

Combining inverted and ANN indexes for scale

Anubhav Bindlish
Rockset




About me



- Software Engineer at Rockset (2021-Present)
 - We are a **real time search and analytics platform** based on Rocksdb
 - Have worked on data indexing and query execution
 - Vector search integration using FAISS (IVF)
- Software Engineer at Facebook (2016 - 2021)
 - Team built a rule engine which decided “Is this action allowed”
 - Written in Haskell
 - Clear transition from heuristic -> ML based rules in the 5 years I was here
 - Trend likely going to intensify in the industry with the advent of LLMs

Contents

1. Converged Index
2. How to add an ANN index to this?
 - a. Manage **memory** required to hold vector data and indexed structures
 - i. Choose indexing approach wisely
 - ii. Inverted indexes on flat files
 - iii. Single-stage metadata filtering + CBO 
 - b. How to **distribute an ANN graph** across multiple shards and avoid expensive reindexing
 - c. How to **update vector embeddings** or metadata quickly
 - d. How to **avoid contention** between heavy indexing and vector search

Inverted indexes for text search

1. Map **terms=>docId**
2. Store posting lists per **term**

Doc 1: Give a man a program frustrate him for a day

Doc 2: Teach a man to program frustrate him for a lifetime

term	posting lists
give	1
a	1,2
man	1,2
program	1,2
frustrate	1,2
him	1,2
for	1,2
day	1
teach	2
to	2
lifetime	2

What we built at Rockset- Converged Index

Index the value of fields in an **Inverted Index, Column Store and Row Store**.

```
<doc 0>
{
  "name": "Igor"
}

<doc 1>
{
  "name": "Dhruba"
}
```



Key	Value	
R.0.name	Igor	Row Store
R.1.name	Dhruba	
C.name.0	Igor	Column Store
C.name.1	Dhruba	
S.name.Dhruba.1		Search index
S.name.Igor.0		

```
SELECT keyword, count(*)
FROM search_logs
WHERE keyword='chair'
ORDER BY count(*) DESC
```

Search Index

(for highly selective queries)

```
SELECT keyword, count(*)
FROM search_logs
GROUP BY keyword
ORDER BY count(*) DESC
```

