Poster: Prototype of Configurable Redfish Query Proxy Module

Chanyoung Park[†], Yoonsue Joe[†], Myounghwan Yoo[‡], Dongeun Lee[§], Kyungtae Kang[†]
[†]*Hanyang University,* [‡]*XSLAB Inc.,* [§]*Texas A&M University-Commerce*
{chanyoung, suejoe, ktkang}@hanyang.ac.kr, yoo@xslab.co.kr, dongeun.lee@tamuc.edu

***Abstract*—Redfish is a next-generation API standard for the management of data center infrastructures. This rich API can flexibly obtain data using a query string from the client side. However, this feature is optional and not fully supported by many services. We implemented a prototype Redfish query processing module on Nginx, a well-known open source web server. The Redfish query processing module can run with a proxy module and work with any server-side or client-side applications. Additionally, our prototype implementation can be *configured* to properly utilize queries, which are supported on a backend server, and improve performance. Our implementation was evaluated on an OpenBMC server and a mockup server and showed potential for performance improvement.**

***Index Terms*—redfish, query, proxy, bmc**

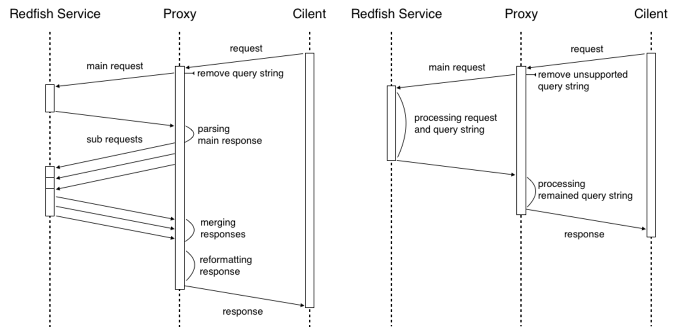
I. Introduction

Distributed Management Task Force (DMTF)'s Redfish is a standard API that has over 2000 properties and is built using building blocks that are widely used in modern web services designed to manage modern data centers. The specification also includes optional but useful features, such as standardized client-side queries [1].

Despite its usefulness, the query feature is not well supported by many services. The feature could be easily supported by implementing a query translation layer, assuming that the service uses a single data source with a powerful tool to process query language, such as a relational database. However, in the baseboard management controller (BMC) software, developers must implement query functions independently to retrieve data from various sources (i.e., sensor daemons). This is a reason for why many Redfish services selectively support only the queries required to manage a few target resources.

We propose a web server module that runs with a proxy web server and handles queries on behalf of the Redfish server. It can be configured so that the processing of queries is partly (or entirely) delegated to the backend server if the server can handle them. In addition, if the proxy is located close to the Redfish server, a performance improvement can be expected as the proxy needs to communicate with the Redfish server repeatedly, in contrast to clients directly sending multiple requests for query processing.

This work was supported in part by the NRF (National Research Foundation of Korea) Grant funded by the Korean Government (NRF-2017-Global Ph.D. Fellowship Program), and in part by the Next-Generation Information Computing Development Program through the NRF funded by the Ministry of Science and ICT (No. 2017M3C4A7083676).



(a) Redfish service's target resource handler does not support any query. (b) Redfish service's target resource handler supports `$expand` query.

Fig. 1. Query processing flow examples.

II. Design and Implementation

While the Redfish has many schemas (i.e., data tables), they are all managed by a single BMC. Therefore, processing a query outside the Redfish service, rather than directly modifying its data layer, would help achieve acceptable performance, not to mention the benefit expected from the proxy. As such, we support query processing using a module of the web server independently from the Redfish service. Utilizing a proxy server handling Redfish queries on the same machine as or a machine close to the Redfish service creates the following three advantages.

1) Any server or client can use this function without modifying its program, and modules can be added or removed at any time. 2) It is implemented as a module of a mature open-source web server; therefore, it is stable and may have a synergistic effect with many other modules. 3) A performance improvement can be expected in comparison to similar programs on the client side.

Figure 1 shows the query processing flow in our prototype proxy module. When a request containing a query string is received from the client, the proxy checks the uniform resource identifier first and identifies the queries that the target resource handler (running on the backend server) can take care. If there are no queries supported by the target resource handler, then the main request is forwarded to the Redfish service, after removing the unsupported query string, as shown in Figure 1a.

