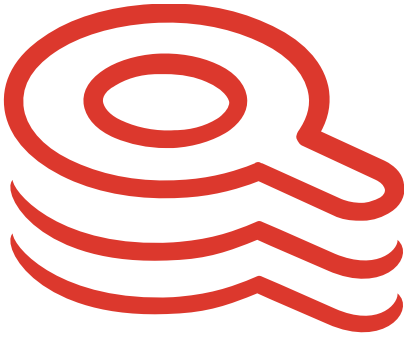
!

RediSearch Demystified



Search Engines

Rank			DBMS	Database Model	Score		
May 2019	Apr 2019	May 2018			May 2019	Apr 2019	May 2018
1.	1.	1.	Oracle	Relational, Multi-model	1285.55	+5.61	-4.87
2.	2.	2.	MySQL	Relational, Multi-model	1218.96	+3.82	-4.38
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	1072.19	+12.23	-13.66
4.	4.	4.	PostgreSQL	Relational, Multi-model	478.89	+0.17	+77.99
5.	5.	5.	MongoDB	Document	408.07	+6.10	+65.96
6.	6.	6.	IBM Db2	Relational, Multi-model	174.44	-1.61	-11.17
7.	↑ 8.	↑ 9.	Elasticsearch	Search engine, Multi-model	148.62	+2.62	+18.18
8.	↓ 7.	↓ 7.	Redis	Key-value, Multi-model	148.40	+2.03	+13.06
9.	9.	↓ 8.	Microsoft Access	Relational	143.78	-0.87	+10.67
10.	↑ 11.	10.	Cassandra	Wide column	125.72	+2.11	+7.89
11.	↓ 10.	11.	SQLite	Relational	122.90	-1.32	+7.44
12.	12.	↑ 14.	MariaDB	Relational, Multi-model	86.52	+1.29	+21.53
13.	13.	13.	Splunk	Search engine	85.24	+2.15	+20.15
14.	↑ 15.	↑ 18.	Hive	Relational	77.90	+3.19	+20.93
15.	↓ 14.	↓ 12.	Teradata	Relational	76.04	+0.69	+1.63
16.	16.	↓ 15.	Solr	Search engine	60.80	+0.57	-0.72

!

!

Y U NO

SEARCH ENGINE

Search Engine

- software that builds **indexes** on **documents**
- and **answers queries** using those indexes

Indexing

Documents

- a document is a collection of fields
- full text, tag, numeric, geo

Inverted Index

maps keywords to docs

- Inverted index: instead of mapping docs to keywords, it maps keywords to docs
- List of all the docs a word appears in
- as well as term frequency
- and offsets where the term appeared in the doc
- Offsets are used for "exact match" type searches, or for ranking of results

Tokenization

- Whitespace
  `list of words` → `list` | `of` | `words`
- Punctuation
  `foo-bar.baz···bag` → `[foo, bar, baz, bag]`

Stop words

Extremely common words

a	is	the	an	and	are
as	at	be	but	by	for
if	in	into	it	no	not
of	on	or	such	that	their
then	there	these	they	this	to
was	will	with			

Stop Words

```
This is a list of words
```

↓

```
list words
```

- words that are usually so common that they do not add much information to search, but take up a lot of space and CPU time in the index
- stop-words are ignored both during indexing and searching
- Stop-words for an index can be defined (or disabled completely) on index creation using the STOPWORDS argument in the FT.CREATE command

Stemming

Reduce a word to its simplest form

```
running
```

↓

```
run
```

English Stemmer

- `am, are, is` → `be`
- `abode, abided, abidden` → `abide`
- `cat, cats, cat's, cats'` → `cat`

Romance Stemmer

		Fr	Spa	Por	Ita
noun	ANCE	ance	anza	eza	anza
adjective	IC	ique	ico	ico	ico
noun	ATION	ation	ación	ação	azione
adjective	ABLE	able	able	ável	abile

Synonyms

- `{boy, child, baby}`
- `{girl, child, baby}`
- `{man, person, adult}`

> - Search for 'child' and receive documents contains 'boy', 'girl', 'child' and 'baby'.
> - RediSearch uses a simple HashMap to map between the terms and the group ids. During building the index, we check if the current term appears in the synonym map, and if it does we take all the group ids that the term belongs to.

