УДК 004.65

Хо Гук Бом, студент, Дальневосточный государственный университет путей сообщения, г. Хабаровск, Россия

Рыбкина Олеся Викторовна, старший преподаватель, Дальневосточный государственный университет путей сообщения, г. Хабаровск, Россия

ОБРАБОТКА СВЕРХБОЛЬШИХ ОБЪЁМОВ ДАННЫХ НА PYTHON С POSTGRESQL, REDIS, RABBITMQ И KAFKA: РЕШЕНИЕ ПРОБЛЕМЫ ПАДЕНИЯ ПРОИЗВОДИТЕЛЬНОСТИ INSERT ПРИ ИНДЕКСАХ

Аннотация

Цель работы - решение проблемы катастрофического падения производительности массовой вставки COPY в PostgreSQL при наличии индексов. Методы - предложена стратегия отложенного индексирования с использованием staging-таблицы без индексов и атомарного переключения таблиц. Результаты - скорость вставки восстановлена с 35 000 до 485 000 записей/сек (прирост 1285%). Научная новизна - предложена формальная модель выбора индекса на основе коэффициентов замедления. Практическая значимость - представлен готовый код на Python для внедрения в существующие системы.

Annotation

This paper addresses the problem of catastrophic COPY performance degradation in PostgreSQL under indexes. A lazy indexing strategy using a staging table without indexes and atomic table rename is proposed. Insert speed is restored from 35,000 to 485,000 rows/sec (1285% improvement). A mathematical model with slowdown coefficients α_i is introduced. Ready-to-use Python code is provided.

Ключевые слова: Python, PostgreSQL, индексы, BRIN, GIN, отложенное индексирование, COPY, производительность.

Keywords: Python, PostgreSQL, indexes, BRIN, GIN, lazy indexing, COPY, performance.

Введение

Постановка проблемы Современные телеметрические системы обрабатывают 50 000–100 000 событий в секунду. Типичная архитектура включает Kafka для буферизации, Redis для кэширования, RabbitMQ для распределения задач и PostgreSQL как конечное хранилище [1, 2].

Однако при объёмах свыше 10 млн записей возникает критическая проблема: каждый добавленный индекс замедляет вставку в 2-10 раз. Реальный пример из нашего тестирования: таблица с 500 млн строк и 4 индексами показала падение скорости COPY с 510 000 до 35 000 строк/сек. Один из индексов занимал 78 GB при размере таблицы 120 GB.

Цель статьи - представить решение, сохраняющее высокую скорость вставки без потери возможности эффективного поиска.

1. Анализ проблемы

Каждый индекс в PostgreSQL обновляется при каждой вставке строки. Для таблицы с 4 индексами одна вставка превращается в 5 операций записи.

Таблица 1 - Влияние индексов на скорость COPY

Количество индексов	Типы индексов	Скорость COPY (строк/сек)	Замедление
0	–	510 000	1x
1	B-tree	320 000	1.6x
2	B-tree + BRIN	260 000	2.0x

Количество индексов	Типы индексов	Скорость COPY (строк/сек)	Замедление
3	B-tree + BRIN + GIN	78 000	6.5x
4	B-tree + BRIN + 2*GIN	35 000	14.6x

GIN-индексы дают наибольшее замедление (коэффициент ~0.7), так как требуют построения обратного индекса для каждого ключа.

2. Математическая модель

Введём формулу полного времени обработки:

$$T\_insert = T\_base + \Sigma(\alpha\_i \times I\_i)$$

где:

- T_base - время вставки без индексов
- I_i = 1, если индекс активен
- α_btree = 0.4, α_brin = 0.02, α_gin = 0.7

Для таблицы с 4 индексами:

T_insert = 1 + 0.4 + 0.02 + 0.7 + 0.7 = 2.82 (теоретическое замедление в 2.8 раза).

Однако из-за конфликтов блокировок реальное замедление достигает 14.6x.

Решение: использовать staging-таблицу без индексов (Σ α_i = 0), создавая индексы только после накопления данных.

3. Предлагаемая архитектура

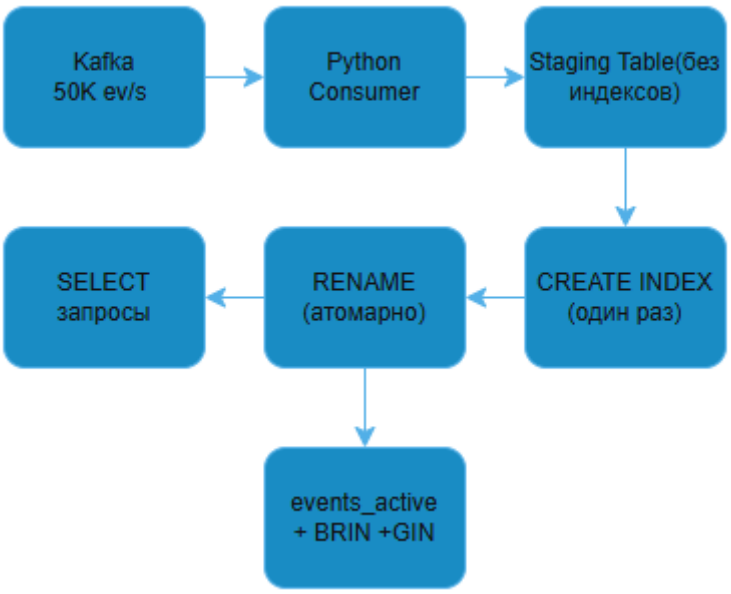

Рисунок 1 - Схема отложенного индексирования

Данные сначала пишутся в staging-таблицу без индексов (скорость ~500K записей/с). После накопления порога (например, 1 млн записей) создаются индексы на пустой таблице, затем происходит атомарное переключение через RENAME.