After receiving and analyzing the response for the main request, the module immediately receives data via the subrequest function of Nginx to access additional resources. After all the

TABLE I
EXPERIMENTAL SETUP

#	Server	← Network →	Proxy	← Network →	Client	Processing OH	Network OH
LAN-P/NP	OpenBMC's bmcweb	LO, SSL	Nginx/-	LAN, SSL	DMTF's Redfishtool	Exists	Low
VPN-P/NP	OpenBMC's bmcweb	LO, SSL	Nginx/-	VPN, SSL	DMTF's Redfishtool	Exists	High
MOCK-P/NP	DMTF's mockup	LO	Nginx/-	LAN	DMTF's Redfishtool	Minimum	Low

P: Proxy. *NP:* No proxy. *LO:* Loopback. *OH:* Overhead. *LAN:* Same subnet, connected with two L2 switches.
Server hardware: 64bit ARM Cortex-A53, 16GB RAM, 1Gbps Ethernet. *Client hardware:* x86_64 i5-6600, 64GB RAM, 1Gbps Ethernet.

TABLE II
TEST REQUESTS

#	URL	Subrequests
A	/AccountService	0
B	/AccountService/Accounts?$expand=.	2
C	/AccountService/Roles? $expand=.&$filter="Members/Id" eq "Callback"	0 or 4
D	/Managers?only	1

received data are merged, the data are reformatted to fit the query, and the final response is sent back to the client. If there are query strings that the target resource handler can manage, then processing of those queries is delegated to the backend server, as shown in Figure 1b.

All the queries recommended in the specification document, except the `excerpt` query, can be processed by combining subrequests and reformatting their responses. To support `excerpt`, the query module must understand the Redfish service's schema files. This can be implemented either by dynamically analyzing the Redfish's schema document in module loading or by statically analyzing it in module compilation. Currently, we implement the queries `$expand`, `only`, and incomplete `$filter` as a proof of concept.

III. Preliminary Experimental Results

We evaluated the prototype of the proposed query processing proxy module under different processing and network overheads to observe how they affect performance. Table I summarizes our experimental setup. The proposed scheme was implemented in the Nginx module and set up as a reverse proxy. Two Redfish services were used: the OpenBMC environment, which retrieves data from the actual data layer, and the static mockup data server provided by DMTF. The `redfishtool`, a command-line tool provided by DMTF, was used as the client program. Because the Redfish service is not usually exposed to a WAN, the network environment is reconfigured into a LAN environment and a VPN environment.

Table II shows the requests used in the experiment and the number of generated subrequests in processing each original request. These requests were selected from commands that exist in the `redfishtool` and can be reconstructed using Redfish queries. Since all features have not been implemented yet, those requests are mainly composed of `$expand` queries that generate many subrequests. We modified the `/AccountService/Roles` resource handler operating in the backend server to support the `$expand` query and configured the proxy to acknowledge it. Therefore, in scenario C, the number of subrequests is 0 and 4 when the query is going through and bypassing the proxy, respectively. Each of the

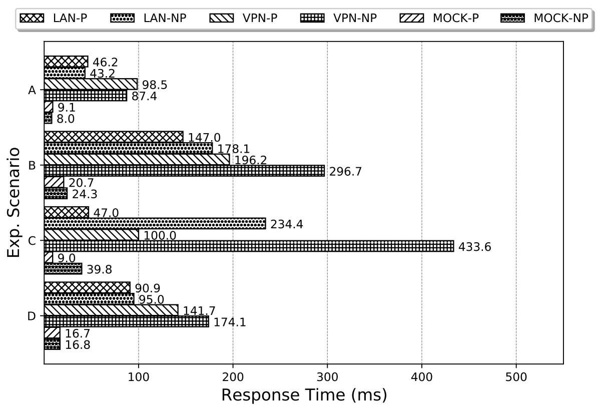


Fig. 2. Query processing performance.

requests were iteratively processed 10 times, and the average result in each case was recorded.

Figure 2 shows the average response time versus different request scenarios for different experimental environments. In scenario A, which has no subrequests, all non-proxy requests performed slightly better than proxied requests by 7%–13%. However, the others having subrequests showed performance improvement proportional to the number of subrequests or the network delay. In scenario C, with leveraging of the query processing by the backend server, LAN-P decreased the response time by approximately 80% compared with LAN-NP. In the case of VPN-P and VPN-NP, the response time was reduced by approximately 76%. Even in the case of MOCK-P and MOCK-NP, which only transmitted static files, performance improvement was observed. Therefore, we can reasonably expect our module to improve performance.

IV. Conclusions

We proposed a prototype configurable Redfish query processing module to run with a proxy server on BMC. It can fully utilize all the queries supported by the backend server to improve query processing efficiency. Preliminary evaluation demonstrated improved system performance in terms of query response time. We expect our approach to allow many Redfish services to support the query functions recommended in a specification document. We will complete this study using a fully supported query module, and we also plan to support multiple BMC nodes.

References

[1] DMTF, "Redfish Specification." (2019). Accessed: Aug. 21, 2020. [Online]. Available: https://www.dmtf.org/sites/default/files/standards/documents/DSP0266_1.9.0.pdf