Tag Fields

Similar to full-text fields but more compact

> - no stemming
> - simpler tokenization
> - cannot be found from general full-text search
> - index resides in single Redis key, not key per term
> - index is simpler and more compact
> - no freqs, offsets, field flags

!

[background] | *spaceballs_we_aint_found.gif*

Query Language

- Multi-word phrases: `foo bar baz`
- Exact phrases: `"hello world"`
- Prefix: `hel*`
- Or (union): `hello|hallo|shalom|hola`
- Negation: `hello -world`

Query Language

- Specific fields: `@field:hello world`
- Numeric range: `@field:[1 10]`
- Geo-radius: `@field:[-77 39 5 km]`
- Tags: `@field:{tag1 | tag2}`
- Optional: `~bar`

Query Execution

based on chained iterators

!

```
hello
```

↓

```
read("hello")
```

!

```
hello world
```

↓

```
intersect(
    read("hello"),
    read("world")
)
```

!

```
"hello world"
```

↓

```
exact_intersect(
    read("hello"),
    read("world")
)
```

!

```
"hello word" foo
```

↓

```
intersect(
    exact_intersect(
        read("hello"),
        read("world")
    ),
    read("foo")
)
```

Fuzzy Matching

```
%%Hamberders%%
```

↓

```
Hamburgers
```

- Dictionary of all terms in the index can also be used to perform Fuzzy Matching. Fuzzy matches are performed based on Levenshtein distance (LD). Fuzzy matching on a term is performed by surrounding the term with '%'

Covfefe?

Phonetic Matching

!

- How can we help a non-native speaker or a 5-year old?

AIHEOPDERF

- `AI` → `I`
- `HEOP` → `help`

- `D` → `the`
- `ERF` → `earth`

Double Metaphone

- primarily designed for American English names
- also encodes most English words well
- **double** encoding for a given word
  - likely pronunciation
  - optional alternative pronunciation

