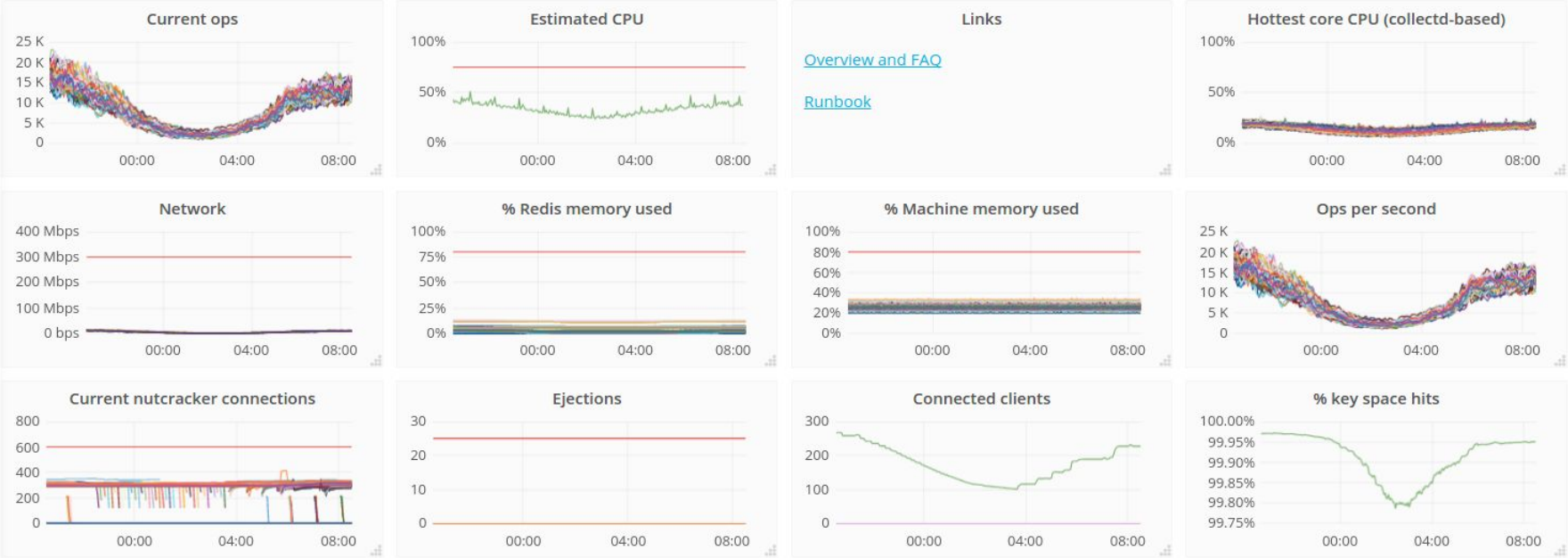
- Fully compatible with redis-py StrictRedis
- Stats
- Retry
- Pipeline management for interleave and targeted retry



```
from lyftredis import NutcrackerClient
redis_client = NutcrackerClient('locations')
```

Observability

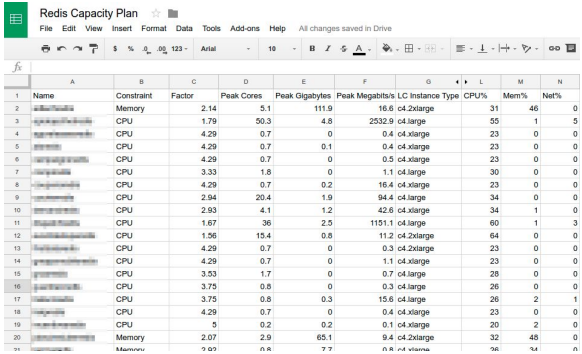
```
{% macro redis_cluster_stats(redis_cluster_name, alarm_thresholds) -%}
```



Capacity planning

Combine APIs and stats for global capacity plan

- Difficult to track 50 clusters
- Google Sheets API for display
- Internal stats for actual usage, EC2 API for capacity
- Automatically determine resource constraint (CPU, memory, network)
- Currently aim for 4x headroom due to difficulty of cluster resize
- At-a-glance view of peak resource consumption, provisioned resources, cost, resource constraint



