Performance Measurement of GraphQL API in Home ESS Data Server

1st Eunggi Lee
Energy IT Convergence Research Center
Korea Electronics Technology Institute
Seongnam-si, Republic of Korea
leg429@keti.re.kr

2nd Kiwoong Kwon
Energy IT Convergence Research Center
Korea Electronics Technology Institute
Seongnam-si, Republic of Korea
kiwoong.kwon@keti.re.kr

3rd Jungmee Yun
Energy IT Convergence Research Center
Korea Electronics Technology Institute
Seongnam-si, Republic of Korea
yunjm@keti.re.kr

***Abstract*—This paper implemented an API server using GraphQL to transport only the data necessary for user of home ESS(Energy Storage System) management application. In addition, this paper measured the response time and mobile data consumption for the request of GraphQL API client and the performance was compared with the legacy API (RESTful API) server.**

***Index Terms*—home energy storage system, API, GraphQL, RESTful**

I. Introduction

In the HEMS(Home Energy Management System) application, the users can set up dashboard of the application to display only the information what they want to see. According to the users' setting, the application decides which ESS status data to request from the API server. In existing legacy system using RESTful API, the application had to send dozens of request messages to authenticate and get data from the server. This process required a lot of data resources and time in a mobile environment.

In this paper, we implemented GraphQL API server to solve above problem. GraphQL is a data query language developed by Facebook. GraphQL is not depend on a specific database or storage engine, and it can be connected to various types of database systems. It can even connect to other API systems and website. Type system of GraphQL makes this possible [1].

In this paper, the construct of the home ESS data server was described. In addition, the form of request and response message for ESS status data using GraphQL was described. In Chapter 3, the performance of the GraphQL API server implemented in this paper was measured and compared with the legacy server using RESTful API. In conclusion, future works are explained.

II. Construct of Home ESS Data Server

The home ESS data server implemented in this paper is constructed as shown in the figure 1. All data stored in the ESS DB is transported through the GraphQL server at the request of the GraphQL Client.

A. ESS DB

All status data generated in ESS in a second is stored in ESS DB. ESS DB used My-SQL. There are

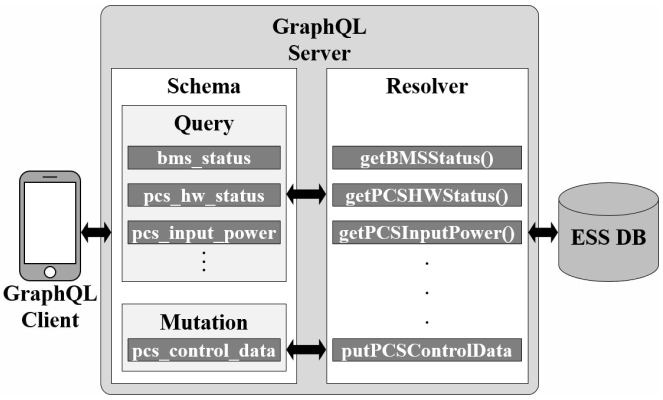

Fig. 1. Construct of GraphQL server

'ess_info', 'bms_status', 'pcs_hw_status', 'pcs_input_power', 'pcs_output_power', 'pcs_overcurrent', 'pcs_overvoltage', 'pcs_temperature', 'state_of_charge', 'pcs_control_cmd' and 'sw_reset' tables. Figure 2 shows the relationship and columns of five of these tables. The ess_info table contains data such as unique identification information, specifications, location, and network information of ESS. The 'bms_status' table contains BMS mode and error data such as cell voltage deviation, low voltage and high voltage. The 'pcs_hw_status' contains error data of PCS input voltage and current sensor, dc-link voltage sensor, and battery voltage sensor. The 'pcs_input_power' table contains data measured by the PCS input power sensor. The 'pcs_control_cmd' table contains command record data for controlling PCS. All of these tables have 'ess_id' and date-time as keys.

B. GraphQL Server

Figure 1 shows the GraphQL API server that is largely divided into two components, the schema and the resolver. Schema has the query type that queries data from the server and a mutation type that inputs or modifies data in the server. The query type has the numerous object types as the fields that is able to query ESS status such as 'bms_status', 'pcs_hw_status', 'pcs_input_power'. On the other hand, the mutation type has the object types as the fields that insert and