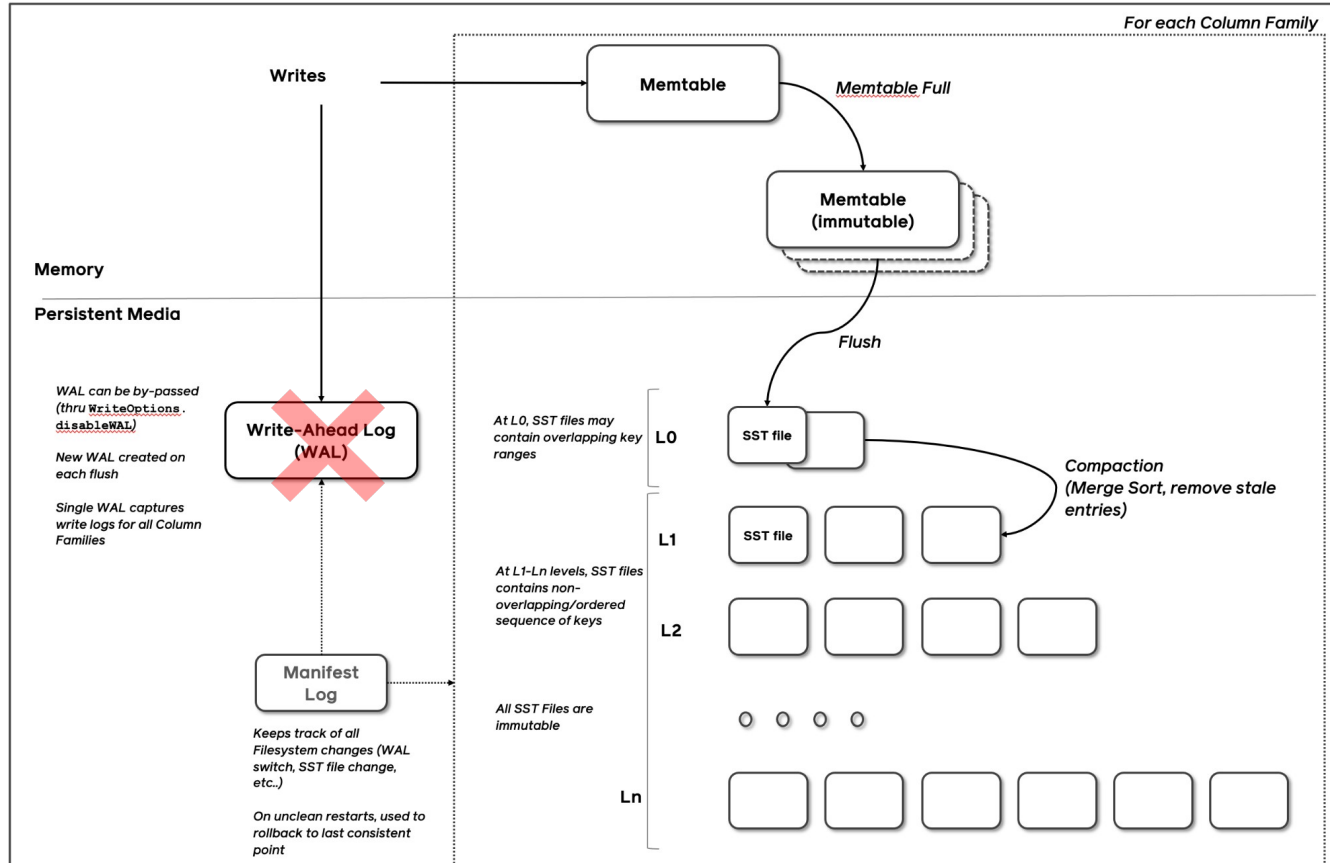
Column Store

(for large scans)

Storage engine

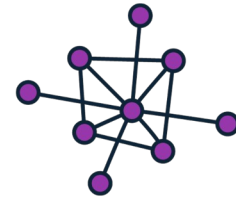
- Rockset uses RocksDB as underlying storage
 - Key-Value store. LSM architecture. Great for ingest throughput.
- Multiple records accumulate in memory and are written into a single SST file
 - Maximise throughput
- Keys are sorted between SST files via compaction in a background process
- Rockset delivers 20Mb/s at 70ms data latency
 - [\[link\]](#)

Rocksdb writes



Converged index + ANN

Key	Value	
R.0.name	Igor	Row Store
R.1.name	Dhruba	
C.name.0	Igor	Column Store
C.name.1	Dhruba	
S.name.Dhruba.1		Search index
S.name.Igor.0		



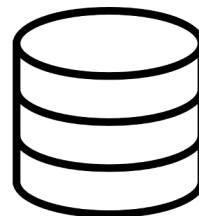
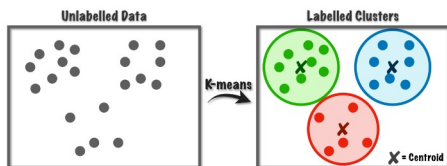
Vectors are a new data type, vector search is just a new query processing approach.

Have a lot in common with already solved database problems.

Databases & ML are not all that different

Lack of a common language

Fragmentation of ideas



Inverted File index for ANN	Search/Inverted Index around since 90s
Metadata Filtering	SQL <code>WHERE</code> predicates
k -nearest neighbors	ORDER BY <i>distance</i> LIMIT k

Why add ANN indexes?

Converged Indexing

- Exact/partial matches (inverted index)
- Filters (inverted index)
- Large scans (column store)
- Joins

ANN Indexes

- Fuzzy/broad search
- Similarity search
- Better user experience, for eg, personal feed

Show me all the restaurants I might like (a vector search) that are located within 10 miles and are low to medium priced (filter)

Find me all images of cat (a vector search) uploaded last week (filter)

Is this user behavior “bad” (vector search) and it is a new user account (filter)



Converged Index



Application

Known design challenges

- **Index building** is expensive
- How to search across **metadata** and vector embeddings
- How to **distribute an ANN graph** across multiple shards and avoid expensive reindexing
- How to **update** vector embeddings and metadata quickly
- How to **avoid contention** between heavy indexing and search

