Ensuring Effective Server Communication in Mobile Applications

Deepak Mariyappa
Ravi Peravali
Abstract

The server communication process in mobility applications should remain asynchronous and perform independently. It must also have control of the application to ensure that measures can be taken to mitigate risks.

This whitepaper presents a server communication process which enables parallel processes to run without any of them affecting the application’s performance.

Introduction

The growing popularity of mobile applications (apps) is rewriting the way consumers act, interact, purchase, sell and search. This has resulted in a proliferation of mobile applications. To attract and retain end users, mobile app companies need to consistently deliver uninterrupted user experiences 24/7.

Mobile application performance is determined to a large extent by networks and servers. By enabling seamless communication between the application and the server (HTTP / custom), companies can mitigate performance issues. They need to also ensure that parallel operations are not blocked.

To summarize, communication process should remain asynchronous during the following three stages:

> The facilitation within the application to communicate
> The communication itself, and
> The post-processing of the result.
Given below is a pictorial representation of a typical server communication along with its corresponding issues:

In this scenario, the blocks that need to communicate should:
- Prepare the request using required data
- Communicate with the server and wait for its response
- Post process, extricate useful information to consume the same
- Handle any error situations that can arise from the server communication.

**Challenges with this approach**

- The server communication code is contained inside the Server Communicator (block) and cannot be reused for other server communications
- It will contain code specific to the server request which will not be generic in nature
- Post processing logic is buried inside the block making it unscalable. Any new responses will require modifications and result in a break in the functionality process
- Thread safety and asynchronous two-way communication
- Difficulty in supporting multiple versions of Server APIs which increases proportionately with multiple servers and versions.
It knows the process to create a request, parameters to be used, headers to be created, etc. It assimilates responses and post-process, extracts information to be consumed by the application. It brings up custom error responses and shows the method to handle the same. The `Action` does not send out the request to the server but acts as the bridge between the two. Any new communication to the server will result in a new action class, keeping the existing ones untouched and unaffected.

Every request that goes to the server is independent and the response is tightly coupled with that request. Hence, a smaller processing unit that deals with a single request and its response, is easier to handle.

**Actions**

An Action class is an independent component which knows the vital points about a particular network action:

- It knows the process to create a request, parameters to be used, headers to be created, etc.
- It assimilates responses and post process, extracts information to be consumed by the application.
- It brings up custom error responses and shows the method to handle the same.
- The `Action` does not send out the request to the server but acts as the bridge between the two.
- Any new communication to the server will result in a new action class, keeping the existing ones untouched and unaffected.
The queue has the option of throttle control to define how many actions are performed in parallel. This control resides within the Network and is dependent on it. The Network engine based on throttle settings picks up Action objects for processing. It uses an interface method to prepare a request. The action subclass which knows the specifics of that action, prepares the request and feeds data with generic parameters to the Engine. The Engine then packages all this information into a URL Request and sends it to the remote server. Neither the action nor the action creator is isolated from the server location.

The action class does not send out the request. The Network Engine does it. Only the Network Engine knows how to communicate with the server, hence isolating this information from the rest of the application helps it to switch from one server to another or to change the network protocol (for ex. use https) while the application remains unaffected.

The action class does not send out the request. The Network Engine does it. Only the Network Engine knows how to communicate with the server, hence isolating this information from the rest of the application helps it to switch from one server to another or to change the network protocol (for ex. use https) while the application remains unaffected.

The action class does not send out the request. The Network Engine does it. Only the Network Engine knows how to communicate with the server, hence isolating this information from the rest of the application helps it to switch from one server to another or to change the network protocol (for ex. use https) while the application remains unaffected.

The action class does not send out the request. The Network Engine does it. Only the Network Engine knows how to communicate with the server, hence isolating this information from the rest of the application helps it to switch from one server to another or to change the network protocol (for ex. use https) while the application remains unaffected.

The action class does not send out the request. The Network Engine does it. Only the Network Engine knows how to communicate with the server, hence isolating this information from the rest of the application helps it to switch from one server to another or to change the network protocol (for ex. use https) while the application remains unaffected.

The action class does not send out the request. The Network Engine does it. Only the Network Engine knows how to communicate with the server, hence isolating this information from the rest of the application helps it to switch from one server to another or to change the network protocol (for ex. use https) while the application remains unaffected.

The action class does not send out the request. The Network Engine does it. Only the Network Engine knows how to communicate with the server, hence isolating this information from the rest of the application helps it to switch from one server to another or to change the network protocol (for ex. use https) while the application remains unaffected.

The action class does not send out the request. The Network Engine does it. Only the Network Engine knows how to communicate with the server, hence isolating this information from the rest of the application helps it to switch from one server to another or to change the network protocol (for ex. use https) while the application remains unaffected.