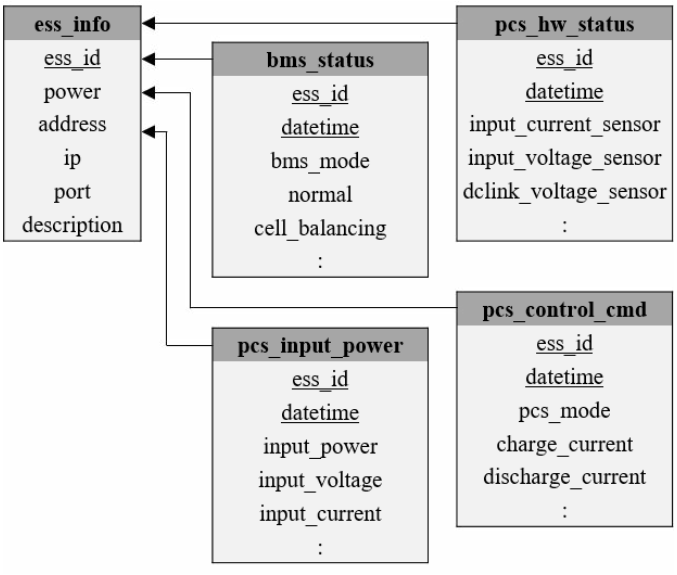

Fig. 2. Schemas of ESS DB

update the data to the ESS DB such as 'pcs_control_data', 'sw_rest' for managing ESS.

In this paper, we defined the field types of the query and the mutation the same as the table schema of ESS DB to request all ESS status data. The GraphQL clients can request the desired data to the server by selecting only required object types and fields according to these defined schema, thus they can receive the only desired data with desired form. Additionally, by defining the arguments per each object type, they can receive the specified arguments if they request data. Figure 3 shows the example of GraphQL schemas. As shown in this figure, the proposed system enables the GraphQL clients to request only the data of the specific ESS by using the 'tags' argument or the data of the specific time duration by using 'from' and 'to' arguments. These schemes run by the clients and call the proper resolver matching with each field. In order to get and process data, there are the variety of resolvers such as 'getBMSStatus()', 'getPCSHWStatus()', 'getPCSInputPower()' by matching with schema types. Each resolver retrieves the ESS status data in the ESS DB via SQL. If the argument exists, the data is specified using 'WHERE' SQL syntax. The ESS status data is immediately forwarded to the GraphQL, or it is refined by the resolver sometimes.

C. Legacy System (RESTful API)

Before using GraphQL, the existing HEMS system used RESTful API. RESTful API expresses all ESS status data as resources. The resources can be accessed through the path specified by URI. In addition, CRUD (Create, Read, Update, Delete) operations can be applied to each resource through HTTP methods such as POST, GET, PUT, and DELETE. Figure 4 shows some examples of ESS status RESTful API. The first GET operation is an API that requests the 'ess_info' table which ESS information is stored in. The second GET

```
type bms_status {
   time: String
   bms_mode: Int
   normal: Int ...}
type AllBMSStatus {
   tags: ess_info
   data: [bms_status] }
type Query {
   bms_status(tags: ess_input, from: String, to: String): [AllBMSStatus] }
```

Fig. 3. Example of GraphQL schemas

operation is an API that requests the most recently stored 'normal' resource in the 'bms_status' table. The third GET operation has the same role as the second API. However it can be filtered using 'since' and 'until', 'size' filters to specify time duration and size of requested resource with a question mark. The fourth operation is an API that creates data in the 'pcs_control_cmd' table. In this way, HEMS clients can access all resources of ESS DB using RESTful APIs [3].

III. Performance Measurement

In this paper, an experiment was conducted to compare the response time and response size of the ESS status data API server using GraphQL and the existing ESS RESTful API server. The experiment was conducted in the following order [2].

First, selects randomly a certain number of resources among various resources of ESS DB.

Second, according to the selected resource, each request message of GraphQL and RESTful API is created.

Third, record the response time and response size by requesting the created request message to each API server. Fourth, repeat the first to third process 30 times and calculate the mean and standard deviation.

Fifth, repeat the above process increasing the number of selected resources in the first process.

The specifications of the server where the experiment was conducted and the important modules (versions) of the API package are shown in Table 1.

As a result of experimenting in the above order, we could obtain graphs like figure 5 and figure 6. The x-axis of the graph represents the number of random resources requested to the server. The y-axis of figure 5 and figure 6 shows the average response time and size for 30 requests to the API

```
GET  /ess
GET  /ess/:ess-id/bms-status/normal
GET  /ess/:ess-id/bms-status/normal
       ?since='yyyy-mm-dd'&until='yyyy-mm-dd'&size='1000'
POST /ess/:ess-id/pcs-control-cmd
```

Fig. 4. Example of RESTful APIs

TABLE I
THE SPECIFICATIONS OF THE SERVER

CPU	Intel(R) Core(TM) i5-7260U CPU @ 2.20GHz
Memory	16GB
OS	Ubuntu 16.04.6 LTS
DB	My-SQL(14.14)
Server Dependencies	node.js(12.14.0), express(4.17.1), graphql(14.6.0)
Client Dependencies	python(3.5.6), requests(2.24.0), numpy(1.19.1)

server, respectively. Blue and orange bars (or lines) represent GraphQL and RESTful, respectively. The black range line displayed above each bar in figure 5 represents the standard deviation.