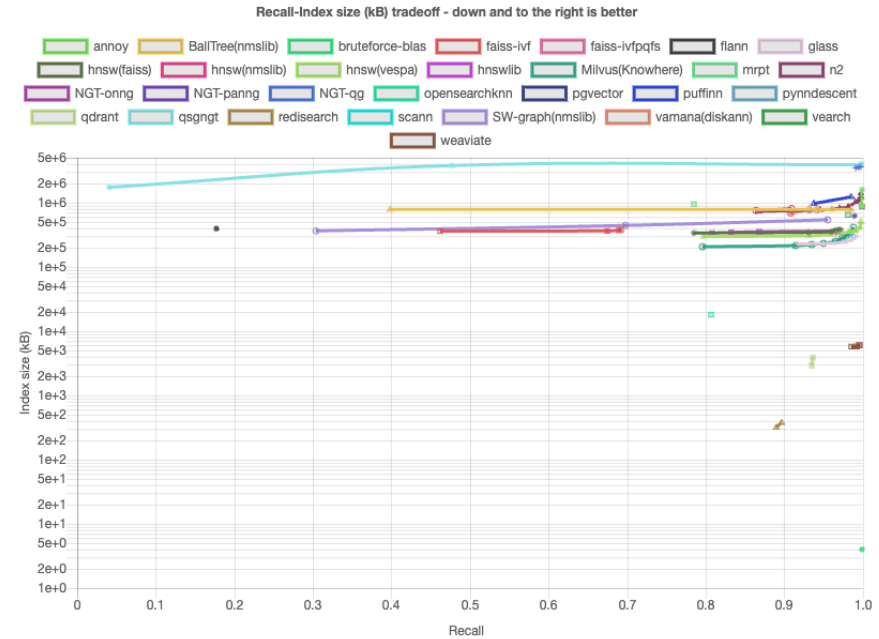
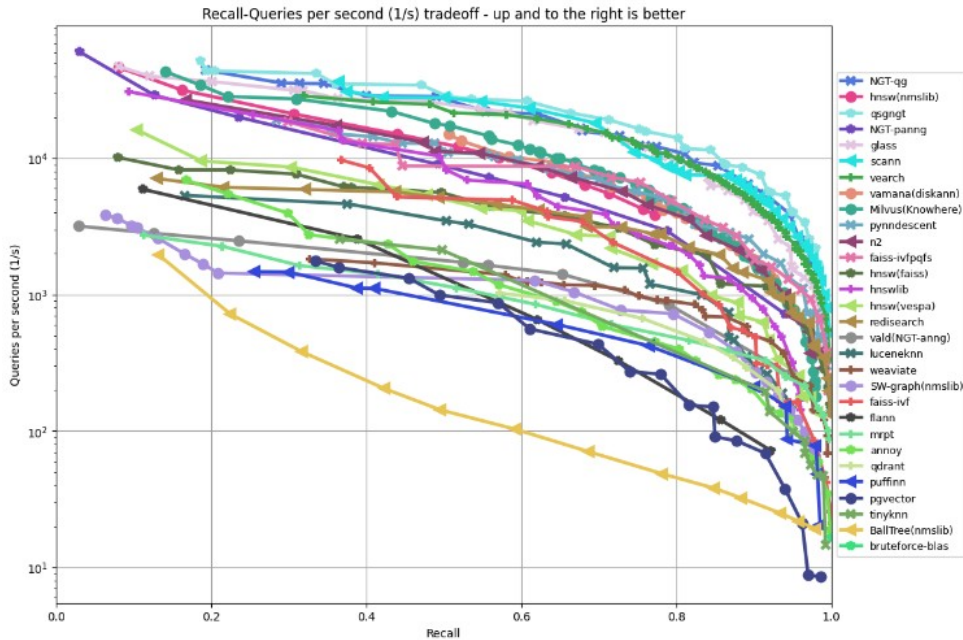
ANN indexes

Find me K vectors closest to my query vector

Note: high dimensional space.



[https://en.wikipedia.org/wiki/
Curse_of_dimensionality#Nearest_neighbor_search](https://en.wikipedia.org/wiki/Curse_of_dimensionality#Nearest_neighbor_search)

Trade offs, Trade offs



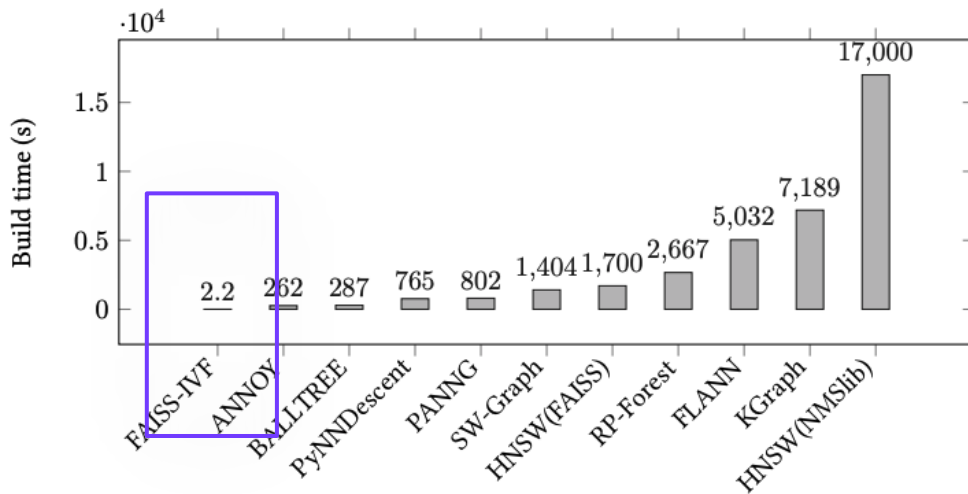
Source: https://ann-benchmarks.com/nytimes-256-angular_10_angular.html

ANN indexes

- **Inverted File (IVF)** 
 - Decent search performance; fast insertions; low memory usage
- **Hierarchical Navigable Small worlds (HNSW)** 
 - Very good search performance; slow insertions; high memory usage
- **FlatIP / FlatL2**
 - Terrible search speeds; only suitable for small datasets
- **Locality Sensitive Hashing (LSH)**
 - Suitable for low dimensional data
- **Composite**
 - Cause why not?