The action class does not send out the request. The Network Engine does it. Only the Network Engine knows how to communicate with the server, hence isolating this information from the rest of the application helps it to switch from one server to another or to change the network protocol (for ex. use https) while the application remains unaffected.

The action class does not send out the request. The Network Engine does it. Only the Network Engine knows how to communicate with the server, hence isolating this information from the rest of the application helps it to switch from one server to another or to change the network protocol (for ex. use https) while the application remains unaffected.

The action class does not send out the request. The Network Engine does it. Only the Network Engine knows how to communicate with the server, hence isolating this information from the rest of the application helps it to switch from one server to another or to change the network protocol (for ex. use https) while the application remains unaffected.

The action class does not send out the request. The Network Engine does it. Only the Network Engine knows how to communicate with the server, hence isolating this information from the rest of the application helps it to switch from one server to another or to change the network protocol (for ex. use https) while the application remains unaffected.

The action class does not send out the request. The Network Engine does it. Only the Network Engine knows how to communicate with the server, hence isolating this information from the rest of the application helps it to switch from one server to another or to change the network protocol (for ex. use https) while the application remains unaffected.

The action class does not send out the request. The Network Engine does it. Only the Network Engine knows how to communicate with the server, hence isolating this information from the rest of the application helps it to switch from one server to another or to change the network protocol (for ex. use https) while the application remains unaffected.

The action class does not send out the request. The Network Engine does it. Only the Network Engine knows how to communicate with the server, hence isolating this information from the rest of the application helps it to switch from one server to another or to change the network protocol (for ex. use https) while the application remains unaffected.

The action class does not send out the request. The Network Engine does it. Only the Network Engine knows how to communicate with the server, hence isolating this information from the rest of the application helps it to switch from one server to another or to change the network protocol (for ex. use https) while the application remains unaffected.

The action class does not send out the request. The Network Engine does it. Only the Network Engine knows how to communicate with the server, hence isolating this information from the rest of the application helps it to switch from one server to another or to change the network protocol (for ex. use https) while the application remains unaffected.

The action class does not send out the request. The Network Engine does it. Only the Network Engine knows how to communicate with the server, hence isolating this information from the rest of the application helps it to switch from one server to another or to change the network protocol (for ex. use https) while the application remains unaffected.

The action class does not send out the request. The Network Engine does it. Only the Network Engine knows how to communicate with the server, hence isolating this information from the rest of the application helps it to switch from one server to another or to change the network protocol (for ex. use https) while the application remains unaffected.

The action class does not send out the request. The Network Engine does it. Only the Network Engine knows how to communicate with the server, hence isolating this information from the rest of the application helps it to switch from one server to another or to change the network protocol (for ex. use https) while the application remains unaffected.

The action class does not send out the request. The Network Engine does it. Only the Network Engine knows how to communicate with the server, hence isolating this information from the rest of the application helps it to switch from one server to another or to change the network protocol (for ex. use https) while the application remains unaffected.

The action class does not send out the request. The Network Engine does it. Only the Network Engine knows how to communicate with the server, hence isolating this information from the rest of the application helps it to switch from one server to another or to change the network protocol (for ex. use https) while the application remains unaffected.

The action class does not send out the request. The Network Engine does it. Only the Network Engine knows how to communicate with the server, hence isolating this information from the rest of the application helps it to switch from one server to another or to change the network protocol (for ex. use https) while the application remains unaffected.

The action class does not send out the request. The Network Engine does it. Only the Network Engine knows how to communicate with the server, hence isolating this information from the rest of the application helps it to switch from one server to another or to change the network protocol (for ex. use https) while the application remains unaffected.

The action class does not send out the request. The Network Engine does it. Only the Network Engine knows how to communicate with the server, hence isolating this information from the rest of the application helps it to switch from one server to another or to change the network protocol (for ex. use https) while the application remains unaffected.

The action class does not send out the request. The Network Engine does it. Only the Network Engine knows how to communicate with the server, hence isolating this information from the rest of the application helps it to switch from one server to another or to change the network protocol (for ex. use https) while the application remains unaffected.

The action class does not send out the request. The Network Engine does it. Only the Network Engine knows how to communicate with the server, hence isolating this information from the rest of the application helps it to switch from one server to another or to change the network protocol (for ex. use https) while the application remains unaffected.

The action class does not send out the request. The Network Engine does it. Only the Network Engine knows how to communicate with the server, hence isolating this information from the rest of the application helps it to switch from one server to another or to change the network protocol (for ex. use https) while the application remains unaffected.

The action class does not send out the request. The Network Engine does it. Only the Network Engine knows how to communicate with the server, hence isolating this information from the rest of the application helps it to switch from one server to another or to change the network protocol (for ex. use https) while the application remains unaffected.

