Теоретический анализ векторной базы данных Qdrant: Фундаментальные принципы и математический аппарат

1. Введение: Математические основы векторного поиска

Векторные базы данных, такие как Qdrant, оперируют в рамках многомерных евклидовых или метрических пространств. Фундаментальной задачей, которую они решают, является поиск по сходству (similarity search), который формализуется как задача поиска ближайшего соседа (Nearest Neighbor, NN).

Определение 1: Векторное пространство и эмбеддинги. Пусть $\mathcal{X}$ — пространство исходных объектов (например, текстов, изображений, аудиосигналов). Эмбеддинг (vector embedding) представляет собой отображение $f: \mathcal{X} \to \mathbb{R}^d$, которое сопоставляет каждому объекту $x \in \mathcal{X}$ вектор $\vec{v} \in \mathbb{R}^d$ в d-мерном вещественном векторном пространстве. Предполагается, что это отображение сохраняет семантическую близость: если объекты x_1 и x_2 семантически близки, то расстояние между их векторными представлениями $f(x_1)$ и $f(x_2)$ мало.

Определение 2: Метрики сходства. Сходство или различие между векторами $\vec{u}, \vec{v} \in \mathbb{R}^d$ измеряется с помощью метрики расстояния $d(\vec{u}, \vec{v})$ или функции сходства $S(\vec{u}, \vec{v})$. Qdrant поддерживает несколько ключевых метрик:

1. **Косинусное сходство (Cosine Similarity):**

$$S_C(\vec{u}, \vec{v}) = \frac{\vec{u} \cdot \vec{v}}{\|\vec{u}\|\|\vec{v}\|} = \frac{\sum_{i=1}^{d} u_i v_i}{\sqrt{\sum_{i=1}^{d} u_i^2} \sqrt{\sum_{i=1}^{d} v_i^2}}$$

   Косинусное сходство эквивалентно косинусу угла θ между векторами и не зависит от их магнитуды. Оно принимает значения в диапазоне $[-1, 1]$. В контексте поиска, часто используется косинусное расстояние $d_C(\vec{u}, \vec{v}) = 1 - S_C(\vec{u}, \vec{v})$.

2. **Евклидово расстояние (Euclidean Distance, L2-норма):**

$$d_E(\vec{u}, \vec{v}) = \|\vec{u} - \vec{v}\|_2 = \sqrt{\sum_{i=1}^{d} (u_i - v_i)^2}$$

   Эта метрика представляет собой прямолинейное расстояние между двумя точками в d-мерном пространстве.

3. **Скалярное произведение (Dot Product):**

$$S_{DP}(\vec{u}, \vec{v}) = \vec{u} \cdot \vec{v} = \sum_{i=1}^{d} u_i v_i$$

   Для векторов, нормализованных до единичной длины (т.е. $\|\vec{u}\| = \|\vec{v}\| = 1$), скалярное произведение эквивалентно косинусному сходству. Qdrant может оптимизировать вычисления для этого случая.

Определение 3: Задача поиска приблизительного ближайшего соседа (Approximate Nearest Neighbor, ANN). Ввиду вычислительной сложности точного поиска NN в пространствах высокой размерности (проклятие размерности), на практике решается задача ANN. Пусть $P \subset \mathbb{R}^d$ — набор векторов, а $q \in \mathbb{R}^d$ — вектор запроса. Задача состоит в том, чтобы найти такой вектор $p' \in P$, который удовлетворяет условию:

$$d(q, p') \leq (1 + \varepsilon) \min_{p \in P} d(q, p)$$

где $\varepsilon > 0$ — параметр, определяющий допустимую погрешность, а $\min_{p \in P} d(q, p)$ — расстояние до истинного ближайшего соседа. Qdrant использует индексирующие структуры для эффективного решения задачи ANN.

---

2. Основная структура индексации: Hierarchical Navigable Small World (HNSW)

HNSW является основным алгоритмом индексации в Qdrant для решения задачи ANN. Он основан на создании многоуровневого графа близости.

Определение 4: Граф близости (Proximity Graph). Граф близости $G = (V, E)$, где V — множество векторов, а ребро $(u, v) \in E$ существует, если вектор v является одним из "ближайших" соседей вектора u согласно выбранной метрике.

Теоретические основы HNSW: Алгоритм строит иерархию графов $G_0, G_1, \ldots, G_L$.

- **Базовый слой (G_0):** Содержит все векторы из набора данных. В этом графе каждый узел соединен с некоторым числом своих ближайших соседей. Этот граф обладает свойством "маленького мира" (small world), но навигация по нему может быть неэффективной.
- **Верхние слои ($G_l, l > 0$):** Каждый последующий слой G_l является подграфом предыдущего G_{l-1}, то есть $V_l \subset V_{l-1}$. Узлы для верхних слоев выбираются вероятностным образом, что обеспечивает логарифмическое уменьшение числа узлов с ростом номера слоя.