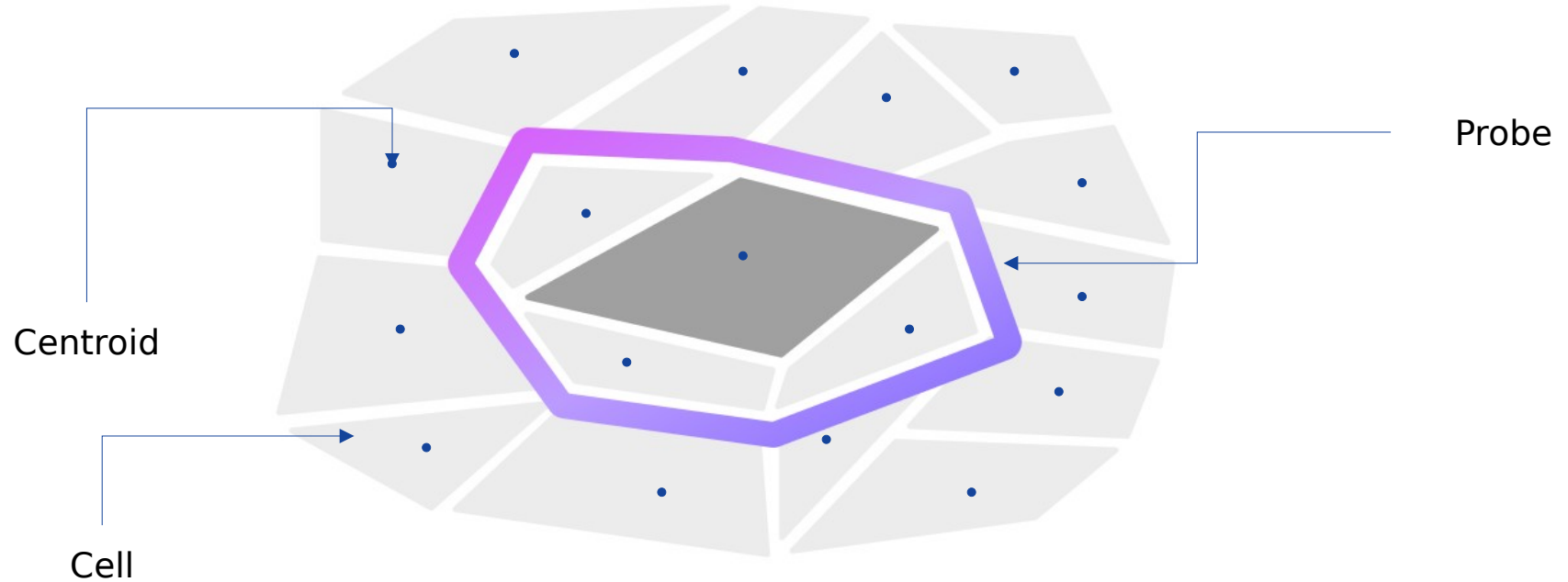
Index build times

- Important if you add new vectors to your database often
- Or update existing vectors (for eg: user's preferences might change for recommendation engine)



■ **Figure 10** Index build time in seconds for dataset GLOVE. The plot shows the minimum build time for an index that achieved recall of at least 0.9 for 10-NN.

How Inverted File Index works



FAISS assigns vectors to Voronoi cells. Each cell is defined by a centroid.

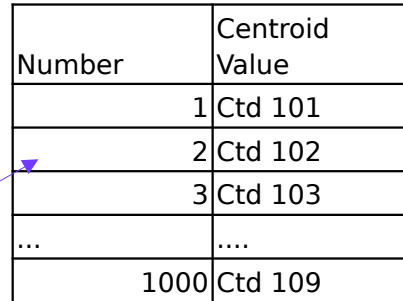
Add IVF index to the Converged Index

1. Create similarity index and store file of centroid identifiers in memory

Create similarity index using configuration (ie: FAISS:IVF:100...FAISS:IVF:1000...). Cluster data in cells by centroid.

FAISS provides 1000 centroids and stores in a small file of cell identifiers in memory.

Small file in
memory



Number	Centroid Value
1	Ctd 101
2	Ctd 102
3	Ctd 103
...
1000	Ctd 109

Add IVF index to the Converged Index

2. Store centroid and residual with each record

Leverage multiple threads in parallel to create the similarity index.

Add two fields to each record: Centroid and Residual

Data may need to be retrained periodically (i.e., recompute centroids)

FAISS returns centroid and residual

Vector embedding passed to FAISS

Centroid	Residual	Name	Age	Location	Vector Embedding
1	10.36	Edwin Jarvis	49	Malibu, California	[.....]
1	4.53	Samantha Morton	46	Los Angeles, California	[.....]
3	2.13	Marvin Adams	23	Everywheresville, United Kingdom	[.....]

