Artifact

Communication between MicroServices Async vs Sync Comparing HTTP vs AMQP-RPC

by Radoslav Radev

Advanced Software Engineering Semester 6

15/03/2024

ver. 1.0

Synchronous vs. Asynchronous Request-Response

Synchronous Request-Response

In synchronous communication, a client (which can be another microservice) makes a request to a server (the service it wants to communicate with) and waits for the response before continuing its operation. The client is blocked and cannot perform other tasks during this wait time.

Characteristics:

- **Blocking**: The client waits for the server's response before proceeding.
- **Direct**: The client knows which service to call and waits for its response.
- **Simple**: Easier to understand and implement, especially for straightforward request-response interactions.

Example communications that are considered synchronous:

- **HTTP**
- **gRPC**
- **GraphQL**

Asynchronous Request-Response

Asynchronous communication allows a client to make a request to a server and continue its operation without waiting for a response. The client can handle the response at a later time, whenever it arrives, without being blocked in the meantime.

Characteristics:

- **Non-blocking**: The client can continue other tasks without waiting for the server's response.
- **Decoupled**: The client and server are more decoupled, with less direct dependency on each other's availability.
- **Complex**: More complex to implement, especially when tracking and handling responses or errors.

Disadvantages:

- **Latency Management**: While it improves scalability, managing the latency of responses and ensuring timely processing can be challenging.

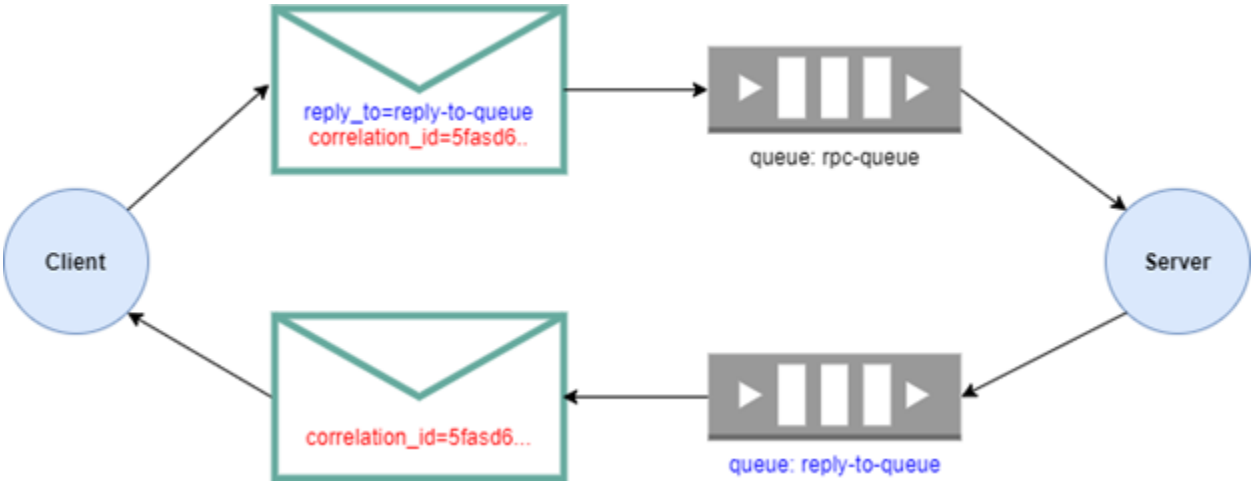
Example communications that are considered asynchronous :

- **Message Brokers (AMQP)**
- **GraphQL**
- **Webhooks Concept (http)**

Implementing True Async Request-Response communication:

To implement truly asynchronous communication we will need separate request and receive channels:

Diagram utilizing a message broker:



