Qdrant
A Vector Search Engine in Rust

github.com/qdrant/qdrant

Who am I

- Arnaud Gourlay
- Full time contributor @qdrant
- OSS and Rust <3
- github.com/agourlay
- agourlay.github.io

Agenda

- About Qdrant *(pronounced quadrant)*
- General introduction to vector search engines
- Use cases for the technology
- How it works internally
- What makes Rust a good fit for the job

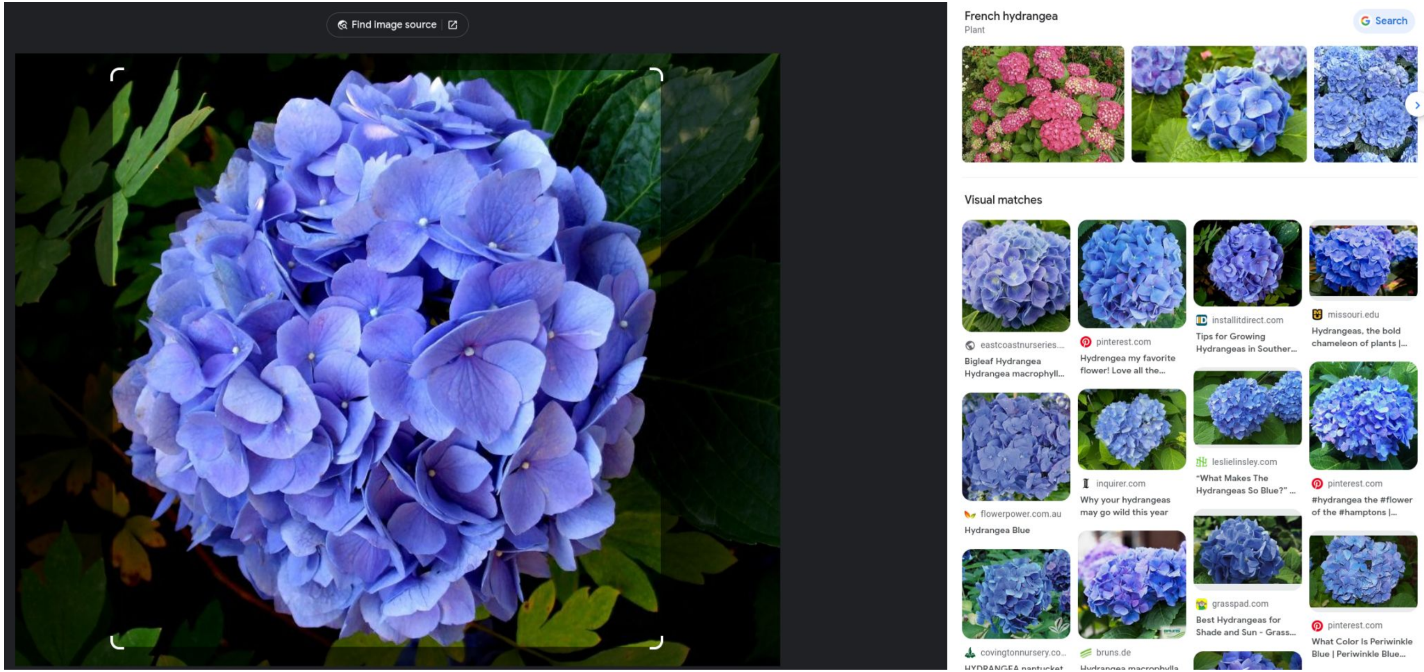
What is Qdrant?

- Open source vector search engine
- Written in Rust
- Interactions via HTTP/JSON or gRPC
- Official clients in Python, Rust & Go
- Distributed deployment
- Saas cloud offering

Evolution of search

- Traditionally exact match on a text keyword
- Growing amount of unstructured data
- Recommendations are everywhere
- Similarity search

Reverse image search (Google Lens)



Neural search

- Real example on (Simple) Wikipedia (170k documents)
- Query: `What is the capital of the United States?`
- Top-3 Hits

Lexical Search (BM25)

- **Capital** punishment (the death penalty) has existed in the **United States** [...]
- Ohio is one of the 50 **states** in the **United States**. Its **capital** is Columbus. [...]
- Nevada is one of the **United States'** **states**. Its **capital** [...]