**You have the
index
Now what?**



SQL for similarity search

We didn't want to add a DSL for similarity search. So how to express vector similarity in SQL?

Use ORDER BY

For eg; find restaurants I might like in LA

```
SELECT name
FROM restaurants
WHERE location= 'Los Angeles, California'
ORDER BY approx.sim_index(emb,query)
LIMIT 5
```

SQL for similarity search

Under the hood, Rockset's query optimizer asks FAISS for the centroids which are closest to query vector

FAISS returns 3 centroids. Rockset now rewrites the query as:

```
SELECT name
FROM restaurants
WHERE location= 'Los Angeles, California'
WHERE centroid = 2 OR centroid = 4 OR centroid = 1000
LIMIT 5
```

Now we can leverage our inverted index to evaluate this query.

Still have the selectivity estimation problem!

“Give me 5 nearest neighbors where <filter>.”

- What if the centroid for query vector is “dense”
- Different predicates. Different selectivities.
- Reordering? Planning? Optimizing?
- Running filters fast is a [well studied database problem](#).
- [Cost-based query optimizer](#)
- [Build histograms => Run cost models => Select Access Path](#)

Metadata filtering



Pre-filter + Single-stage filter + Selectivity estimation = 🔥

“Give me 5 nearest neighbors where <filter>.”



Filter is not very selective.....

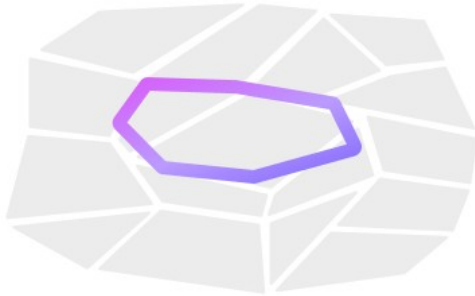
- Post filter
- First get points in cells, then apply filter

Filter is very selective.....

- Pre-filter
- Filter first, and then just scan

Single-stage filtering

Search x closest centroids
(ie: $x = 3$)



Apply filter




Sort Results

NOT to be confused with post or pre filtering.

Filters are run after finding centroids, but before heap-sort.

Single Stage Filtering balances trade offs better.

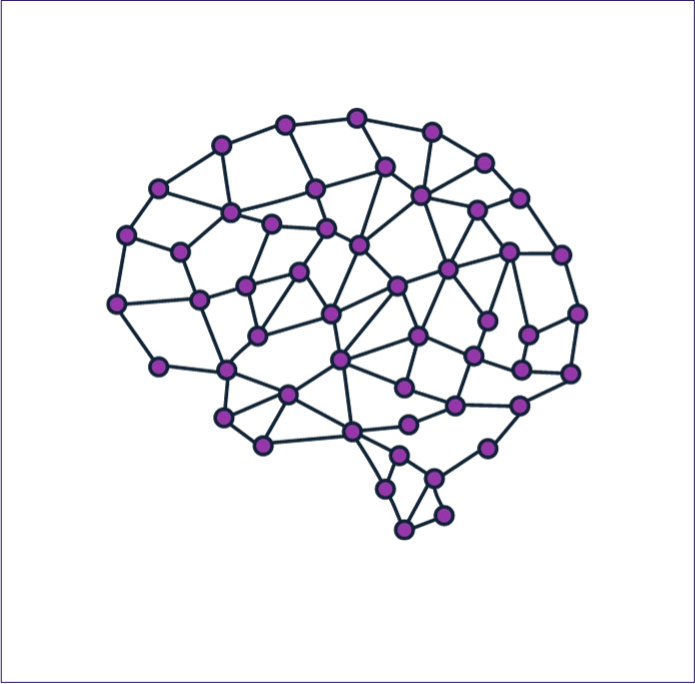
Might consider alternatives for highly selective queries.

Pre-filtering	Great for highly selective queries
Post-filtering	Great for “non-dense” cells
Single-stage filtering	

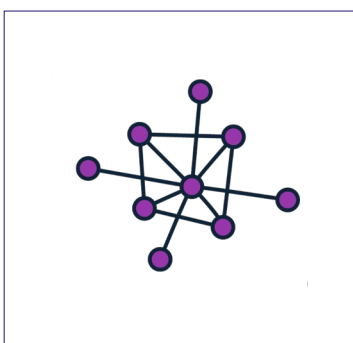
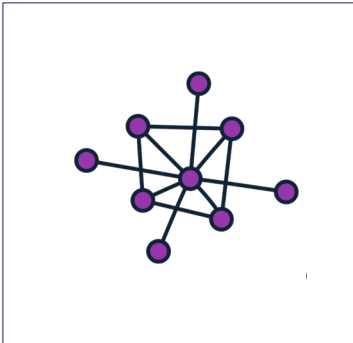
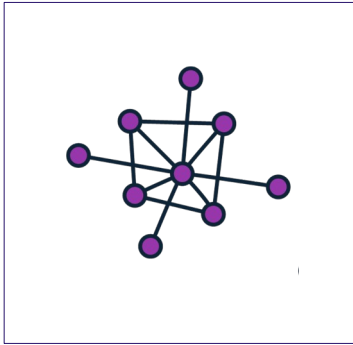
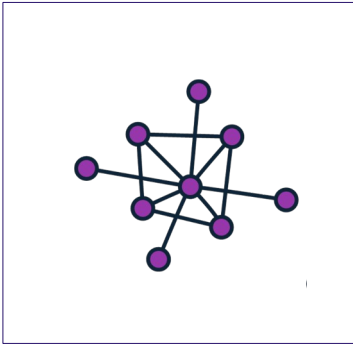
The problem of sharding



