Design and Implementation of Real-Time Management System Architecture based on GraphQL

Ying Guo, Fang Deng and Xiudong Yang

Department of Computer Science, Beijing University of Posts and Telecommunications, room 513, Student Apartment 4, No.10 Xitucheng Road, Haidian District, Beijing, China
Email: 584038547@qq.com

Abstract. In times of big data, the increasing growth of data sets is a big challenge for both data storage and data access, especially the management system with high real-time requirements. For real-time management system, there are some performance problems in the traditional Web service architecture based on REST style. In this paper, it proposes to establish GraphQL server in the real-time subsystem and integrate it into the existing systems. Through this architecture, the client can obtain the specified data from the server side, thus reducing the number of network requests and avoiding excessive capture or capture of data. Experiments are carried out in a real-time monitoring system. The experimental results show that the proposed architecture reduces the transmission data packets by 57.38% in real-time monitoring and management system, and reduces the average time of data transmission by 8.9%.

1. Introduction

With the continuous development of Web, the development mode of Web service architecture is also evolving. From the Front-end and back-end mixing development pattern to the Front-end and back-end separation, the Web is developing at the fastest speed in history. Nowadays, the main way of interaction between clients and servers is mainly based on REST style service architecture. These services can meet the interoperability requirements within a partly range. However, with the expansion of the scale of data application, the REST style Web service architecture has gradually revealed the problems of insufficient scalability, large complexity and low performance, and it is difficult to overcome these problems due to the constraints of its architectural style, especially in the system with high real-time requirements. In order to solve these problems, Facebook's open source GraphQL language specification has entered the field of vision. A new conceptual framework of data access interface type on Web is provided. It is being used more and more widely.

2. Related work

2.1. REST Overview

REST software architecture was first proposed by Dr. Roy Thomas Fielding in 2000. REST is the abbreviation of Representational State Transfer. REST itself is just an architectural style for distributed hypermedia system. REST is a set of design concepts based on the characteristics and requirements of network software, which aims to reduce the complexity of the development, improve the scalability of the system and reduce the communication load. The REST Web service architecture is shown in Figure 1.
Figure 1. REST web service architecture.

REST service architecture puts forward several main design criteria for Web application:

- all things on the Internet are abstracted as resources.
- each resource corresponds to a unique resource identifier.
- operate the resources through the generic connector interface.
- various operations on resources will not change the resource identifier.
- all operations are stateless.

The "resource identifier" in the second point is used on Web to refer to the URI, and the "generic connector interface" in the third point is used on Web to refer to the standard actions of GET, POST, PUT, DELETE, HEAD in the HTTP protocol, which are identical on the whole Web.

The above constraints are the key to the success of the REST architecture. Now, because of the REST web service architecture is simple and easy to use, it has been widely used by service providers and most of the software developer community.

2.2. Problems

In typical REST system, objects can be addressed through URI and interact with verbs in HTTP protocol. Although REST is simple, but in REST service architecture, a way of URI representing a resource will lead to the coupling of resource types. With the scale of application has expanded, the amount of data is large and the data type is rich, the complexity and quality requirements of the system are increased, and the shortcomings of the REST style Web architecture are also exposed, mainly in the following points:

- when the requirements or data changes, the server-side needs to build a new interface to adapt to the changes, and the continuous addition of the interface will cause the continuous growth of the server-side code, which causes the server expansion and maintenance difficulties.
- with the change of system requirements, fields and additional data are always added to the REST endpoint. However, the old clients will also receive these additional data, because the data acquisition specification is encoded on the server-side rather than on the client-side. Therefore, the request URL of each resource will return the redundant data.
- in a REST service architecture, we need to send HTTP requests to get all the data we want. Getting complex object requires multiple trips between client and server to render a single view. For mobile applications running under variable network conditions, these multiple round trips are not ideal.

In particular, in the system with high real-time performance requirements, the simplicity of REST not only lead to some limitations, but also reduce the performance under the environment of poor network environment. Therefore, in the real-time system with high performance, we need to design a higher performance architecture to improve the performance.
3. Overview of the GraphQL language
In recent years, JavaScript has developed rapidly. The emergence of Node.js technology brings new ideas for Web development mode. Node.js not only shares the development language with the front-end, but also reuses the code with the front-end. and also give JavaScript the ability to execute on the server side. In 2015, The open source GraphQL language specification is published by Facebook company. The design goals of the language are as follows:

- compared with REST Web service model, it can reduce unnecessary costs by reducing the number of unnecessary data transfers and individual queries.
- reduce the possibility of error caused by part of the invalid query on the client.
- support changing data models without API version control.

The interaction mode between different clients and GraphQL server is shown in Figure 2.

![Figure 2. GraphQL server interaction mode.](image)

Figure 2 shows that different client applications can send requests to the GraphQL server, verify and collect information from different data sources to execute queries.

GraphQL interaction mode contains the following two messages:

- a request, which must be provided in a well-defined query language and send as plain string to the single GraphQL endpoint
- a response of the GraphQL server specified as a JSON document.

A GraphQL query is hierarchical and structured, which can provide a single entity with all the relationships from a API endpoint. The structure of the query represents the expected data to be returned. In general, every level of GraphQL query corresponds to a specific type. They can be nested, or even recursively. The resulting document is a set of entities that specify their relational entities in the type system. This clarifies the meaning of the name of GraphQL, that is, entities and relationships can be regarded as a graph.