Redis Capacity Plan

File Edit View Insert Format Data Tools Add-ons Help All changes saved in Drive

	A	B	C	D	E	F	G	L	M	N
1	Name	Constraint	Factor	Peak Cores	Peak Gigabytes	Peak Megabits/s	LC Instance Type	CPU%	Mem%	Net%
2	[illegible]	Memory	2.14	5.1	111.9	16.6	c4.2xlarge	31	46	0
3	[illegible]	CPU	1.79	50.3	4.8	2532.9	c4.large	55	1	5
4	[illegible]	CPU	4.29	0.7	0	0.4	c4.xlarge	23	0	0
5	[illegible]	CPU	4.29	0.7	0.1	0.4	c4.xlarge	23	0	0
6	[illegible]	CPU	4.29	0.7	0	0.5	c4.xlarge	23	0	0
7	[illegible]	CPU	3.33	1.8	0	1.1	c4.large	30	0	0
8	[illegible]	CPU	4.29	0.7	0.2	16.4	c4.xlarge	23	0	0
9	[illegible]	CPU	2.94	20.4	1.9	94.4	c4.large	34	0	0
10	[illegible]	CPU	2.93	4.1	1.2	42.6	c4.xlarge	34	1	0
11	[illegible]	CPU	1.67	36	2.5	1151.1	c4.large	60	1	3
12	[illegible]	CPU	1.56	15.4	0.8	11.2	c4.2xlarge	64	0	0
13	[illegible]	CPU	4.29	0.7	0	0.3	c4.2xlarge	23	0	0
14	[illegible]	CPU	4.29	0.7	0	1.1	c4.xlarge	23	0	0
15	[illegible]	CPU	3.53	1.7	0	0.7	c4.large	28	0	0
16	[illegible]	CPU	3.75	0.8	0	0.3	c4.large	26	0	0
17	[illegible]	CPU	3.75	0.8	0.3	15.6	c4.large	26	2	1
18	[illegible]	CPU	4.29	0.7	0	0.4	c4.xlarge	23	0	0
19	[illegible]	CPU	5	0.2	0.2	0.1	c4.xlarge	20	2	0
20	[illegible]	Memory	2.07	2.9	65.1	9.4	c4.2xlarge	32	48	0
21	[illegible]	Memory	2.82	0.8	7.7	0.8	c4.xlarge	26	34	0

Object serialization

Benefits

- Lower memory consumption, I/O
- Lower network I/O
- Lower serialization cost to CPU

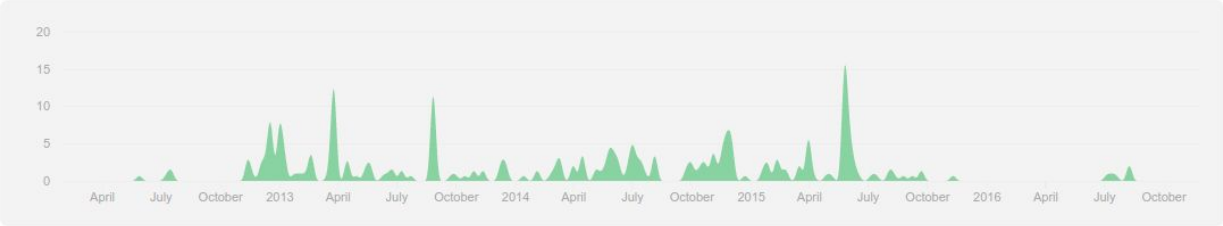
JSON	MessagePack	protobuf Protocol Buffers
1012 bytes	708 bytes	190 bytes
(original)	69%	18%

The road ahead

Nutcracker issues

- Deprecated internally at Twitter and unmaintained
- Passive health checks
- No hot restart (config changes cause downtime)
- Difficult to extend (e.g. to integrate with service discovery)
- When EC2 instance of Redis dies, we page an engineer to fix the problem 😭

twitter / **twemproxy**



What is Envoy?

Open-source C++11 service mesh and edge proxy

As an advanced load balancer, provides:

- Service discovery integration
- Retry, circuit breaking, rate limiting
- Consistent hashing
- Active health checks
- Stats, stats, stats
- Tracing
- Outlier detection, fault injection

Envoy was designed to be extensible!

envoy by lyft

Introducing Envoy Redis

In production at Lyft as of May 2017

Support for `INCR`, `INCRBY`, `SET`, `GET` (ratelimit service)

Service discovery integration (autoscaling!)

Active healthcheck using `PING` (self-healing!)

Pipeline splitting, concurrency

Basic stats

- **redis: PONG is a simple string** ✓
  #933 by mattklein123 was merged 19 days ago • Approved
- **redis: add active health check support** ✓
  #899 by mattklein123 was merged 21 days ago • Approved
- **redis: support case insensitive commands** ✓
  #844 by mattklein123 was merged on Apr 27 • Approved
- **redis: op timeouts** ✓
  #693 by mattklein123 was merged on Apr 6 • Approved
- **redis: downstream and splitter stats** ✓
  #656 by mattklein123 was merged on Mar 31 • Approved
- **redis: conn pool stats and connect timeouts** ✓
  #652 by mattklein123 was merged on Mar 30 • Approved
- **redis: command splitting** ✓
  #616 by mattklein123 was merged on Mar 27 • Approved
- **redis: hashing connection pool implementation** ✓
  #523 by mattklein123 was merged on Mar 2 • Approved
- **HTTP consistent hash routing**
  #496 by mattklein123 was merged on Feb 22 • Approved
- **redis: initial codec and proxy support** ✓
  #309 by mattklein123 was merged on Jan 9 • Approved

Envoy Redis Roadmap

Additional command support

Error handling

Pipeline management features

Performance optimization

Replication

- Failover
- Mitigate hot read shard issues with large objects
- Zone local query routing
- Quorum
- Protocol overloading? e.g. `SET key value [replication factor] [timeout]`

More!

Q&A

- Thanks!
- Email technical inquiries to dhochman@lyft.com
- Participate in Envoy open source community! lyft/envoy
- Lyft is **hiring**. If you want to work on scaling problems in a fast-moving, high-growth company visit https://www.lyft.com/jobs