Алгоритм вставки нового элемента q:

1. **Выбор максимального слоя:** Для нового вектора q выбирается максимальный слой l_{max}, на котором он будет присутствовать. Вероятность выбора слоя l экспоненциально убывает: $P(l) \propto e^{-l/m_L}$, где m_L — нормализующий множитель. Это гарантирует, что на верхних слоях будет экспоненциально меньше узлов.
2. **Поиск точки входа:** Поиск начинается с самой верхней точки входа на самом верхнем слое L.
3. **Итеративный спуск:** На каждом слое l от L до $l_{max} + 1$ выполняется жадный поиск ближайшего к q элемента. Найденный элемент на слое l становится точкой входа для поиска на слое $l - 1$.
4. **Вставка и соединение:** На каждом слое l от $\min(L, l_{max})$ до 0 выполняется следующая процедура: a. С помощью beam search (поиска с ограничением ширины) находится M ближайших соседей для q среди узлов слоя l. b. Устанавливаются двунаправленные ребра между q и найденными соседями. c. **Эвристическое прореживание:** Если число соседей у какого-либо узла превышает заданный максимум M_{max}, "лишние" ребра (ведущие к наиболее удаленным соседям) удаляются для поддержания ограниченной степени вершин графа.

Алгоритм поиска k ближайших соседей к запросу q:

1. **Поиск точки входа:** Аналогично вставке, начинается поиск с верхнего слоя L.
2. **Жадный спуск:** На каждом слое от L до 1 выполняется жадный поиск для нахождения узла, ближайшего к q. Этот узел служит точкой входа для следующего, более плотного слоя.
3. **Поиск на базовом слое:** На слое G_0 выполняется beam search, начиная с точки входа, найденной на слое G_1. Используется приоритетная очередь для хранения кандидатов, что позволяет найти k наиболее близких векторов с высокой вероятностью.

Анализ сложности: Средняя сложность поиска в HNSW составляет $O(\log N)$, где N — общее число векторов. Это достигается за счет иерархической структуры: верхние слои с "длинными" ребрами позволяют быстро перемещаться по пространству, а нижние, более плотные слои, обеспечивают точность локального поиска. Параметры `ef_construct` и `ef_search` контролируют компромисс между скоростью построения/поиска и точностью (recall).

---

3. Оптимизации и уникальные механизмы Qdrant

Qdrant расширяет стандартные подходы, внедряя собственные механизмы для оптимизации производительности, потребления памяти и функциональности.

3.1. Квантование (Quantization)

Квантование — это процесс сокращения точности представления векторов для уменьшения занимаемой памяти и ускорения вычислений.

A. Скалярное квантование (Scalar Quantization, SQ): SQ преобразует 32-битные числа с плавающей запятой (float32) в 8-битные целые числа (int8).

- **Математическая деривация:** Пусть дан вектор $\vec{v} = (v_1, \ldots, v_d) \in \mathbb{R}^d$. Для каждой компоненты v_i необходимо найти ее квантованное представление v'_i.

  1. **Определение диапазона:** Для всего набора данных (или его подвыборки) находятся минимальное (v_{min}) и максимальное (v_{max}) значения компонент. Для простоты можно считать, что для каждой компоненты i есть свой диапазон $[min_i, max_i]$. Однако Qdrant часто использует перцентили (например, 2-й и 98-й) для устойчивости к выбросам.

  2. **Вычисление шага квантования:** Для b-битного квантования (в Qdrant обычно $b = 8$, что дает $2^8 = 256$ уровней) шаг квантования Δ вычисляется как:

  $$\Delta = \frac{v_{max} - v_{min}}{2^b - 1}$$

  3. **Квантование:** Каждая компонента v_i преобразуется в целое число v'_i:

$$v'_i = \text{round}\left(\frac{v_i - v_{min}}{\Delta}\right)$$

где $\text{round}(\cdot)$ — функция округления до ближайшего целого.

4. **Де-квантование (восстановление):** При необходимости, приближенное исходное значение $\hat{v}_i$ можно восстановить:

$$\hat{v}_i = v'_i \cdot \Delta + v_{min}$$

- **Применение на практике:** Qdrant может вычислять метрики расстояния непосредственно на квантованных векторах. Например, для скалярного произведения, вычисление $\sum(a_i \cdot b_i)$ с использованием целочисленных инструкций (особенно с SIMD-расширениями, такими как AVX) значительно быстрее, чем с float32. Это снижает потребление памяти в 4 раза (с $d \times 4$ байт до $d \times 1$ байт) и ускоряет вычисления. Ошибка квантования при этом вносится, но для многих задач RAG (Retrieval-Augmented Generation) она является приемлемой.