To shard or not shard your index



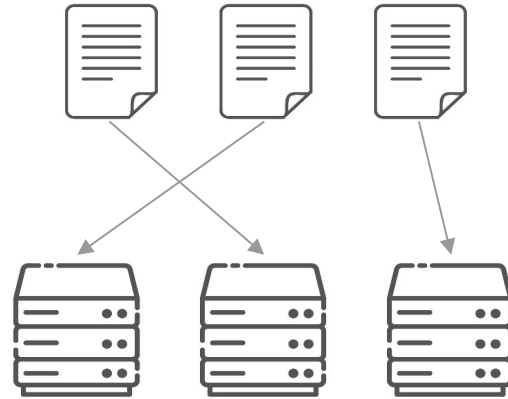
Single machine



Sharded DB

Rockset leverages document sharding for low latency

- A single query is able to parallelize across all CPU on your cluster
- **Microshards** within every shard to allow easier re-sharding
- Every shard has its own **Converged Index**
- Just merge and return results

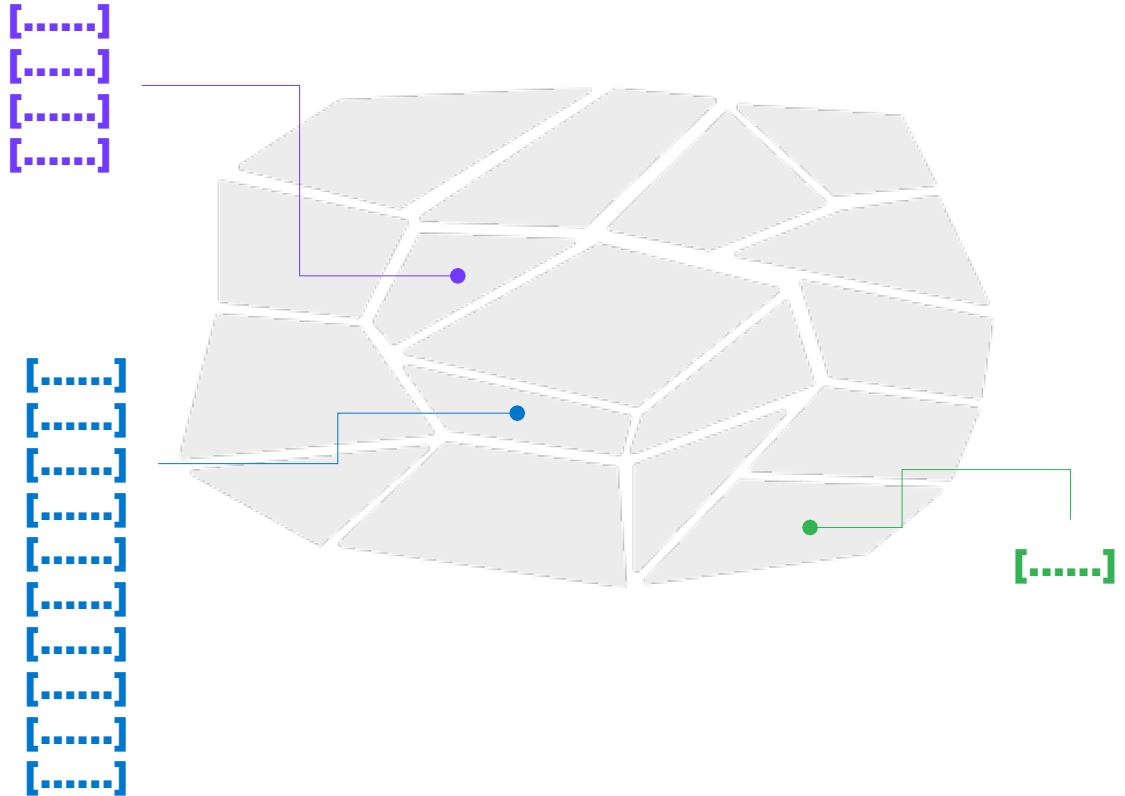


Hard problem: Incremental updates



How do we handle updates?

- Can I add new vectors/update vectors to my database?
- Data latency! New vectors need to be queryable
- Naively inserting has speed + accuracy
 - Balanced-binary-search tree analogy
- Also, mutation, deletion.