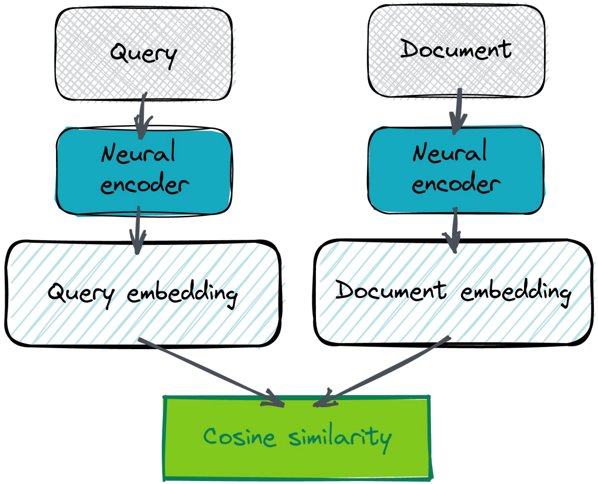
Neural Search

- Washington, D.C. [...] is the **capital of the United States**. [...]
- A capital city (or capital town or just capital) is a city or town, [...]
- The United States **Capitol** is the building where the United States Congress meets [...]

source: Nils Reimer

Similarity models

- Trained via machine learning
- “similar” inputs have vectors “close to each other” in space



Vector search engine

- Enables similarity search
- Storage: persists durably vector embeddings
- Search: find vectors most similar to an input vector
- Efficiently!

Usage example

Python

```python
from sentence_transformers import SentenceTransformer
from tqdm import tqdm
tqdm.pandas()

# load the model
model = SentenceTransformer('multi-qa-MiniLM-L6-cos-v1')

# enco
df["ve
```

Python

```python
from qdrant_client import QdrantClient
from qdrant_client.http.models import * #VectorParams

client = QdrantClien
client.recreate_coll
    collection_name=
    vectors_config=V
)
```

Python

```python
search_term = "earth observation"

search_result = client.search(
    collection_name="test_collection",
    query_vector=model.encode(search_term),
    limit=3
)

search_result
```

Source: https://geo.rocks/post/geospatial-vector-search-qdrant/

Vector payload

- Attach additional data to a vector
- Filtered search on payload fields
- Text keyword, numeric, geo coordinates …

```python
from qdrant_client import QdrantClient
from qdrant_client.http import models

client = QdrantClient(host="localhost", port=6333)

client.upsert(
    collection_name="{collection_name}",
    points=[
        models.PointStruct(
            id=1,
            vector=[0.05, 0.61, 0.76, 0.74],
            payload={
                "city": "Berlin",
                "price": 1.99,
            },
        ),
        models.PointStruct(
            id=2,
            vector=[0.19, 0.81, 0.75, 0.11],
            payload={
                "city": ["Berlin", "London"],
                "price": 1.99,
            },
        ),
        models.PointStruct(
            id=3,
            vector=[0.36, 0.55, 0.47, 0.94],
            payload={
                "city": ["Berlin", "Moscow"],
                "price": [1.99, 2.99],
            },
        ),
    ]
)
```

Naive vector search

- Store vectors in a “table” (4 dimensions in example)
- Compute similarity between input vector and **each** vector
- Return vector id with max similarity

