6 Hard Parts of Vector Search at Scale

Ari Ekmekji
Rockset

Agenda

1. Intros
2. 6 Hard Parts Of Vector Search At Scale
3. Real World Solutions
4. Q&A

Intros

About me

Senior Engineering Manager @ Rockset (2019-Present)

- Built Converged Index and query execution engine
- Lead platform org– controlplane and ingestion

Engineering Lead @ Focal Systems (2016 - 2019)

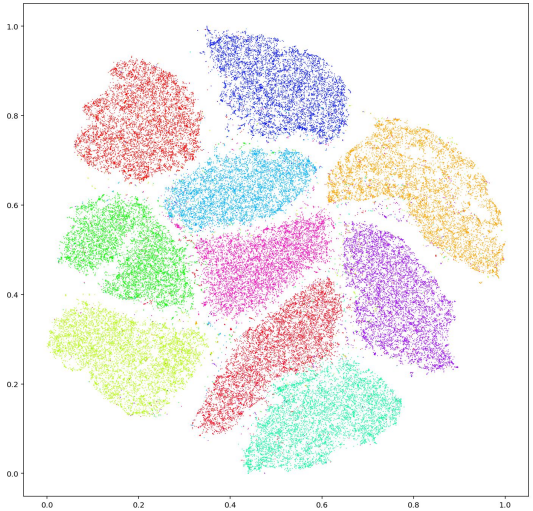
- Data and ML infra for deep learning computer vision

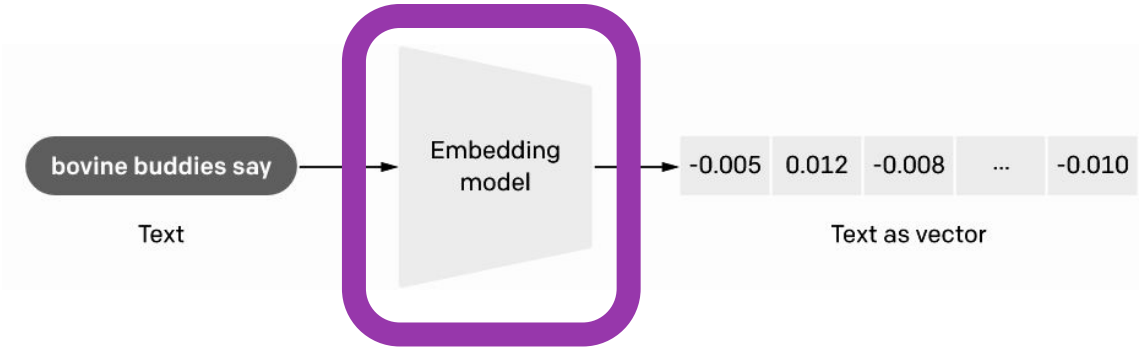
Software Engineer @ Facebook (2015)

- RocksDB team, optimizing LSM compaction

About vectors

- Given unstructured data.
  - e.g. images, movies, users
- …project/embed them into a vector space.
- …that preserves some semantic relationship (“likeness”) we care about.





6 Hard Parts

6 Hard Parts of Vector Search at Scale

1. Vector search ≠ vector database
2. Incremental indexing of vectors
3. Data latency for vectors and metadata
4. Metadata filtering
5. Filter selectivity estimation
6. Contention between indexing and vector search

Hard problem #1

Vector search ≠ vector database

- Vector DATABASES solve many, many **more "database" problems** than "vector" problems
- Beware of homegrown "vector search infra".
  - You can download HNSW, FAISS, etc. for prototyping.
  - Productionizing them will result in accidentally inventing a database.