Double Metaphone

- John → `JN`
- Jon → `JN`
- Jawn → `JN`

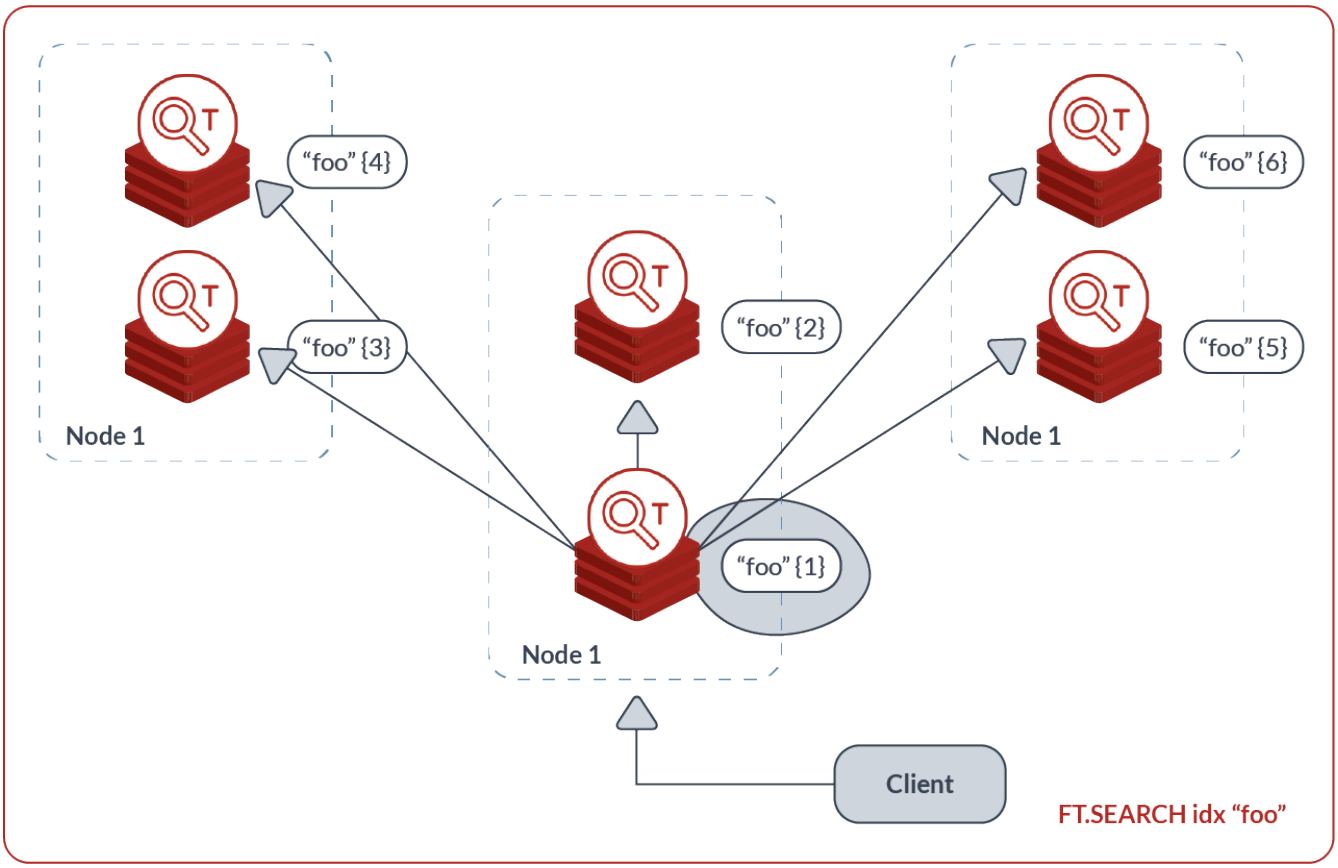
Index Partitioning

- index split across many partitions by document ID
- a partition has complete index of all its documents
- query partitions concurrently and merge results
- … need **search coordinator**

> - While RediSearch is very fast and memory efficient, if an index is big enough, at some point it will be too slow or consume too much memory for a single machine. Then it will have to be scaled out and partitioned over several machines, each of which will hold a small part of the complete search index.
> - Traditional clusters map different keys to different “shards” to achieve this. However, in search indexes this approach is not practical. If we mapped each word’s index to a different shard, we would end up needing to intersect records from different servers for multi-term queries.
> - The way to address this challenge is to employ a technique called index partitioning, which is very simple at its core.

!

!