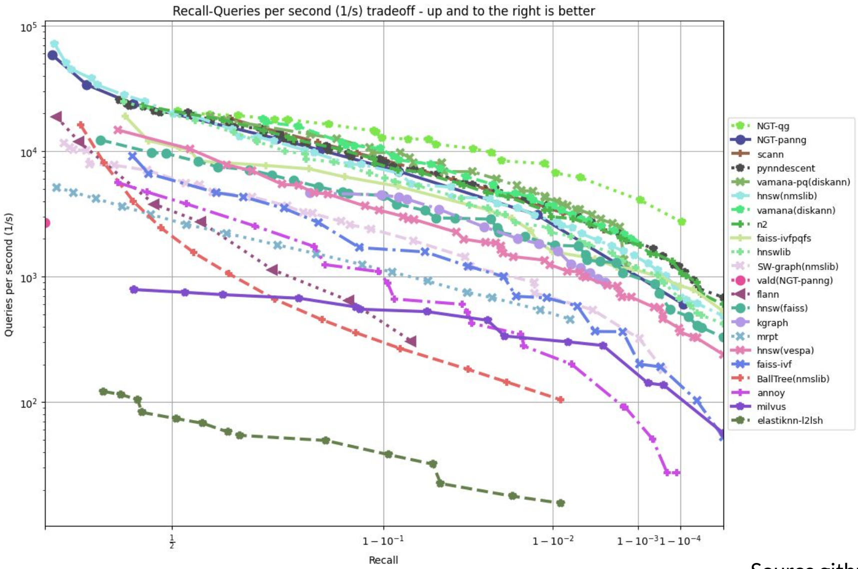
id	w	x	y	z
1	1.22	24.61	8.79	49.08
2	3.45	13.09	44.32	2.27
3	0.05	67.54	76.87	6.91
…	…	…	…	…

We need a vector index

- Traditional indexes do not fit
- Geospatial indexes (KD-trees)
- k-nearest neighbors (kNN)
- Curse of dimensionality
- How to handle very large dimensions?

ANN Search

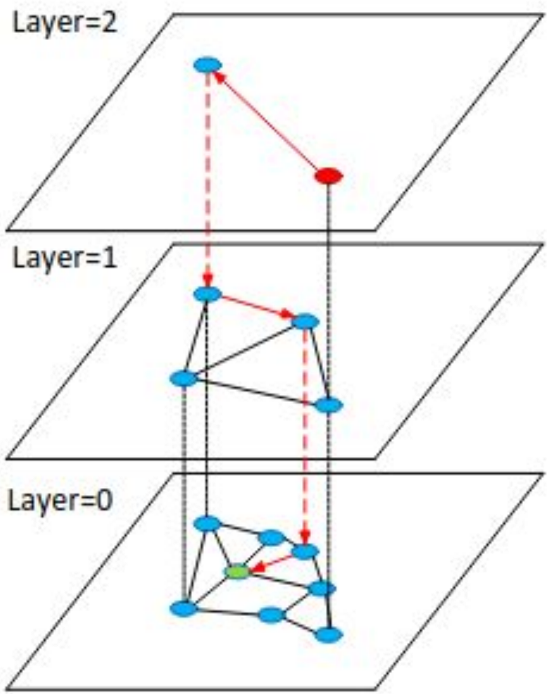
- Approximate Nearest Neighbors
- Tradeoff: precision vs speed



Source github.com/erikbern/ann-benchmarks/

Hierarchical Navigable Small Worlds

- Proximity graphs
- Skip-List
- 'M' friends per vector
- 'efSearch' per layer



*Source: [HNSW paper]

HNSW filtering

- Custom implementation to support payload filters
- Post-filtering vs Pre-filtering
- Enrich graph with indexed payload info
- Single stage search
- https://qdrant.tech/articles/filtrable-hnsw/

Distance metrics

- Very common operation (indexing & search)
- Several similarity metrics available depending on encoder

```rust
pub trait Metric {
    /// Enum value
    fn distance() -> Distance;
    /// Greater the value - closer the vectors
    fn similarity(v1: &[f32], v2: &[f32]) -> f32;
    /// Necessary vector transformations performed before adding it to the collection (normalization)
    /// Return None if metric does not required preprocessing
    fn preprocess(vector: &[f32]) -> Option<Vec<f32>>;
}
```