“Database” problems

Atomicity

Transactions

Consistency

Indexes

Query Optimization

Ingestion Engine

Durability

Backups

Access Controls

Multi-tenancy

Scaling

Sharding

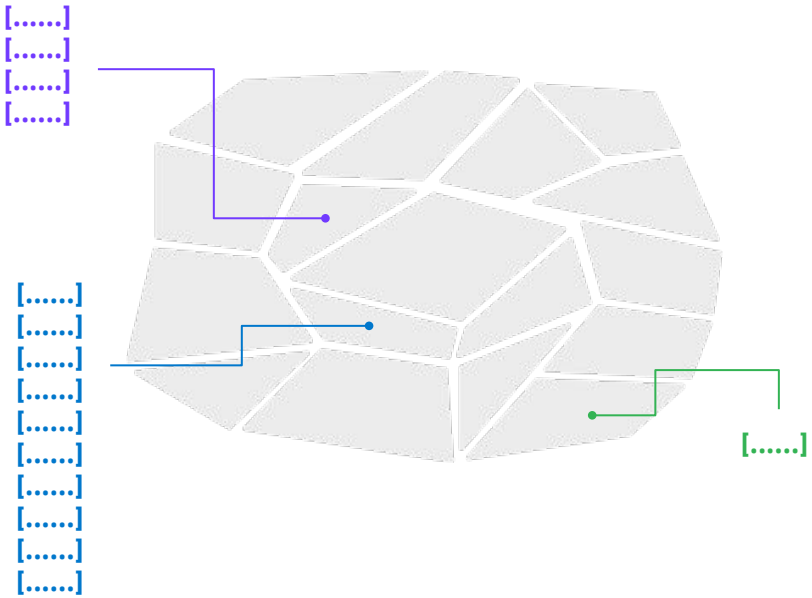
MVCC

…

Hard problem #2

Incremental indexing of vectors

- What happens when I add new vectors?
- Index is increasingly out of date.
- Naively updating is inefficient and suboptimal.
- Periodic full rebuilds are slow and resource intensive.



Strategies for incremental indexing of vectors

- Embrace suboptimal incremental updates.
- Embrace batch index.
  - Multiple indexes, making new ones as you go.
  - Occasionally “compact” them.
- Improve the core algorithms.
  - Lessen penalty for incremental updates
  - (and then publish it)

Hard problem #3

Data latency for vectors and metadata

- How long after you “create” a vector do you need it to be searchable in your index?
- Understand tolerance for data latency.
  - Both for the vector AND its metadata
- Streaming vectors is requires different architecture than batch.
- Beware propagation delays with replicated storage.

Hard problem #4

Metadata filtering

- **The WHERE clause**
- “show me all the images like this one…”
  - “… that were uploaded in the last 10 minutes.”
- “show me all the streams I might like…”
  - “… where the user is online now.”

What makes metadata filtering hard

- Index is precomputed.
  - You don’t have an index of the _filtered_ data.
- Many choices…
  - Post-filter. Overfetch vectors, then filter.
  - Pre-filter. Filter first, then scan.
  - Single-stage filter. Merge the filtering + ANN search algorithms.

Hard problem #5

Filter selectivity estimation

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

- Filter can be arbitrary expression.
- Different predicates. Different selectivities.
  - Predicates based on user behavior
- Reordering? Planning? Optimizing?
- Running filters fast is a **very well studied and hard problem.**
- **Cost-based query optimizer.**

Contention between indexing and querying

Ingestion and indexing

Search queries

Cluster

Contention between indexing and querying

- “Simple” case, 2 way contention between indexing and queries
- Harder cases at scale, contention between:
  - Multiple (e.g. batch vs streaming) ingestion workloads
  - Multiple applications’ query workloads
  - Single application’s queries as qps spikes
  - Background jobs (e.g. index build) and everything

