Enhanced Architecture for User Search Result

Contents

Purpose

This technical document describes the changes done in the architectural implementation of query/string search to improve the performance of user search result.

Background

With the current implementation of the architecture, when the user search requests are not high, the response time is found good. But when the number of such requests is high, the response time also gets high. This leads to slow performance of the system.

To understand the complete scenario, let's delve into the current architecture and the applied architectural changes to improve the search process.

Current Implementation

There are three services that interact with each other to get the user search result to the user interface.

According to Figure 1:

1. Service3 receives a request from the user interface to search for a string that is in plain text format over the RESTful API call.
2. Service3 requests Service1 to provide output based on the search string, and Service1 responds back with an SQL query.
3. Service3 sends this SQL query to Service2 and requests the output string. Service 2 responds back with the search result (text).
4. Now, Service3 publishes the received search result on the user interface.

All this communication happens over RESTful HTTPs calls.

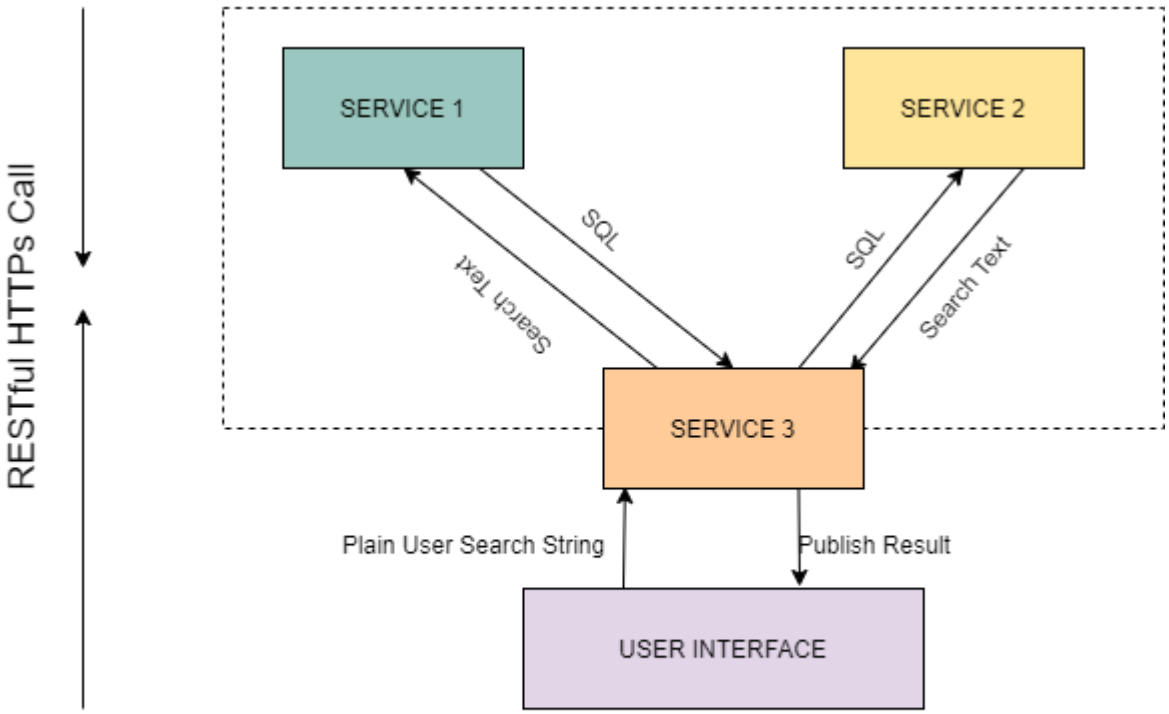

Figure 1: Current Implementation

Real Time Scenario:
The consumer opens the ABC application (user interface) and searches for the user contact details. Service3 receives a request which delegates the request to Service1 to get the query that is used in Service2 to fetch the contact details. The SQL query sent to Service2 provides contact details to Service3 in a text format. This is published to the ABC application.

Challenges

- Due to the communication between three services over HTTPs interface, it possesses a risk of introducing high response time when the traffic is too high.
- HTTPs delays the response time but there is no flexibility in Service3 to use another interface.

Proposed Implementation

The challenges seen in the current implementation are addressed here by tweaking the architecture:

- Introduced BACH as a microservice.
- Changed the protocol from HTTPs to RPC (Remote Procedure Call).