Naive Dot product

```
impl Metric for DotProductMetric {
	fn distance() -> Distance {
	    Distance::Dot
	}

	fn similarity(v1: &[f32], v2: &[f32]) -> ScoreType {
	     v1.iter().zip(v2).map(|(a, b)| a * b).sum()
	}
}
```

Dot product performance

- 70% CPU in “core..iter..traits..accum..SumGT::sum::”

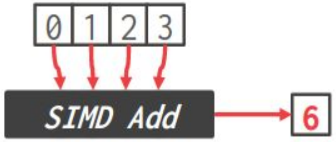
SIMD

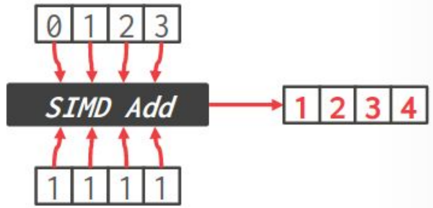
- Single Instruction Multiple Data.
- Same operation on multiple data simultaneously!
- X86: SSEs, AVXs
- ARM: NEON
- Hand-coded or auto-vectorizing compiler.

Horizontal vs vertical

- Horizontal

- Vertical





*source: CMU CS Andy Pavlo

SIMD dynamic feature detections

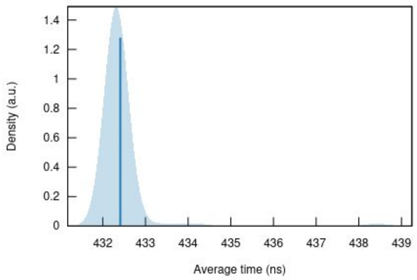
```rust
fn similarity(v1: &[VectorElementType], v2: &[VectorElementType]) -> ScoreType {
    #[cfg(target_arch = "x86_64")]
    {
        if is_x86_feature_detected!("avx")
            && is_x86_feature_detected!("fma")
            && v1.len() >= MIN_DIM_SIZE_AVX
        {
            return unsafe { dot_similarity_avx(v1, v2) };
        }
    }

    #[cfg(any(target_arch = "x86", target_arch = "x86_64"))]
    {
        if is_x86_feature_detected!("sse") && v1.len() >= MIN_DIM_SIZE_SIMD {
            return unsafe { dot_similarity_sse(v1, v2) };
        }
    }

    #[cfg(all(target_arch = "aarch64", target_feature = "neon"))]
    {
        if std::arch::is_aarch64_feature_detected!("neon") && v1.len() >= MIN_DIM_SIZE_SIMD {
            return unsafe { dot_similarity_neon(v1, v2) };
        }
    }

    dot_similarity(v1, v2)
}
```

AVX SIMD dot product

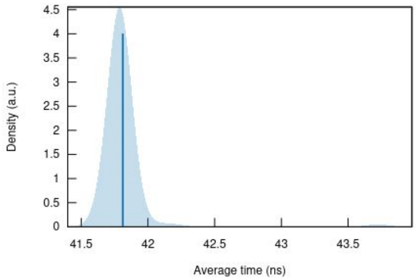
```rust
use std::arch::x86_64::*;

#[target_feature(enable = "avx")]
#[target_feature(enable = "fma")]
pub unsafe fn dot_avx(v1: &[f32], v2: &[f32]) -> f32 {
    let mut sum256: __m256 = _mm256_setzero_ps(); // accumulated sum
    for i in 0..v1.len() / 8 {
        let v1_256: __m256 = _mm256_loadu_ps(v1[8 * i..].as_ptr()); // vector floats from v1
        let v2_256: __m256 = _mm256_loadu_ps(v2[8 * i..].as_ptr()); // vector floats from v2
        sum256 = _mm256_fmadd_ps(v1_256, v2_256, sum256); // sum256 += v1_256 * v2_256
    }

    // let's sum all elements in sum256
    let sum128: __m128 = _mm_add_ps(_mm256_extractf128_ps(sum256, 1), _mm256_castps256_ps128(sum256));
    let sum64: __m128 = _mm_add_ps(sum128, _mm_movehl_ps(sum128, sum128));
    let sum32: __m128 = _mm_add_ss(sum64, _mm_shuffle_ps(sum64, sum64, 0x55));
    _mm_cvtss_f32(sum32)
}
```