The action class does not send out the request. The Network Engine does it. Only the Network Engine knows how to communicate with the server, hence isolating this information from the rest of the application helps it to switch from one server to another or to change the network protocol (for ex. use https) while the application remains unaffected.

The action class does not send out the request. The Network Engine does it. Only the Network Engine knows how to communicate with the server, hence isolating this information from the rest of the application helps it to switch from one server to another or to change the network protocol (for ex. use https) while the application remains unaffected.

The action class does not send out the request. The Network Engine does it. Only the Network Engine knows how to communicate with the server, hence isolating this information from the rest of the application helps it to switch from one server to another or to change the network protocol (for ex. use https) while the application remains unaffected.

The action class does not send out the request. The Network Engine does it. Only the Network Engine knows how to communicate with the server, hence isolating this information from the rest of the application helps it to switch from one server to another or to change the network protocol (for ex. use https) while the application remains unaffected.

The action class does not send out the request. The Network Engine does it. Only the Network Engine knows how to communicate with the server, hence isolating this information from the rest of the application helps it to switch from one server to another or to change the network protocol (for ex. use https) while the application remains unaffected.

The action class does not send out the request. The Network Engine does it. Only the Network Engine knows how to communicate with the server, hence isolating this information from the rest of the application helps it to switch from one server to another or to change the network protocol (for ex. use https) while the application remains unaffected.

The action class does not send out the request. The Network Engine does it. Only the Network Engine knows how to communicate with the server, hence isolating this information from the rest of the application helps it to switch from one server to another or to change the network protocol (for ex. use https) while the application remains unaffected.

The action class does not send out the request. The Network Engine does it. Only the Network Engine knows how to communicate with the server, hence isolating this information from the rest of the application helps it to switch from one server to another or to change the network protocol (for ex. use https) while the application remains unaffected.

The action class does not send out the request. The Network Engine does it. Only the Network Engine knows how to communicate with the server, hence isolating this information from the rest of the application helps it to switch from one server to another or to change the network protocol (for ex. use https) while the application remains unaffected.

The action class does not send out the request. The Network Engine does it. Only the Network Engine knows how to communicate with the server, hence isolating this information from the rest of the application helps it to switch from one server to another or to change the network protocol (for ex. use https) while the application remains unaffected.

The action class does not send out the request. The Network Engine does it. Only the Network Engine knows how to communicate with the server, hence isolating this information from the rest of the application helps it to switch from one server to another or to change the network protocol (for ex. use https) while the application remains unaffected.
It is possible that multiple entities might send out the same request to the server or a similar request comes in when one is already in process. Sending the second redundant request will be a waste of bandwidth. To avoid this, an Observer store is maintained that maps a particular request (with the same set of parameters and sent to the same destination) to an array of observers. Upon successful reception of response and when processing is complete, these observers are intimated with a result that they can consume.

This Observer store could either be contained in the Network engine or as a separate singleton object. The emphasis here being - synchronized accessibility to all action classes.

The advantage of this approach is that actions can be run through a simulator and tested while the remote server is being readied for implementation. This extra bandwidth is useful to prevent timeline issues.

The simulator can be implemented to specifically provide the response needed to test positive and negative flows.
Conclusion

It is possible to fit in new server engines to support new servers, without disturbing the eco system. By following the suggested method, every communication block is independent and works in tandem. This pattern enables flexible solution implementation without affecting the functioning of other modules and improves the performance of the app.

The decision of slow network and no network behaviors is in the hands of the action classes and hence can be changed without affecting other components in the eco system. Thread safety is taken care of and server communications can happen in parallel. This makes it easier to test for both positive and negative flows.

Summary

Thus, it is possible to run parallel server communication processes without any of them affecting the application’s performance. By following the suggested method, it is possible to ensure that every communication block is independent and yet works in tandem. In this process, action objects, even though specific to a single task, can be modified without disturbing the server communication eco system. The capacity to isolate action objects that makes this process beneficial in agile development (the preferred model for mobile app development).

*Disclaimer: The process described in this whitepaper is generic in nature and hence can be used in any development environment like iOS or Android.
Tavant Technologies is a specialized IT solutions & services provider that leverages its expertise to provide impactful results to its customers. We have leveraged our unrivaled capabilities and domain insights to create game changing results for leading businesses across chosen industry micro-verticals. We are known for our long-lasting customer relationships, engineering excellence and passionate employees. Founded in 2000, we are headquartered in Santa Clara, California and service customers across North America, Europe, and Asia-Pacific.

About The Authors

**Ravi Peravali** - Director, Media and Entertainment practice at Tavant.
He works with a number of leading media companies and the likes to develop Web and mobile solutions.

**Deepak Mariyappa** - Technical Architect working on Mobile Application development for the Media and Entertainment practice at Tavant.