Now, according to the proposed implementation:

1. Service3 receives request from the user interface to search for a string over the REST API, which this service sends to the BACH in a text format.
2. BACH sends this text request to Service1 to provide output based on the search string. Service1 responds back to BACH with an SQL query.
3. BACH sends the SQL query (received from Service1) to Service2 and requests the output data (similar to the **Current Implementation)**. This time, the communication happens over RPC.
4. BACH provides the search result to Service3 which is then published on the user interface over REST.

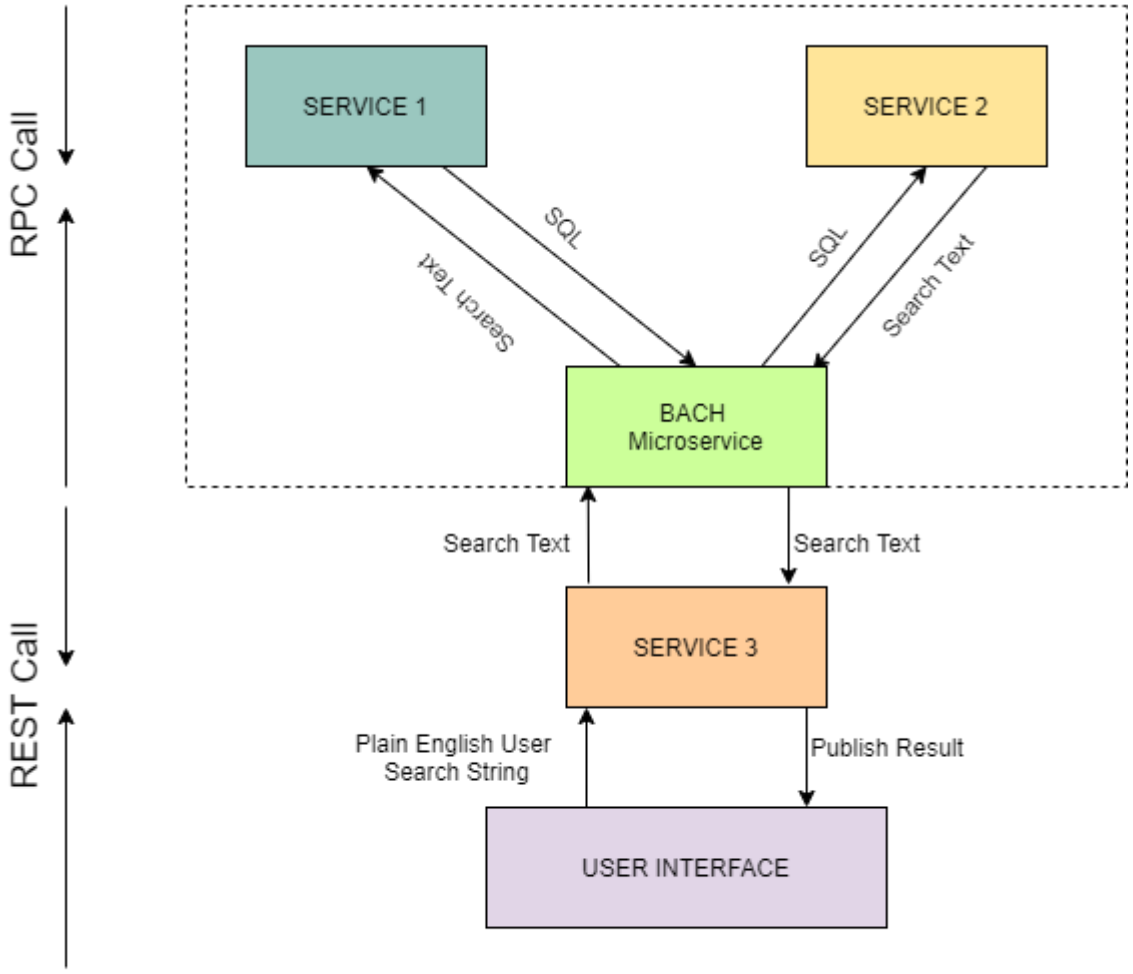

Figure 2: Proposed Implementation

What changed the response time?

- RPC improves performance when there are a high number of requests coming to system. It omits multiple protocol layers to reduce overall response time.
- Instead text, RPC uses binary data which makes communication more compact and efficient.
- The implementation of RPC requires the introduction of a microservice called BACH as it has lesser number of functions to be executed. For example:
    - To get the request over REST API, make an RPC call to Service2
    - To get the response over RPC from Service1/2, and to respond to Service3 over the REST API
- The microservice is easily scalable when the volume of requests increases over time.

Conclusion

The integration of BACH microservice with RPC call improves the overall performance of the user search result as it reduces the response time. Such changes are done in the existing architecture for back-end communication.