TimescaleDB

Pavel Stěhule
2018

O výkonu rozhodují

- Algoritmy
- Datové struktury
- 80-90 léta - vize univerzálních SQL databází
- Po roce 2000 - specializované databáze

Relační SQL databáze

- Běžně optimalizována na 70/30 SELECT/UPDATE operací
- Obecně navržené – stejné algoritmy, stejný způsob uložení se používají pro socioekonomické systémy i pro logy, naměřená data, atd. Základní datové struktury jsou halda a index (explicitně seřazená data). Nebere se v potaz, že některá data jsou přirozeně řazená (není to zaručeno).
- Běžné funkce jsou pro optimalizátor black box - a to ať zákaznické nebo vestavěné

Relační SQL databáze

- SELECT * FROM X
  WHERE EXTRACT(YEAR FROM d) = 2017
- SELECT * FROM X
  WHERE d >= ‘2017-01-01’
  AND d < ‘2018-01-01’
- Funkce nemají definovanou sémantiku. Nesleduje se monotónnost funkce. Naopak některé operátory ji definovanou mají.

Time series data (časové řady)

- Relativně bežná velkoobjemová data
  - snadno se pořizují (automaticky, sondy)
  - 100% přírůstková
  - možnost komprimace, případně aproximace
  - jednoduché úlohy často spojené s vizualizací
  - velké objemy - přírůstky v řádech MB/sec

Time series databases TSDB

- Optimalizované pro uložení časových řad
- NoSQL
- Velice rychlé dohledání dat za určité období
- Rychlý zápis
- Kompaktní formát (velké možnosti komprimace

Time series databases

- InfluxDB
- Graphite
- Druid
- Prometheus
- {{Grafana}} - vizualizace
- ??? TimescaleDB ????

TimescaleDB

- Extenze do Postgresu
  - limitováno vlastnostmi Postgresu
  - k dispozici funkcionalita Postgresu
- Rozšiřuje partitioning směrem k time series db
- Modifikuje optimalizátor - optimalizátor pozná některé funkce a některé vzory
- Mike Freedman (CTO, spoluzakladatel)

TimescaleDB partitioning

- Partition je definována časovým obdobím.
- Partitions jsou vytvářené automaticky a vkládání do partitions je také automatické. Vše je pro uživatele transparentní.
- Pohodlná funkce pro odstranění partitions v závislosti na stáří záznamů.
- Podpora Nd partitioningu - možnost separace dat podle dalších atributů (dimenzí).
- Partitioning začne být aktivní po vytvoření tzv. hypertabulky.
- Levný pohodlný partitioning - menší tabulky -> menší indexy -> rychlejší aktualizace indexů.

TimescaleDB

```
shared_preload_libraries = 'timescaledb'
CREATE EXTENSION IF NOT EXISTS timescaledb CASCADE;

CREATE TABLE xx(
  c1 TIMESTAMP NOT NULL,
  c2 INTEGER,
  ...
);

SELECT create_hypertable('xx', 'c1', 'c2', 2,
                                        create_default_indexes=>FALSE);
```

TimescaleDB optimalizace

```
SELECT date_trunc('day', pickup_datetime) as day,
        COUNT(*) FROM rides
  GROUP BY day
  ORDER BY day
  LIMIT 5;
```

- Pro PG špatně optimalizovatelné bez funkcionálního indexu nad date_trunc()
- Další index ale znamená zpomalení INSERTu

Agregace v PostgreSQL

- HashAgg - průběžně se aktualizuje hash tabulka s mezivýsledky -- HASH blokuje optimalizaci LIMITu
- GroupAgg - vždy se počítá jedna konkrétní skupina - vyžaduje seřazená data -- sort blokuje optimalizaci LIMITu

TimescaleDB optimalizace dotazu

- Víme, že partitions podle času jsou ve správném pořadí.
- Víme, že funkce date_trunc je monotónní - pokud je vstup ve správném pořadí, tak i výstup je ve správném pořadí
- Pokud máme data pro agregaci správně seřazená, tak už je nemusíme řadit a můžeme použít v tuto chvíli lacinou a nejrychlejší metodu GroupAgg
- GroupAgg je možné efektivně zastavit LIMITem

Optimalizace klauzule LIMIT

- Často používaná klauzule .. LIMIT 10
- Někdy může zrychlit provedení dotazu rychlým stopnutím výpočtu
- Nepomůže, když je nutné data dříve seřadit nebo provést hashagg (pak je nutné zpracovat všechna data).

PostgreSQL

```
Limit
  ->  Sort
        ->  HashAggregate
              ->  Seq Scan on rides
```

TimescaleDB

```
Limit
  ->  GroupAggregate
        ->  Result
              Output: (date_trunc('day'::text, rides.pickup_datetime))
              ->  Merge Append
                    ->  Index Only Scan Backward using rides_p...
                    ->  Index Only Scan Backward using _hyper_1_1_chunk_rides..
                    ->  Index Only Scan Backward using _hyper_1_2_chunk_rides..
                    ->  Index Only Scan Backward using _hyper_1_3_chunk_rides_p..
```

TimescaleDB

- Jednoduché řešení, které nemá velkou režii (CPU, pracnost, naučení) a pro určitý charakter dat prokazatelně zrychlí operace. Kompatibilní s PG.
- Zatím cca rok a půl vývoje, a zdá se to použitelné (nezasahuje citlivé vnitřnosti PostgreSQL). Velký prostor pro interní vývoj + poroste s PG.
- Plánované funkce v PostgreSQL (custom storages, clustering) jsou přínosem i pro TimescaleDB.
- Těžko porovnatelné s nativními time series databázemi (implementovanými v GO a s vlastním optimalizovaným storage).
- Může být stejné rychlé nebo rychlejší než jednodušší TSDB a pro řadu úloh dostatečně rychlé.

Bonus pro Postgres

- Open Source - GitHub
- Motivace pro custom storage nebo alespoň pro insert only storage
- Další vývojáři se znalostmi vnitřností Postgresu
- Motivace pro rozvoj extenzí - nové callbacky (planner, storage)
- Možná některé fígle by mohly být interně v PG