Scalable Search With Redis

Dvir Volk

Señor System Architect, Redis Labs
March 2016

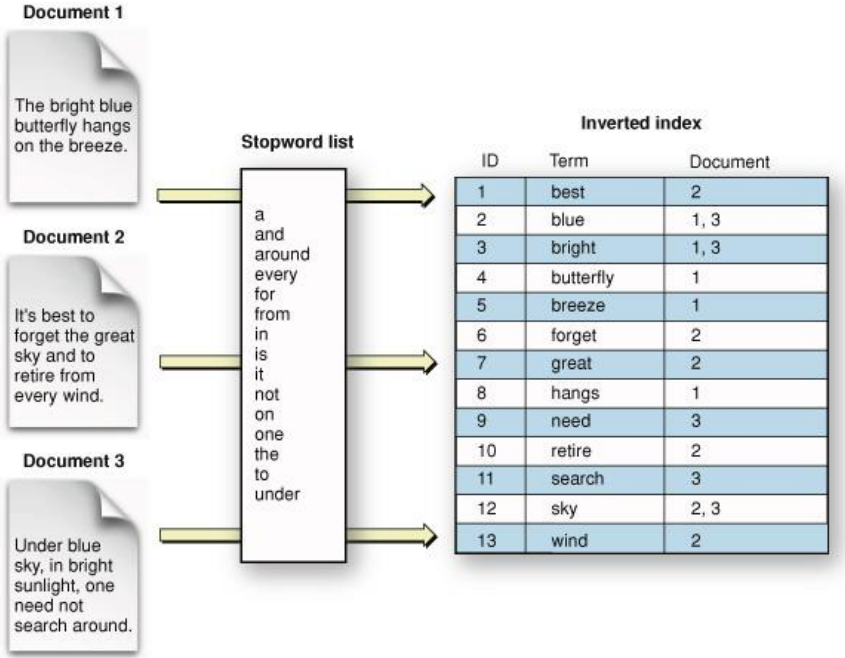
Search 101 - The Basic Problem

- We have a collection of entities (Documents)
- We need to find a set of words inside them
- The collection is too large to grep :)
- We might want to filter by other properties
- So we need to create an index

Search 101 - Inverted Index

- Take a bunch of documents
- Break down to tokens
- Weight word significance
- Apply Secret Sauce™
- Save to an index:
  word ⇒ {doc1, doc2, … }

- World Domination Achieved!



Classic Search on Redis

- Traditional approach - **SORTED SET**
  - **Key**: word
  - **Value**: doc Id
  - **Score**: *TF * doc_score*

ft:hello {4567}	
doc_1	0.5
doc_5	0.675
doc_70	0.1
doc_1337	0.1337
...	...

ft:world {2354}	
doc_1	0.5
doc_6	0.545
doc_70	0.789
doc_2448	0.246
...	...

Classic Search on Redis

- **hello AND world** ⇒ ZINTERSTORE
- **hello OR world** ⇒ ZUNIONSTORE

- Composable to complex queries
- Supports document similarity
- Has its limitations...

ft:hello {4567}	
doc_1	0.5
doc_5	0.675
doc_70	0.1
doc_1337	0.1337
...	...

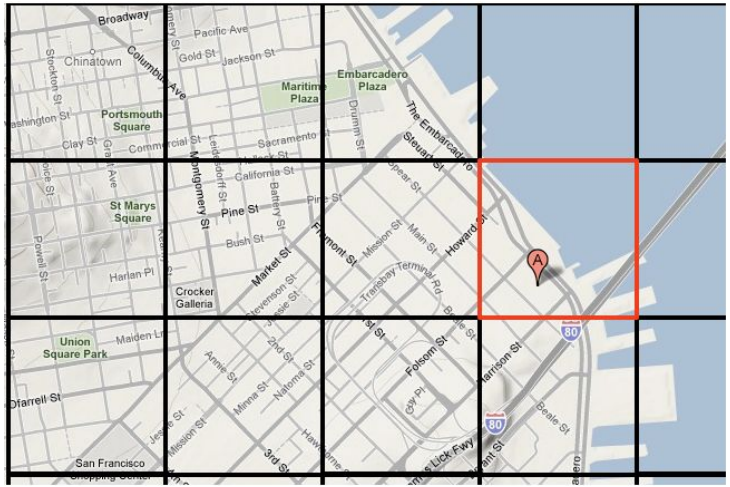
ft:world {2354}	
doc_1	0.5
doc_6	0.545
doc_70	0.789
doc_2448	0.246
...	...