A simple example of the concept utilizing a WebHook:

```
//SERVICE 1
public class SomeService
{
    public async Task RequestSomeData()
    {
        using (var httpClient = new HttpClient())
        {
            var jsonRequestData = """{"SomeParametherName":"someRequestSpecificData"}""";
            var content = new StringContent(jsonData, Encoding.UTF8, "application/json");
            httpClient.PostAsync("http://service2/someEndpoint", content); //not awaiting
        }
    }
    public async Task  doSomethingWithTheDataWeRequested()
    {
    }
}

[Route("api/[controller]")]
[ApiController]
public class WebhookController(SomeService _someService) : ControllerBase
{
    [HttpPost]
    public IActionResult Receive([FromBody] dynamic data)
    {
       _someService.doSomethingWithTheDataWeRequested(data)
        return Ok();
    }
}
```

```
//SERVICE 2
[Route("api/[controller]")]
[ApiController]
public class RequestConsummerController(SomeServiceProvidingTheData _someServiceProvidingTheData) : ControllerBase
{
    [HttpPost]
    public IActionResult Receive([FromBody] dynamic requestData)
    {
        _someServiceProvidingTheData.HandleRequest(requestData)
            return Ok();
    }
}

public class SomeServiceProvidingTheData
{
    public async Task HandleRequest(requestData)
    {
        //get the requested data from some repo or other service
        var responce =  getData(requestData)

        using (var httpClient = new HttpClient())
        {
            var content = new StringContent(responce, Encoding.UTF8, "application/json");
            httpClient.PostAsync("http://service1/theWebhook", content); //not awaiting
        }
    }
}
```

What do we want to achieve while using Async:

We want to be able to execute other tasks in a non-blocking way while waiting for the response.

Can we achieve that while using synchronous protocols in an async way?

Example:

```
Public  async Task<VideoRecommednations> GetHotVideosViaHttpGet()
{
   using (var _client = new HttpClient())
   {
     string recommendationRequestJson = JsonConvert.SerializeObject(new { TopN = topN });
     var recommendationRequestContent = new StringContent(recommendationRequestJson, Encoding.UTF8, "application/json");
     var recommendationResponse = await _client.GetAsync("http://recomendationAlgo.local/hotVideos", recommendationRequestContent);
     //In areal world usecase  will use Conul to get the ip of an availbale instance of the recommendationAlgo service
     var recommendationResponseContent = await recommendationResponse.Content.ReadAsStringAsync();
     return JsonConvert.DeserializeObject<VideoRecommednations>(recommendationResponseContent);
   }
}

public async Task<List<VideoDTO>> DoSomethingWithHotVideos(int topN)
{
  var results = await Task.WhenAll(GetHotVideosViaHttpGet(), doSomeOtherWork(), GetSomeOtherData());
  //we can do something to combine the data
  //for example we can  return a combined result , or query oru own repo to get data
}
```

Most of the time when we need a Request-Response we need to perform something with the response data immediately, so we need to wait for it to come back.

Implemented Example of using RabbitMQ with MassTransit`s Request-Reply feature:

```
1 usage   Radoslav Radev *
public async Task<List<VideoDTO>> GetRecommendedVideos(Guid userId,VideoCategory category,int topN )
{
    var videoRecommendationsResponse = await _videoRecommendationsRequestClient.GetResponse<RecommendationVideoResponse>( values: new
    {
        UserId = userId,
        Category = category,
        TopN = topN
    }); // Task<Response<...>>
    var videos :List<Video> = await _videoRepository.GetVideosByVideoIds(videoRecommendationsResponse.Message.VideoIds);
    return videos.Select(_videoMapper.VideoToVideoDto).ToList();
}
```

Use Case: We need to provide the client with video recommendations(Title, Descr, Thumbnail, WT, etc), to do so we need to first get some video recommendations for our client, (our recommendation service only has information about the videoIDs).
In this implementation: inside our video library service, we make a request to the recommendation service, to get a list of recommended videos, and then we query our own database and return the videos to our client.

So in this implementation, the video library gets the video recommendations on behalf of the client and constructs the response (the client gets his data in only one request).

Why don't we make the client get the list of recommended videoIDs himself, and then provide it back to us, so we can provide him those videos?
In our case, the client is a web application, and the traffic needs to go through the internet (the client could be on the other side of the planet).
When making the request internally we use our extremely fast Kubernetes network, we are also flexible to use any communication method in our own network, in contrast, the client sits behind a firewall and is in a sandboxed browser environment, so our communication is mostly limited to something carried over HTTP.

