Caching GraphQL:

Approaches to automate caching data for GraphQL

Tanmai Gopal | @tanmaigo
Hasura

GraphQL engine

Instant realtime GraphQL on Postgres

Connect to services & get a unified GraphQL API

Runs as a docker container in your infrastructure or use hasura.io/cloud

Open-source ❤

http://github.com/hasura/graphql-engine
Query caching vs Data caching

- Cache queries:
  - Cache query execution plan

- Cache data:
  - Don’t hit the upstream data source
Query Caching

Algorithm:

- For each incoming GraphQL query, normalise it
- Hash the GraphQL query, and store the sequence of resolvers to be called in a map.
  - Use an LRU strategy to bound the size of the cache
- Run the resolvers and return data
- If the same GraphQL query or a variation comes in, do a lookup on the map and run the resolvers
- If the client supports making a query using a hash directly, even better because no normalization step is required
- graphql-jit / fastify-graphql
10x win: Pair with DB query caching (aka prepared statements)

- Instead of a pure resolver approach, consider a “pushdown” approach
- Take an incoming GraphQL query, extract the parts of it that only fetch from a single databases
- Compile that into a single DB query (along with authorization rules)
- Databases cache their query plans as well! (Prepared statements in Postgres/MySQL)
- So session variables + query variables are zoomed through directly & securely to the database

Normal: SQL query → Plan & optimise → Execute
Prepared: (SQL query name, variables) → Execute
Data Caching

- **Purpose:**
  - Reduce load on upstream services: 10k requests will be 10k requests to the database
  - Identify HOT queries and cache their results instead of straining the upstream system

- **Trade-off**
  - Consistency and stale-results :(
Data Caching is hard

- Automatically caching API calls that fetch dynamic is hard (not just for GraphQL)
- There are 2 problems to solve:
  - What to cache?
  - How do we update / invalidate the cache
Data Caching - What to cache?

Who is user-id 1?
What city are they in?
User-id 1 is in SF
Load SF restaurants

Who is user-id 2?
What city are they in?
User-id 2 is in Dublin
Load Dublin restaurants

Who is user-id 3?
What city are they in?
User-id 3 is in SF
Load SF restaurants

SF restaurant cache

Dublin restaurant cache
Data Caching - how do we invalidate & refresh the cache?

#1: Cache for 60s

SF restaurant

Is this an SF restaurant?

#2: Yes. Invalidate cache.

/restaurants?id=123
Update restaurant
3 ways to cache data

1. Before it hits the GraphQL server
2. In GraphQL resolvers
3. At the model level (integrated with logic to fetch the data for a particular model)
1. Cache before the GraphQL server

- Similar to caching GET requests with a CDN
- API server doesn't know about caching at all

- **Algorithm:**
  - Look at the incoming query’s identifier (or normalise and check identifier)
  - See if this query is cacheable (cache list, @cached directive on the client-side)
  - Load data from a cache instead of running resolvers.
    - If data is not available, async-ly populate the cache

- **Caveats:**
  - Only works if you know that the result of the query doesn’t depend on the identity of the user.
  - Eg: public APIs
Cache full API call by treating it like *public* data

- `/restaurants?city=SF`
  - User-id: 1 (SF)
  - No dependency on user identity. Load from cache.

- `/restaurants?city=Dublin`
  - User-id: 2 (Dublin)
  - No dependency on user identity. Load from cache.

- `/restaurants?city=SF`
  - User-id: 3 (SF)
  - No dependency on user identity. Load from cache.
2. Cache at GraphQL resolvers

- Cache inside the GraphQL resolvers

- **Algorithm**:
  - Inside a resolver, create a cache key based on the upstream database query or API call
  - For any execution of the resolver, load the data from a cache using the cache key
    - Or populate the cache if there’s a cache miss

- **Caveats**:
  - Hitting the cache for every resolver. N+1? Cache needs a data-loader also?
  - Potentially a lot of repeated code if multiple resolvers are fetching from the same model
  - Hard to automate
Fetch from cache in resolver instead of fetching from source.

/restaurants
User-id: 1

Restaurants resolver
User-id 1 is in SF
Load SF restaurants from cache or DB

SF restaurant cache

/restaurants
User-id: 2

Restaurants resolver
User-id 2 is in Dublin
Load Dublin restaurants from cache or DB

Dublin restaurant cache

/restaurants
User-id: 3

Restaurants resolver
User-id 3 is in SF
Load SF restaurants from cache or DB
3. Cache using model-level rules

- **Algorithm:**
  - Each model should have *declarative authorization & relationship rules*
  - Resolvers fetch data from a generic model data fetching layer
    - Data fetching layer embeds the authorization rules automatically.
    - *Knowing what to cache is not at the resolver level*
  - When a query comes in, analyse the authorization rules of all the models that will be fetched in the query to determine its dependency on the user identity
  - For multiple user identities, we can determine if the query will result in fetching the same data
  - Use simple data caching at the full-query level (like in approach #1)
Cache-key includes the user’s “group”. Cache full query.

- /restaurants
  - User-id: 1
    - User-id 1 is in SF
      - Use (SF, query) cache key and load from cache
- /restaurants
  - User-id: 2
    - User-id 2 is in Dublin
      - Use (Dublin, query) cache key and load from cache
- /restaurants
  - User-id: 3
    - User-id 3 is in SF
      - Use (SF, query) cache key and load from cache
Caching on Hasura Cloud

- LRU cache
- @cached directive. Client controls tolerance for stale data.

Use a combination of 2 strategies automatically.

1. **Use #1:**
   a. Determine if query is independent of user identity

2. **Use #3:**
   a. If data is from a database, use #3 approach
   b. If data is from an API source where business logic is not known, use #1 if applicable.