How Rockset solves this: Update, update, update.....reindex

Rockset is mutable at the individual field level:

1. You update the vector embedding
2. Query FAISS to generate the centroid and residual
3. Rockset updates the centroid and residual values for the record
4. This means new records are queryable within ~100ms

Still need to retrain index periodically to keep recall high.

! You update this value

Centroid	Residual	Name	Age	Location	Vector Embedding
1	10.36	Edwin Jarvis	49	Malibu, California	[.....]
1	4.53	Samantha Morton	46	Los Angeles, California	[.....]
3	2.13	Marvin Adams	23	Everywhersville, United Kingdom	[.....]

Rockset updates these values

CPU contention between indexing and search



Reindexing \Rightarrow Resource contention

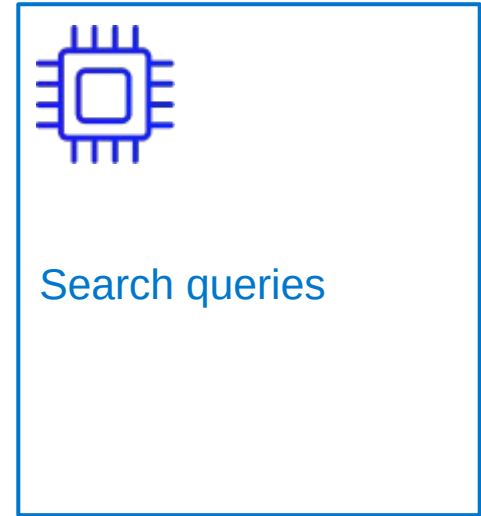


Ingestion and indexing

Search queries

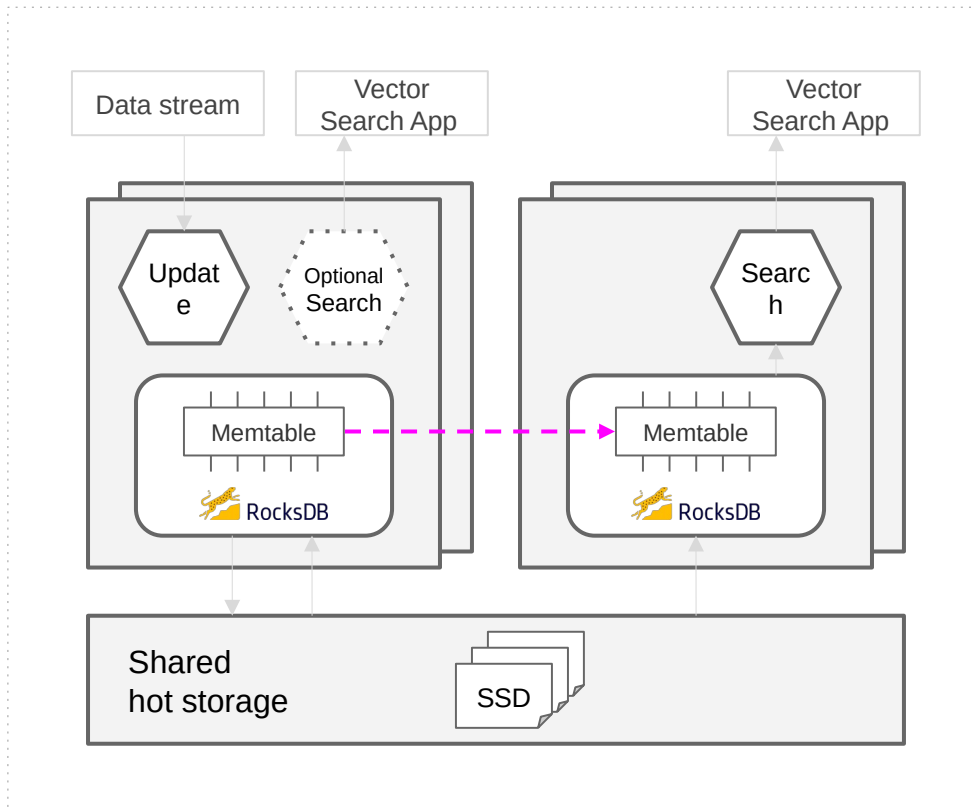
Cluster

Index and search should isolate



But how do we address that real-time thing?