Using Async Event-Based Communication:

When we don't need a response we can utilize our message broker, which will ensure our message gets delivered to an appropriate recipient/s

Example:

```
//before that we uploaded the video file and the thumbnail to the NFS storage bucket
UploadVideoRequest video = new UploadVideoRequest()
{
    VideoId = newVideoId,
    Title = videoMetadata.Title,
    Description = videoMetadata.Description,
    Category = videoMetadata.Category,
    VideoUri = videoDirectory + _pathSepator + "playlist.m3u8",
    ThumbnailUri = videoDirectory + _pathSepator + "thumbnail.jpg",
    uploadedByAccoutId = Guid.Parse(accId),
    UploadDateTime = DateTime.Now,
    IsPublic = false
};
await _publishEndpoint.Publish<UploadVideoRequest>(video);
await _publishEndpoint.Publish<RenderVideoRequest>(message: new RenderVideoRequest() { VideoId = newVideoId, VideoUri = nonRenderedVideoPath });
```

In this example, our Upload service sends a message to our video library, so it can update its database with the new video.
It also sends a message to the render service, so it can start rendering the video.
When the render service renders the video it will send a message to the video library, to update the video as public, so it becomes available to clients.

the _publishEndpoint.Publish is an async Task method, that completes when we send the message to the broker, we are not required to await the message sending.
Since we ourselves are at the end of an async Task method (void), we can await them, this is a good practice because we might want to extend our upload implementation and monitor when all actions in the upload operation are complete (do something after that).

When to Use Asynchronous and When to Use Synchronous Communication:

Synchronous communication is ideal for operations where the client needs an immediate response to continue. This is common in user-facing operations where delays in response could negatively affect user experience.

Asynchronous communication is highly scalable and efficient for handling operations that do not require immediate responses. It allows the system to process other tasks while waiting for the response, optimizing resource utilization.

As we saw in the examples we can use sync communication protocols in an async way, usually the client for a synchronous protocol doesn't even offer synchronous methods(such is the case with .NET`s HttpClient).
Making the majority of our code Async (even typical void methods) offers us the control to await when we decide, and the information when a certain task completes.

Comparing RabbitMQ`s RPC to HTTP POST

“Remote Procedure Call (RPC) and REST are two architectural styles in API design. APIs are mechanisms that enable two software components to communicate with each other using a set of definitions and protocols. Software developers use previously developed or third-party components to perform functions, so they don’t have to write everything from scratch. RPC APIs allow developers to call remote functions in external servers as if they were local to their software. For example, you can add chat functionality to your application by remotely calling messaging functions on another chat application. In contrast, REST APIs allow you to perform specific data operations on a remote server. For example, your application could insert or modify employee data on a remote server by using REST APIs.” - (AWS, n.d.)

“

- REST, RPC - architecture patterns, AMQP - wire-level and HTTP - application protocol which run on top of TCP/IP
- AMQP is a specific protocol when HTTP - general-purpose protocol, thus, HTTP has damn high overhead comparing to AMQP

- AMQP nature is asynchronous where HTTP nature is synchronous
- both REST and RPC use data serialization, which format is up to you and it depends of infrastructure. If you are using python everywhere I think you can use python native serialization - `pickle` which should be faster than JSON or any other formats.
- both HTTP+REST and AMQP+RPC can run in heterogeneous and/or distributed environment