Criterion benchmark

dot-product-group/dot_iter



dot-product-group/dot_simd

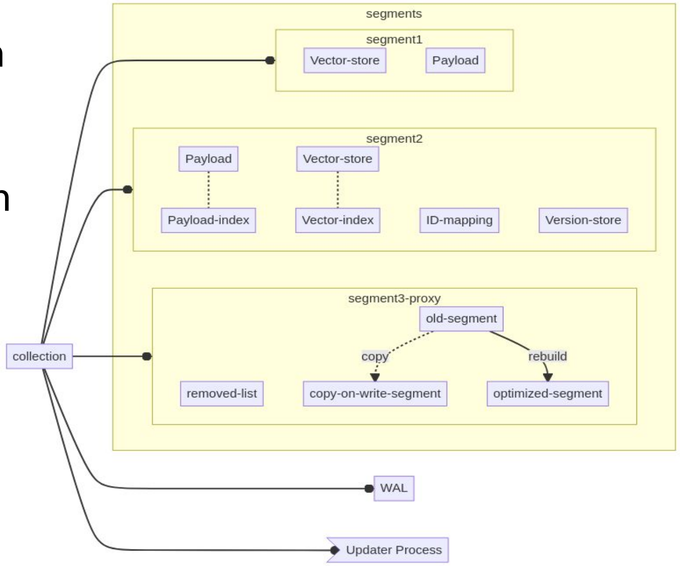


SIMD indexing impact

- 100k vectors of dim. 500
- HNSW index with dot product
- Dot iterator: 333 seconds
- Dot SIMD: 95 seconds

Vector collection

- Several segments per collection
- Persistence with RocksDB
- Optimizers keeping things clean
- In memory vs memmap



Concurrent programing

- Qdrant is inherently stateful
- Manage concurrent accesses
- Threads sharing data via Channel
- Threads synchronizing on Mutex/RwLock

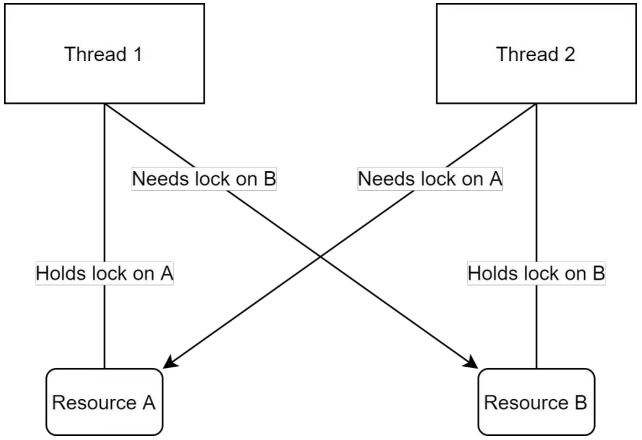
RwLock

- API enforces proper usage
- No unlock method!
- Read guard impl. Deref
- Write guard impl. DerefMut
- Check out parking_lot

```rust
use std::sync::RwLock;

let lock = RwLock::new(5);

// many reader locks can be held at once
{
    let r1 = lock.read().unwrap();
    let r2 = lock.read().unwrap();
    assert_eq!(*r1, 5);
    assert_eq!(*r2, 5);
} // read locks are dropped at this point

// only one write lock may be held, however
{
    let mut w = lock.write().unwrap();
    *w += 1;
    assert_eq!(*w, 6);
} // write lock is dropped here
```