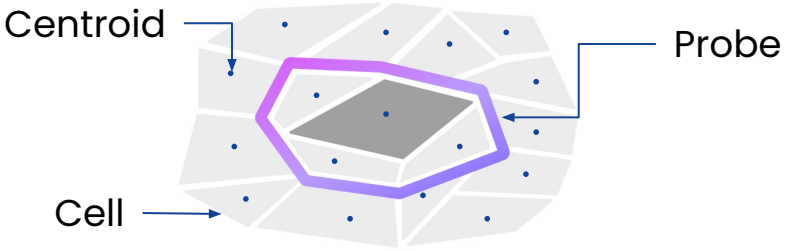
6 Hard Parts of Vector Search at Scale

1. Vector search ≠ vector database
2. Incremental indexing of vectors
3. Data latency for vectors and metadata
4. Metadata filtering
5. Filter selectivity estimation
6. Contention between indexing and vector search

Real World Solutions @ Rockset

Solving Metadata Filtering in Rockset

1. Create FAISS IVF index and store centroid identifiers in memory
2. With each new record, compute and store centroid and residual



centroid	residual	name	age	location	embedding
1	10.36	Edwin Jarvis	49	Malibu	[.....]
1	4.53	Lara Morton	46	San Francisco	[.....]
3	2.13	Marvin Adams	23	London	[.....]

Solving Metadata Filtering in Rockset

3. At query time, Rockset's query optimizer asks FAISS for the closest centroids to the target embedding:

```
SELECT name, age
WHERE location = 'Los Angeles'
FROM users
ORDER BY APPROX_DOT_PRODUCT(embedding, :target) DESC
LIMIT 5
```

Solving Metadata Filtering in Rockset

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

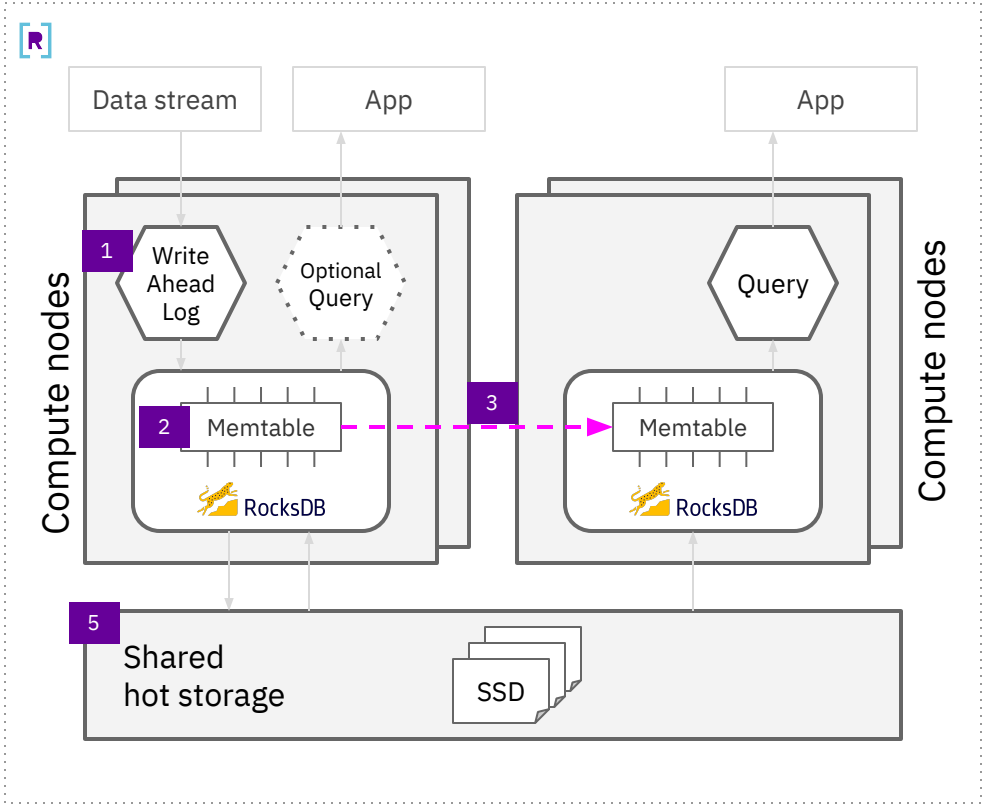
```
SELECT name, age
WHERE location = 'Los Angeles'
  AND centroid IN (2, 4, 1000)
FROM users
ORDER BY residual DESC
LIMIT 5
```