In the case of the response time in figure 5, when the number of resources is less than 20, the average response time of RESTful is slower than GraphQL. However, when the number of resources is 20, it is reversed. After then, the average response time of GraphQL is faster than RESTful. These experimental results are the result of the difference in characteristics of GraphQL and RESTful API. In the case of GraphQL, if the requested resources are in one ESS DB table, it can get the requested resources with a single query. However, in the case of RESTful API, even if the requested resources exist in one table, the number of queries must be increased as much as the number of resources unless the API is modified. Therefore, the increase in response time according to the number of resources is higher in RESTful API than GraphQL.

However, in case of standard deviation, RESTful API is almost constant, whereas GraphQL is not. In the case of GraphQL, it is difficult to ensure consistency of response time according to the number of random resources. Because even if the number of resources is the same, the number of ESS DB queries varies according to the location stored in the ESS DB table.

In the case of the response size in figure 6, the increase in the response size of RESTful according to the number of resources is higher than that of GraphQL. In the case of GraphQL, if randomly selected resources are in one table, they can be expressed together in one schema in the response message. However, in the case of RESTful, even if the randomly selected resources are in one table, there is redundant data in response messages because resources must be requested individually.

IV. Conclusion

In this paper, we implemented an API server for requesting status data of home ESS using GraphQL. In addition, we compared and analyzed response time and response size with the existing RESTful API server. As a result, we found that it is better to use RESTful API for simple services with fewer resources to request at one time, but it is better to use GraphQL for complex services with a large number of resources to request at once.

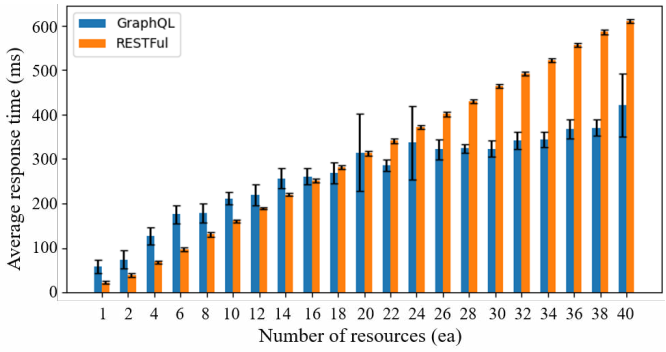

Fig. 5. Response time comparison between GraphQL and RESTful API

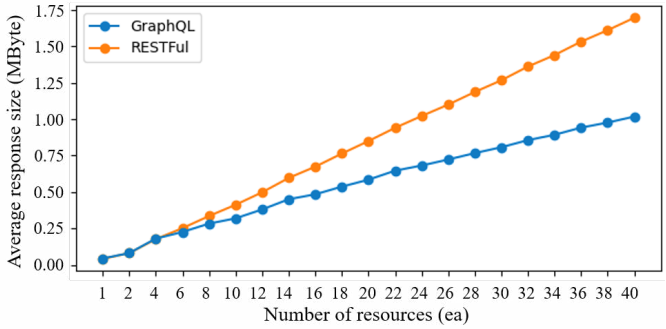

Fig. 6. Response size comparison between GraphQL and RESTful API

In the future work, we will compare the performance of GraphQL and RESTful API in other environment using other database system than the relational database system (My-SQL). And we will be able to suggest an appropriate API system according to the situation of the service.

Acknowledgment

This work was supported by the Korea Institute of Energy Technology Evaluation and Planning(KETEP) and the Ministry of Trade, Industry & Energy(MOTIE) of the Republic of Korea (No. 201927101015E).

References

[1] GraphQL: A query language for APIs. (n.d.). Retrieved Aug 23, 2020, from https://graphql.org/

[2] D. A. Hartina, A. Lawi and B. L. E. Panggabean, "Performance Analysis of GraphQL and RESTful in SIM LP2M of the Hasanuddin University," 2018 2nd East Indonesia Conference on Computer and Information Technology (EIConCIT), Makassar, Indonesia, 2018, pp. 237-240.

[3] D. M. Rathod, M. S. Dahiya and S. M. Parikh, "Towards composition of RESTful web services," 2015 6th International Conference on Computing, Communication and Networking Technologies (ICCCNT), Denton, TX, 2015, pp. 1-6.