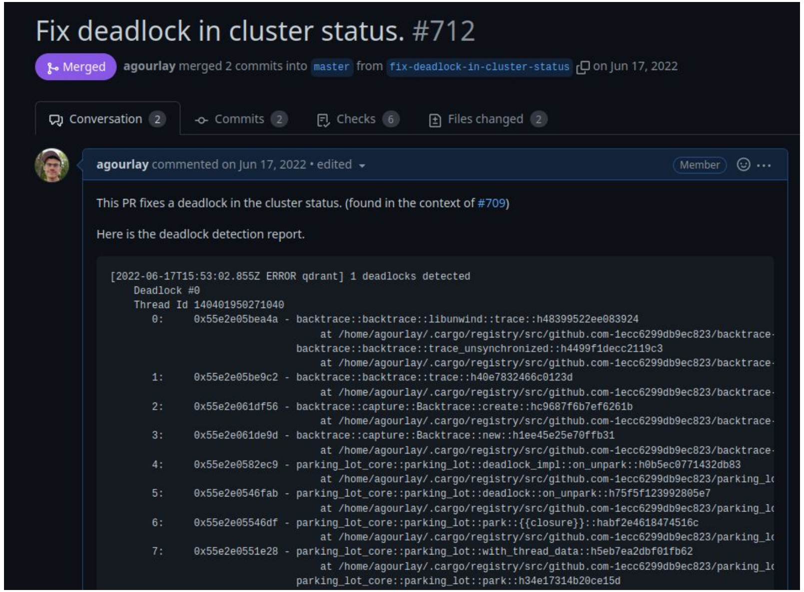
Dead locks

- Threads waiting for each other
- Double reads on the same thread
- Not only locks
- Not caught by rustc
- Requires discipline



Runtime deadlock detector

- parking_lot “deadlock_detection” build feature



Fix deadlock in cluster status. #712

Merged agourlay merged 2 commits into master from fix-deadlock-in-cluster-status on Jun 17, 2022

Conversation 2 Commits 2 Checks 6 Files changed 2

agourlay commented on Jun 17, 2022 • edited Member

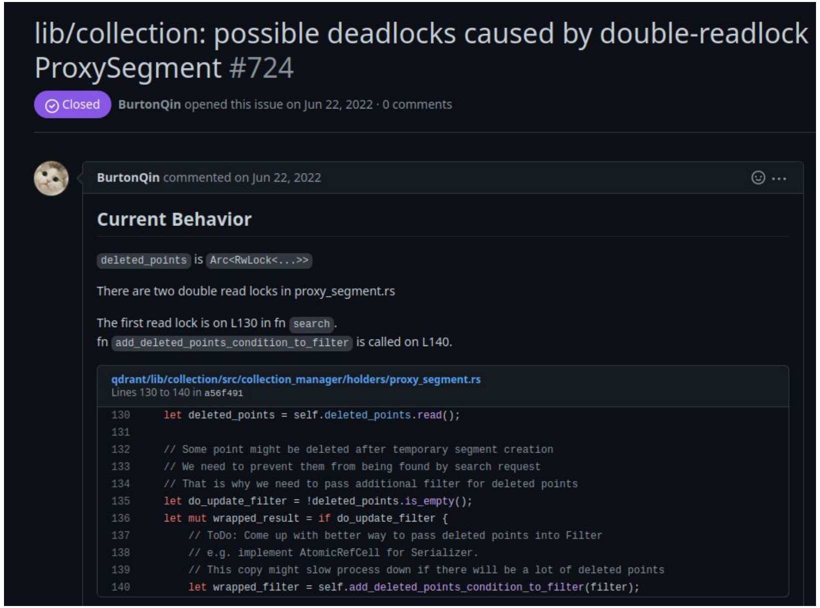
This PR fixes a deadlock in the cluster status. (found in the context of #709)