4. Proposed Architecture
Because of the high performance requirements of the real-time system, therefore, the establishment of a new Web architecture based on GraphQL, that is, integrating the old system or the third party system with the GraphQL server. The architecture is as shown in Figure 3. For real-time system, create new API and query data through generated GraphQL mode to improve real-time system performance.

When different clients send requests to the real-time subsystem, the request is sent to the GraphQL server. When the GraphQL server receives the query, it will parse it through schema and resolver, schema contains types and relationships, through the resolver retrieves the required data from the database of the connected data or some integrated API. Connector is a connection set for different data sources. The rest of the requests get the data in the way of Restful API.
5. Validation and Results

5.1. Testing Criteria
The performance difference of the architecture is mainly through comparing the different architectures of real-time subsystem. The performance will be evaluated by measuring latency and data volume.

The experiment is carried out on the same computer, that is to say, the server and the client are on the same computer and communicate with each other.

Latency or response time is often used to measure the performance difference of data flow on the network. In this experiment, response time is calculated from the time when the client request is sent out, until it returns to the client's response. The latency will be measured in the way of running two different timers in Javascript, The first timer will be started when the request will be made on the client and the second timer will be started when the response was been received by the client. The delayed data recorded in the experiment include the number of requests, the start time, the end time, the time difference and the endpoint URL.

The data volume can be measured by measuring the actual size of the response packets. When data are received on the network as HTTP packets, we measure and compare the actual sizes of these packets through the Wireshark packet capture tool.

The final latency and data volume will be represented in charts. They will compare and clearly show different results in API.

5.2. Testing method
Filling and querying the entire database is not the scope. The research only needs part of the database, so it will reduce the part of the database used for testing. Specifically the tables goods, records,
record_has_materials, and materials. These are the tables that will be populated with test data and the queries will be performed on these corresponding tables. As shown in Figure 4.

The data is generated randomly. These tables will fill in the test data, and queries will be executed on these corresponding tables.

After the request connection is established, the 500ms between each request is set. Each endpoint of each test is requested 100 times.

The following cases were tested. The endpoints for the experiment are four.
- /graphql (GraphQL)
- /goods/record/$id (REST, Get a certain good from an record id)
- /records/good/$id (REST, Get a certain record from a good id)
- /materials/record/$id (REST, Get /materials based on the record id)

GraphQL is built on a single endpoint. It can query all fields contained in the request through a query. However, REST API needs multiple endpoints to complete the experiment.

![Figure 4. Database diagram.](image)

5.3. Results
The query for getting the corresponding information about an record, including the goods that owns the record, and the materials that are a part of the record was carried out on both GraphQL and REST. From the data that was gathered it can be seen that indeed GraphQL is quicker than REST. It is as shown in Figure 5.
The collected data points were cleared for over 200 milliseconds. When the response time is over 50 milliseconds, the value is removed fairly, and no controversial technology is supported.

Query the size of the package by the same experiment. Figure 6 shows the difference of the total packet size between GraphQL and REST. The biggest packet size by far is the return of the REST service when it fetches the material information, because the selected recorder ID has several related materials. This includes name, price and producer. They need quite a number of bytes. Figure 6. shows the size of the packets that each interface calls, because REST has to send three API requests to the server to get all the corresponding information.

If all the REST calls are combined together to form the whole structure of the data, the packet size will look like in Figure 7. This shows that by selecting the specified data, we can indeed reduce the total data size. GraphQL decrease packet size as well as bandwidth usage in whole.
6. Conclusion
From the above experiments, it can be concluded that when extracting from complex query structure, showing GraphQL to be 57.38% faster than the RESTful service. The response time GraphQL is 8.9% shorter than RESTful service. The average response time shown in Figure 8 is as shown below. Based on the above two points, the new architecture in the monitoring and management system can reduce the number of queries to reduce the server's pressure. And, by selecting data more accurately, it can reduce the size of the total packets sent between the client and the server, so that the loading time is faster. The application of new architecture in real-time monitoring and management system can improve the performance of Web site.

7. References
[1] Jamil, Hasan M. "Design of Declarative Graph Query Languages: On the Choice between Value, Pattern and Object Based Representations for Graphs." IEEE, International Conference on Data Engineering Workshops IEEE, 2012:178-185.
[2] Bryant, Mike. "GraphQL for archival metadata: An overview of the EHRI GraphQL API." IEEE International Conference on Big Data IEEE, 2017:2225-2230.
[3] Hartig, Olaf, and J. Pérez. "An Initial Analysis of Facebook's GraphQL Language." Alberto Mendelzon International Workshop on Foundation of Databases and the Web 2017.
[4] Nogatz, Falco, and D. Seipel. "Implementing GraphQL as a Query Language for Deductive Databases in SWI-Prolog Using DCGs, Quasi Quotations, and Dicts." 234(2017):42-56.

[5] Zhao, Peixiang, and J. Han. On graph query optimization in large networks. VLDB Endowment, 2010.

[6] Andrea Vázquez-Ingelmo, Juan Cruz-Benito, and Francisco J. García-Peñaavo. "Improving the OEEU's data-driven technological ecosystem's interoperability with GraphQL." The, International Conference 2017:1-8.