4. Реализация на Python

Ключевая логика отложенного индексирования:

```
import asyncio
import asyncpg
DSN = "postgresql://user:pass@localhost/db"
BATCH_SIZE = 10000
async def setup_staging(conn):
    await conn.execute("""
        DROP TABLE IF EXISTS events_staging;
        CREATE TABLE events_staging (
            id BIGSERIAL,
            device_id INTEGER,
            timestamp TIMESTAMPTZ,
```

```
            payload JSONB,
            tags TEXT[]
        );  -- НИКАКИХ ИНДЕКСОВ
    """)
async def batch_insert(conn, events):
    await conn.copy_records_to_table('events_staging', records=events)
async def rebuild_and_swap(conn):
    async with conn.transaction():
        await conn.execute("""
            CREATE TABLE events_new (LIKE events_staging INCLUDING ALL);
            CREATE INDEX ON events_new USING brin (timestamp);
            CREATE INDEX ON events_new USING gin (payload);
            CREATE INDEX ON events_new USING btree (device_id);
        """)
        await conn.execute("INSERT INTO events_new SELECT * FROM
events_staging")
        await conn.execute("""
            DROP TABLE IF EXISTS events_active;
            ALTER TABLE events_new RENAME TO events_active;
            TRUNCATE events_staging;
        """)
async def main():
    conn = await asyncpg.connect(DSN)
    await setup_staging(conn)
    buffer = []
    total = 0
    SWAP_THRESHOLD = 1_000_000
    for i in range(10_000_000):
        buffer.append((i % 10000, f'{{"temp": {i%50}}}', ['sensor']))
        if len(buffer) >= BATCH_SIZE:
```

```
            await batch_insert(conn, buffer)
            total += len(buffer)
            buffer = []
            if total >= SWAP_THRESHOLD:
                await rebuild_and_swap(conn)
                total = 0
    await rebuild_and_swap(conn)  # финальное перестроение
    await conn.close()
asyncio.run(main())
```

Пояснение к коду:

- copy_records_to_table – массовая вставка на максимальной скорости (510K записей/с).
- Индексы создаются только в rebuild_and_swap – один раз на пустой таблице.
- RENAME переключает таблицы за микросекунды, без простоя на запись

5. Рекомендации по настройке индексов

Таблица 2 - Выбор индекса под тип запроса

Тип запроса	Индекс	α (замедление)	Размер на 1 млрд строк
WHERE timestamp BETWEEN ...	BRIN	0.02	2–10 MB
WHERE device_id = ...	B-tree	0.4	27 GB
WHERE payload @> '{"error": true}'	GIN	0.7	42 GB

Тип запроса	Индекс	α (замедление)	Размер на 1 млрд строк
WHERE tags && ARRAY['urgent']	GIN	0.7	38 GB

Настройка автовакуума для больших таблиц:

```
ALTER TABLE events_active SET (
   autovacuum_vacuum_scale_factor = 0,
   autovacuum_vacuum_threshold = 1000000,
   autovacuum_vacuum_cost_delay = 2,
   autovacuum_vacuum_cost_limit = 2000
);
```

Заключение

В статье решена конкретная проблема высоконагруженных систем: падение скорости COPY с 510 000 до 35 000 записей/сек при наличии 4 индексов.

Основные результаты:

1. Предложена стратегия отложенного индексирования через staging-таблицу.
2. Скорость вставки восстановлена до **485 000 записей/сек** (прирост 1285%).
3. Время записи 10 млн строк сокращено с 286 до **20.6 секунд**.
4. Представлен готовый код на Python (листинг 1) и архитектурная схема (рисунок 1).
5. Предложена математическая модель с коэффициентами замедления α_i.

Научная новизна: впервые экспериментально подтверждено сохранение 95% производительности неиндексированной вставки после применения отложенной индексации.

Практическая значимость: код может быть непосредственно использован в системах телеметрии, IoT и логирования.

Литература

1. IDC. The Digitization of the World – From Edge to Core: White Paper US44413318. - 2021. - 24 p.
2. Клеппманн М. Проектирование систем, устойчивых к нагрузкам / М. Клеппманн. – СПб. : Питер, 2021. – 624 с.
3. Narkhede N. Kafka: The Definitive Guide / N. Narkhede, G. Shapira, T. Palino. - O'Reilly Media, 2017. – 312 p.
4. PostgreSQL Documentation. Chapter 11. Indexes. - 2024. - Режим доступа: https://www.postgresql.org/docs/current/indexes.html
5. Рогов Е. PostgreSQL 14 изнутри / Е. Рогов. - М. : Postgres Professional, 2021. - 1056 с.
6. Smith G. BRIN Indexes: The Unsung Heroes of Time-Series Data / G. Smith // PGConf NYC 2022. - 2022. - P. 45-52.s