B. Продуктовое квантование (Product Quantization, PQ): PQ — более сложный метод, обеспечивающий еще большее сжатие.

- **Математическая деривация:**

1. **Разбиение вектора:** Вектор $\vec{v} \in \mathbb{R}^d$ разбивается на m суб-векторов $\vec{v}_1, \ldots, \vec{v}_m$ размерности d/m каждый.
2. **Создание кодовых книг:** Для каждого из m подпространств с помощью алгоритма k-means на обучающем наборе данных создается своя "кодовая книга" (codebook) C_j, состоящая из k центроид (кодовых слов). Обычно $k = 256$.
3. **Кодирование:** Каждый суб-вектор $\vec{v}_j$ заменяется индексом i_j ближайшего к нему центроида из соответствующей кодовой книги C_j:

$$i_j = \arg\min_{l \in \{1, \ldots, k\}} d(\vec{v}_j, c_{j,l})$$

где $c_{j,l}$ — l-й центроид в j-й кодовой книге. Таким образом, весь вектор $\vec{v}$ представляется набором из m индексов $(i_1, \ldots, i_m)$.

- **Применение на практике (Asymmetric Distance Computation):** Для вычисления расстояния между вектором запроса q и сжатым вектором v не требуется полная декомпрессия.

1. Запрос q также разбивается на суб-векторы $(q_1, \ldots, q_m)$.
2. Для каждого подпространства j предварительно вычисляется таблица расстояний между q_j и всеми k центроидами из кодовой книги C_j: $D_j[l] = d(q_j, c_{j,l})^2$.
3. Квадрат евклидова расстояния между q и v аппроксимируется суммой по предвычисленным значениям:

$$d(q, v)^2 \approx \sum_{j=1}^{m} d(q_j, c_{j,i_j})^2 = \sum_{j=1}^{m} D_j[i_j]$$

Этот метод требует всего m операций сложения и m обращений к памяти, что чрезвычайно быстро. Потребление памяти сокращается до $m \times \log_2(k)$ бит на вектор. Qdrant использует PQ для хранения векторов на диске, обеспечивая возможность работы с наборами данных, превышающими объем RAM.

3.2. Механизм фильтрации

Qdrant предоставляет мощный механизм фильтрации по метаданным (payload), который интегрирован с ANN-поиском. Это решает задачу **фильтрованного ANN-поиска**.

Определение 5: Инвертированный индекс (Inverted Index). Для эффективной фильтрации Qdrant строит инвертированный индекс по полям метаданных. Инвертированный индекс — это структура данных, которая отображает значения полей (`term`) в список идентификаторов векторов (`postings list`), у которых данное поле имеет данное значение. Пример: `{"color": "blue"}` -> `[id_1, id_5, id_42, ...]`.

Проблема и решение Qdrant: Наивный подход — "post-filtering" (сначала найти k соседей, потом отфильтровать) — неэффективен и может вернуть пустой результат. Qdrant реализует "pre-filtering":

1. С помощью инвертированного индекса получается множество ID векторов S_{filter}, удовлетворяющих условию фильтра.
2. ANN-поиск выполняется только по векторам из множества S_{filter}.

Ключевая инновация Qdrant заключается в том, что его реализация HNSW эффективно работает с разреженными наборами идентификаторов. Во время обхода графа, если узел не принадлежит S_{filter}, он пропускается. Это позволяет избежать катастрофического падения производительности, характерного для многих других реализаций HNSW при работе с произвольными подмножествами векторов.

---

4. Надежность и персистентность

Write-Ahead Log (WAL): Qdrant обеспечивает долговечность (Durability из ACID) с помощью журнала упреждающей записи.

- **Принцип:** Любая операция модификации (вставка, удаление, обновление) сначала записывается в последовательный лог на диске и только потом применяется к структурам в памяти.
- **Гарантия:** В случае сбоя, состояние системы можно восстановить, воспроизведя записи из WAL поверх последнего консистентного снимка (snapshot). Это гарантирует, что ни одна подтвержденная запись не будет потеряна.

Динамические обновления и удаления: Удаление узла из HNSW-графа — нетривиальная операция, так как может нарушить связность графа.

- **Решение Qdrant:** Вместо физического удаления узла, он помечается как удаленный (например, с помощью битовой маски). Во время поиска такие узлы игнорируются. Физическое удаление и реорганизация графа происходят асинхронно в процессе оптимизации (compaction), что обеспечивает высокую производительность операций записи и удаления без блокировки чтения.

---

5. Заключение: Синтез теоретических принципов

Qdrant представляет собой сложную систему, построенную на синергии фундаментальных концепций из различных областей информатики:

- **Теория графов и алгоритмы:** HNSW для эффективного ANN-поиска с логарифмической сложностью.
- **Теория информации и обработка сигналов:** Скалярное и продуктовое квантование для сжатия данных и ускорения вычислений.
- **Теория баз данных:** Инвертированные индексы для быстрой фильтрации по метаданным и WAL для обеспечения персистентности и надежности.

Уникальность Qdrant заключается не только в реализации этих алгоритмов, но и в их глубокой интеграции. Способность эффективно выполнять предварительную фильтрацию в HNSW, сочетая ее с многоуровневым квантованием и надежной моделью хранения, выделяет его как высокопроизводительное и функционально богатое решение для задач векторного поиска в промышленных масштабах.