Here is the deadlock detection report.

```
[2022-06-17T15:53:02.855Z ERROR qdrant] 1 deadlocks detected
    Deadlock #0
    Thread Id 140401950271040
       0:     0x55e2e05bea4a - backtrace::backtrace::libunwind::trace::h48399522ee083924
                                   at /home/agourlay/.cargo/registry/src/github.com-1ecc6299db9ec823/backtrace-
                               backtrace::backtrace::trace_unsynchronized::h4499f1decc2119c3
                                   at /home/agourlay/.cargo/registry/src/github.com-1ecc6299db9ec823/backtrace-
       1:     0x55e2e05be9c2 - backtrace::backtrace::trace::h40e7832466c0123d
                                   at /home/agourlay/.cargo/registry/src/github.com-1ecc6299db9ec823/backtrace-
       2:     0x55e2e061df56 - backtrace::capture::Backtrace::create::hc9687f6b7ef6261b
                                   at /home/agourlay/.cargo/registry/src/github.com-1ecc6299db9ec823/backtrace-
       3:     0x55e2e061de9d - backtrace::capture::Backtrace::new::h1ee45e25e70ffb31
                                   at /home/agourlay/.cargo/registry/src/github.com-1ecc6299db9ec823/backtrace-
       4:     0x55e2e0582ec9 - parking_lot_core::parking_lot::deadlock_impl::on_unpark::h0b5ec0771432db83
                                   at /home/agourlay/.cargo/registry/src/github.com-1ecc6299db9ec823/parking_l
       5:     0x55e2e0546fab - parking_lot_core::parking_lot::deadlock::on_unpark::h75f5f123992805e7
                                   at /home/agourlay/.cargo/registry/src/github.com-1ecc6299db9ec823/parking_l
       6:     0x55e2e05546df - parking_lot_core::parking_lot::park::{{closure}}::habf2e4618474516c
                                   at /home/agourlay/.cargo/registry/src/github.com-1ecc6299db9ec823/parking_l
       7:     0x55e2e0551e28 - parking_lot_core::parking_lot::with_thread_data::h5eb7ea2dbf01fb62
                                   at /home/agourlay/.cargo/registry/src/github.com-1ecc6299db9ec823/parking_l
                               parking_lot_core::parking_lot::park::h34e17314b20ce15d
```

Static deadlock detector

- github.com/BurtonQin/lockbud



lib/collection: possible deadlocks caused by double-readlock ProxySegment #724

Closed BurtonQin opened this issue on Jun 22, 2022 · 0 comments

BurtonQin commented on Jun 22, 2022

Current Behavior

`deleted_points` is `Arc<RwLock<...>>`

There are two double read locks in proxy_segment.rs

The first read lock is on L130 in fn `search`.
fn `add_deleted_points_condition_to_filter` is called on L140.

qdrant/lib/collection/src/collection_manager/holders/proxy_segment.rs
Lines 130 to 140 in a56f491

```
130     let deleted_points = self.deleted_points.read();
131
132     // Some point might be deleted after temporary segment creation
133     // We need to prevent them from being found by search request
134     // That is why we need to pass additional filter for deleted points
135     let do_update_filter = !deleted_points.is_empty();
136     let mut wrapped_result = if do_update_filter {
137         // ToDo: Come up with better way to pass deleted points into Filter
138         // e.g. implement AtomicRefCell for Serializer.
139         // This copy might slow process down if there will be a lot of deleted points
140         let wrapped_filter = self.add_deleted_points_condition_to_filter(filter);
```

Going distributed

- Stay available
- Scaling out on commodity machines
- Shards & replicas per collection
- No leader shard for writes
- Transactional vector operations are opt-in

Raft consensus

- Used to synchronize cluster & collection topology
- Agree on sequence of operations (log)
- Leader commits after entry replicated by majority
- raft.rs is not easy but maintainers are responsive
- Debugging can be very difficult

Takeaways

- Demystified vector search engine
- New indexing schemes (ANN/HNSW)
- Apply SIMD to bottlenecks if possible
- Mind the deadlocks
- Distributed systems are hard
- Another data point in favor of Rust

Farewell links

- https://github.com/qdrant/qdrant
- https://qdrant.tech/documentation
- https://qdrant.tech/benchmarks
- https://blog.qdrant.tech
- https://qdrant.to/discord