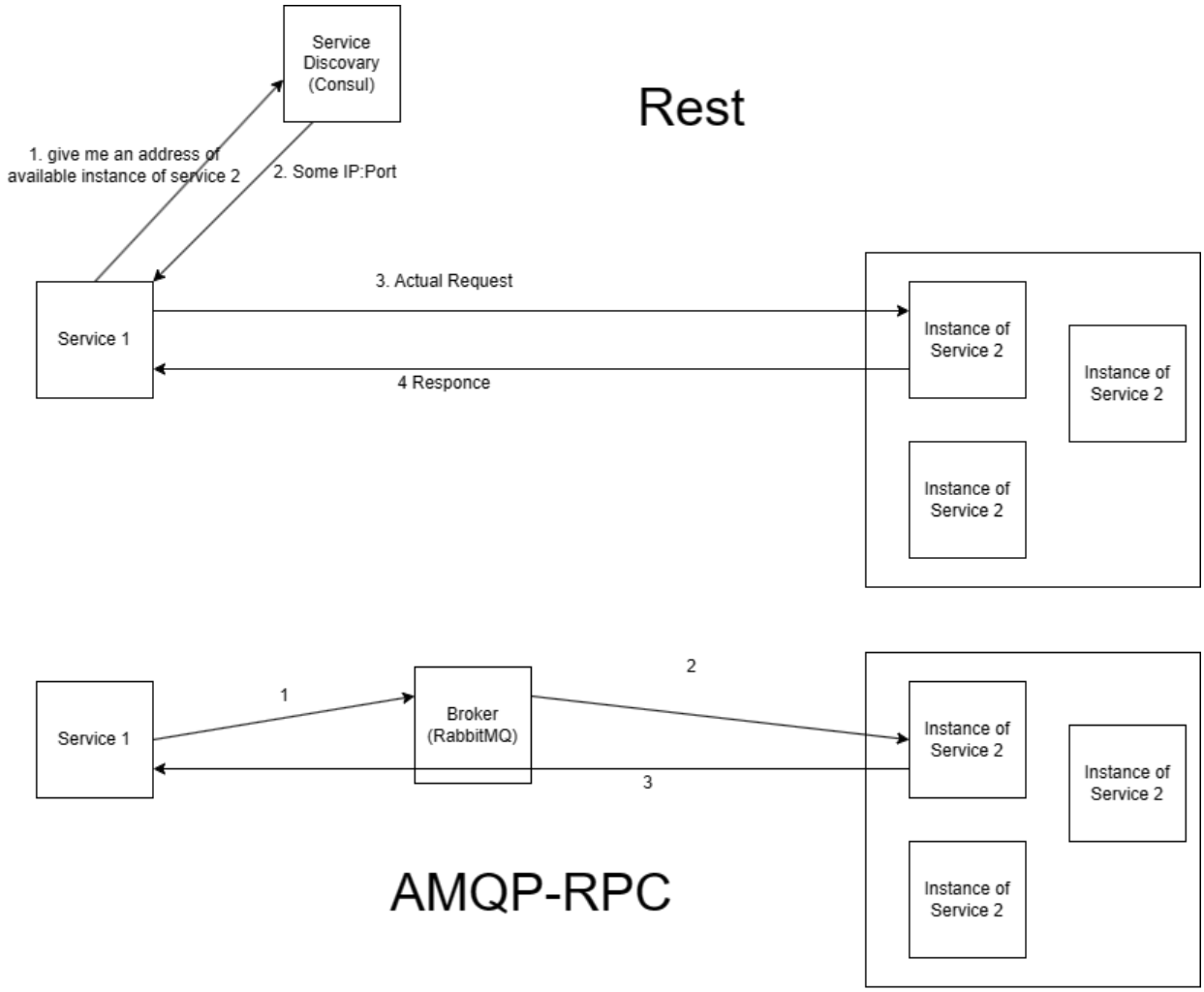
  " (pinepain, 2013)

"So if you are choosing what to use: HTTP+REST or AMQP+RPC, the answer is really subject of infrastructure complexity and resource usage. Without any specific requirements, both solutions will work fine" (pinepain, 2013)

"The determining factor is whether you need some kind of broadcast or routing to multiple subscribers. If it's point-to-point, HTTP will be much simpler to work with, even if async. Use AMQP when you would otherwise reinvent features like tracking subscribers, topic- or header-based routing, etc." (Bill Schneider, 2017)

I wasn't able to find a definitive answer if one is faster than the other, both AMQP-RPC and HTTP use TCP/IP as the underlying protocol, and both serialize the data, however in our context of distributed system with multiple instances of dedicated services, using the broker will be theoretically faster, but both protocols have theoretically the same performance

Diagram:



AMQP-RPC with RabbitMQ: While the response does go through the broker, it’s directed specifically to the consumer who made the request, leveraging the broker's internal mechanisms for efficient delivery. This setup uses the broker for routing but minimizes overhead, making it seem almost as if the response is going directly to the consumer.

Conclusion:

When it comes to a Request-Reply situation where we need our data for the next step, I will go with AMQP-RPC, it offers more features and has certain advantages over HTTP.
In my particular case, it`s also the simpler solution.

References:

AWS. (n.d.). *RPC vs REST - Difference Between API Architectures*. AWS. Retrieved March 15, 2024, from https://aws.amazon.com/compare/the-difference-between-rpc-and-rest/

Bill Schneider. (2017, July 17). *AMQP vs HTTP*. DEV Community. Retrieved March 15, 2024, from https://dev.to/fedejsoren/amqp-vs-http

pinepain. (2013, 6 25). *Service Oriented Architecture - AMQP or HTTP*. https://stackoverflow.com/questions/16838416/service-oriented-architecture-amqp-or-http