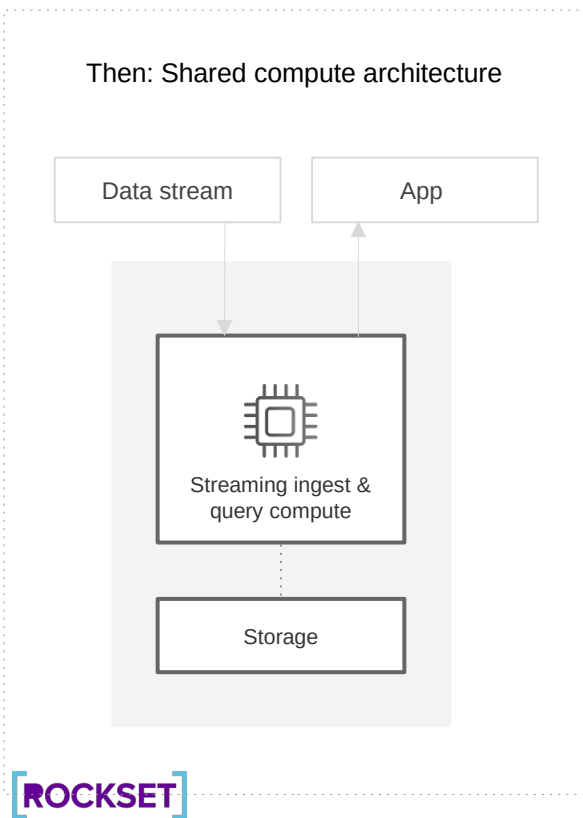
Compute-Compute Separation



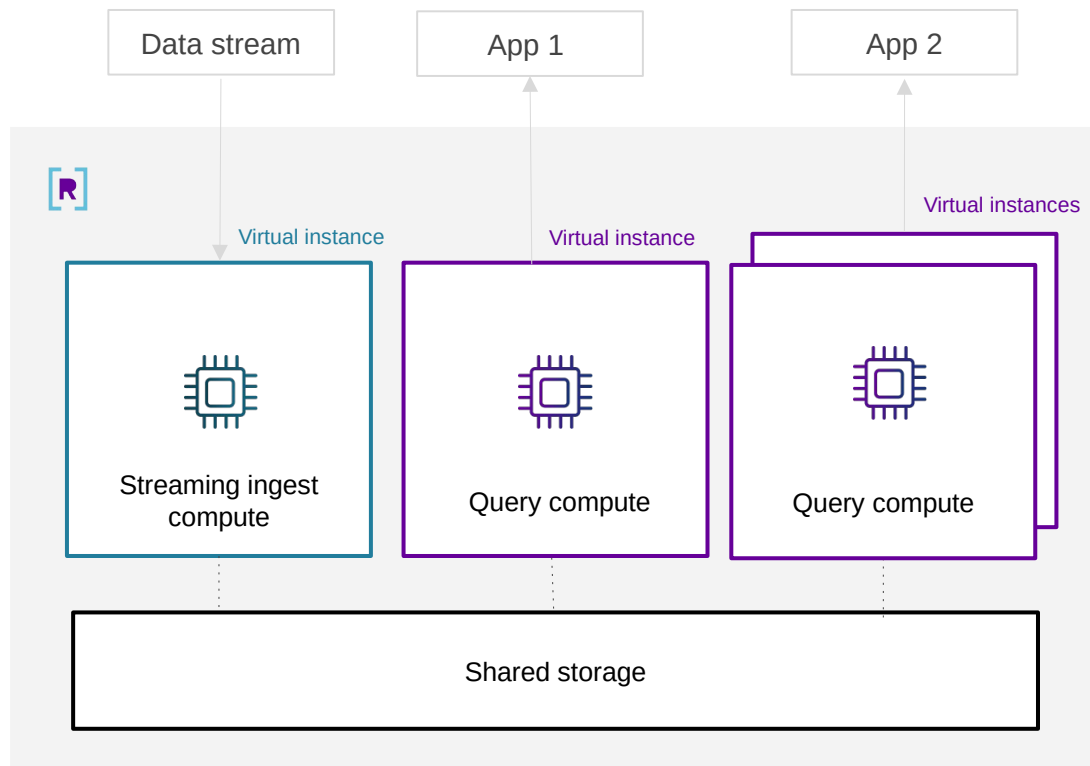
Leader/follower replication makes fresh data available in all RocksDB instances

- Replication stream sends data and metadata changes to follower
- Applying memtable updates takes 6x to 10x less CPU and RAM than complete ingest
- Followers don't run compaction

Introducing compute-compute separation



Now: Compute-compute separation



Lessons learned

- **All** vector search applications will require metadata filtering. Need a system that can do enable these without additional cost.
- Database and ML have more in common than we think, lack of a common language.
- Need to design for scale! This is a big data problem. Don't kick the can (ie: sharding + isolation) down the road.

Thankyou

Twitter [@RocksetCloud](#)

linkedin <https://www.linkedin.com/company/rocksetcloud/>



index²³

Thursday, November 2
9 am - 2:30 pm PT

📺 + 📍

 Uber Girish Baliga Director of Engineering	 Pinterest Shu Zhang Director of Engineering	 whatnot Emmanuel Fuentes Engineering Director Machine Learning & Data Platforms	 CONFLUENT Kai Waehner Global Field CTO
-----------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------

sponsored by

aws CONFLUENT

#indexconf