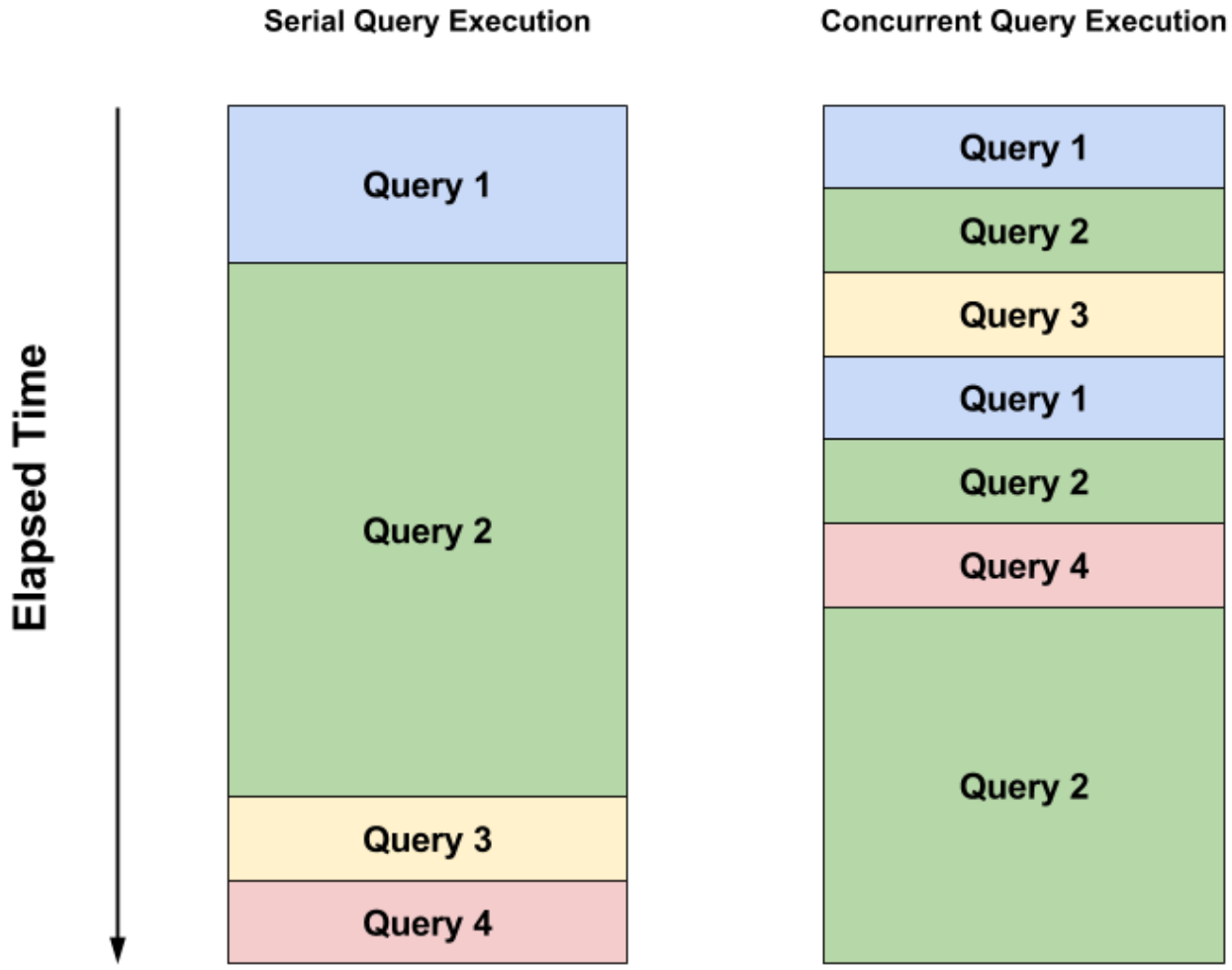


- To enable that, a new component called a "Coordinator" is added to the cluster. When searching for documents, the Coordinator receives the query and sends it to N partitions, each holding a sub index of 1/N documents. Since we're only interested in the top K results of all partitions, each partition returns just its own top K results. We then merge the N lists of K elements and extract the top K elements from the merged list.

Concurrency

- RediSearch is very fast
- … but queries with big datasets can take seconds
- even with index partitioning it can still be too slow
- How to avoid blocking Redis servers for a while?
- Modules have Global Lock & Thread Safe Contexts

!


Serial Query Execution
Concurrent Query Execution
Elapsed Time
Query 1
Query 2
Query 3
Query 4
Query 1
Query 2
Query 3
Query 1
Query 2
Query 4
Query 2

- OS scheduler ensures all queries get CPU time
- While a query is running the rest wait idly
- Execution is yielded 5,000 times/sec
- Fast queries finish in one go
- Slow ones will take many iterations
- Allows queries to run **concurrently**
- Same approach for indexing big documents
- RediSearch has a thread pool for running concurrent search queries.
- When a search request arrives, it gets to the handler, gets parsed on the main thread, and a request object is passed to the thread pool via a queue.
- The thread pool runs a query processing function in its own thread.
- The function locks the Redis Global lock, and starts executing the query.
- Since the search execution is basically an iterator running in a cycle, we simply sample the elapsed time every several iterations (sampling on each iteration would slow things down as it has a cost of its own).
- If enough time has elapsed, the query processor releases the Global Lock, and immediately tries to acquire it again. When the lock is released, the kernel will schedule another thread to run - be it Redis' main thread, or another query thread.
- When the lock is acquired again - we reopen all Redis resources we were holding before releasing the lock (keys might have been deleted while the thread has been "sleeping"), and continue work from the previous state.

!

May the search be with you

…ALWAYS