ZINTERSTORE “hello world” 2 hello world

tmp:hello & world	
doc_1	1.0
doc_70	0.889

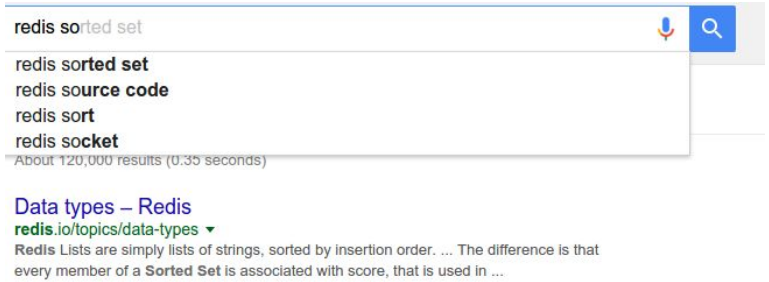
Adding Geo Search

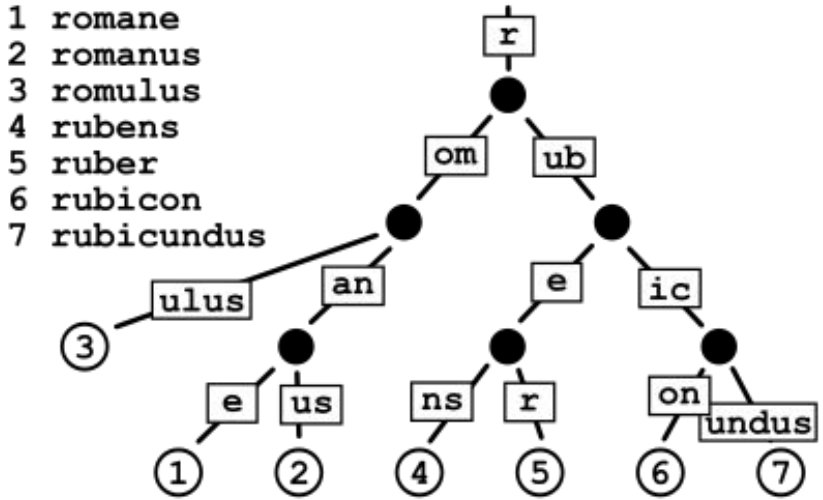
- e.g. “Pizza near me”
- Redis 3.2 adds GEO commands
- Infer “near me” => 1km radius of lat,long
- Intersect GEO and Ordinary sorted sets



Problem 2: Auto-suggest

- Complete By Prefix
- Instant Search
- Usually users TRIE or similar





Auto-Suggest on Redis

- No trie in Redis :(
- Uses Sorted Sets in a different way
- If scores are the sames, entries are sorted lexically
- This means O(Log(n)) behavior

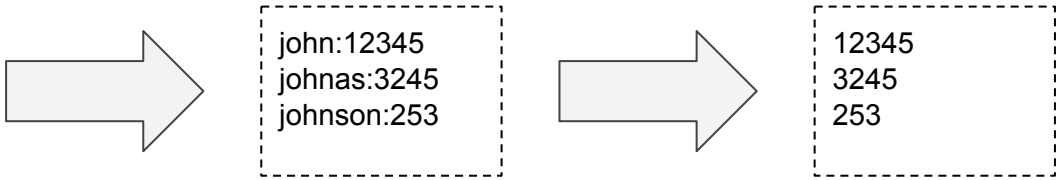
- **ZLRANGEBYLEX key min max**

users:name	
john	0
johnas	0
johnson	0
jonnie	0
jonson	0
...	...

ZRANGEBYLEX In Action

ZRANGEBYLEX users:name (joh [joh\xff

users:name	
johnn:12345	0
johnas:3245	0
johnson:253	0
jonnie:45363	0
jonson:2342	0
...	...



PS Scoring is encoded lexicographically too!

Range and multi property queries with LEX

- Encode non-string values into sorted sets
- Big endian binary for numbers
- Combine multiple elements in a single entry

IN SQLish:

```
"SELECT * FROM USERS
WHERE age='30'
AND SALARY='20000'
AND NAME LIKE 'john%'"
```

In Redish:

```
"ZRANGEBYLEX users:age|salary|name
(0000001E:00004E20:john
(0000001E:00004E20:john\xff"
```

Scaling this mess

- So you have an index per word
- Or one long ZSET per indexed property

- What happens when you exceed the RAM?
- What happens when intersections take too long?

Option 1: Scale UP

- **MOAR RAM PLZ!!!!**

- Hello Very Expensive Machines
- Good luck with replication!
- Don’t forget 25% headroom

Scale Out

- **MOAR MASHINEZ PLZ!!!**

- Welcome to DS Hell™
- What about very large keys?
- What about intersection?
- How do you balance this?

The Solution - Index Partitioning

- On top of the cluster sharding
- Partition by docId
- Each partition has everything for a subset of the docs
- Use your own partitioning function ⇒
  - **CRC32(docId) % num_partitions**
- Use {} key notation ⇒
  - **users:name{1}**
- Query in parallel
- Take top N results from each partition and merge!

Index Partitioning