5. Rockset's execution engine uses the inverted index to perform single stage filtering.

Solving Data Latency In Rockset

- Rockset uses RocksDB as the storage engine
  - LSM engine, write optimized
  - Shred documents into KV pairs
- Rockset is mutable at the individual field level
- New records are queryable within ~100ms
  - Including updates and deletes!
- Inline generation of centroids and residuals by FAISS
- *Still need to retrain index periodically to keep recall high

Solving Resource Contention In Rockset



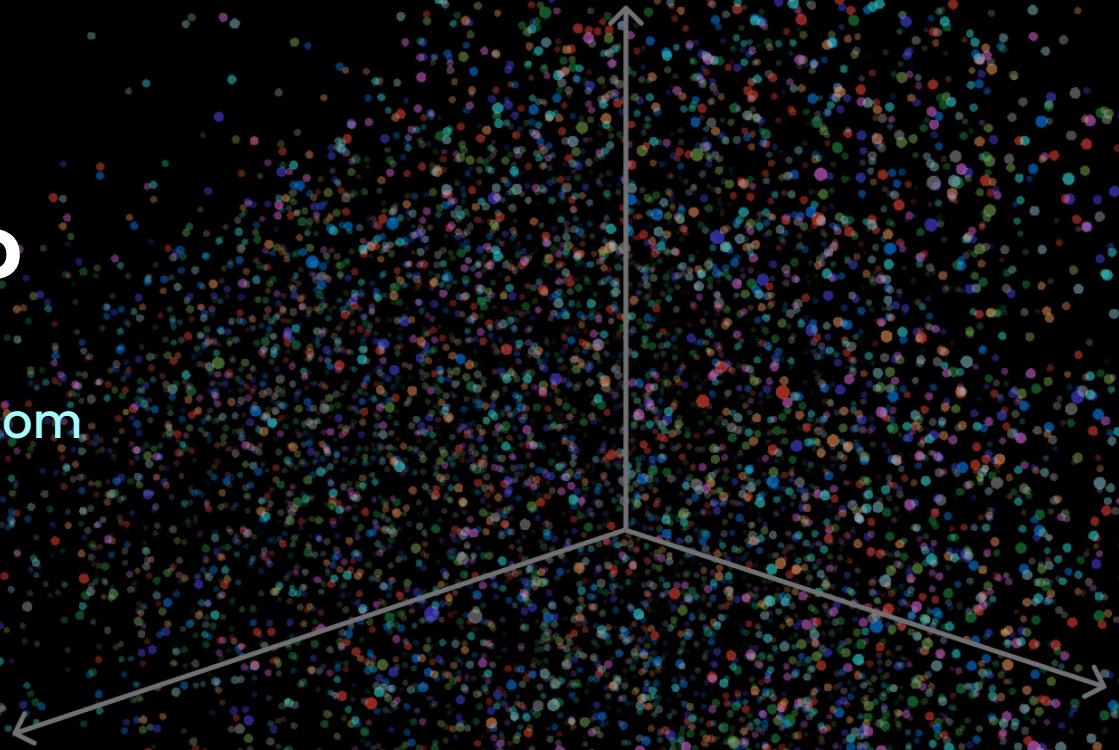
Compute/Storage and Compute/Compute Separation

1. Compaction only on primary compute nodes (VI).
2. Replicate the memtable to send (physical) changes.
3. Applying physical memtable updates takes 10x less CPU.
4. Queries access the latest updates from the memtable.
5. Scale up/down on demand without storage duplication.

Questions?

Contact me: ari@rockset.com

rockset.com/index-conf



index 23

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

+

#indexconf

sponsored by

aws CONFLUENT

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