Mobile Code

Group: GetTogether
Michael Kang (mmkang), Nerman Nicholas (nnichola), Jade Pearce (japearce)

Overview

Mobile code is an architectural style where code is moved from one machine to another across the network and executed remotely. There can be two basic reasons for why this is done:

- The client has resources, but does not have the code
- The client has the code, but not the resources

In the first scenario, the client would request the code from the server and then run it locally. It follows that in the second scenario, the client would send the code to the server, the server runs the code, and then sends the results back to the client.

Components & Definitions

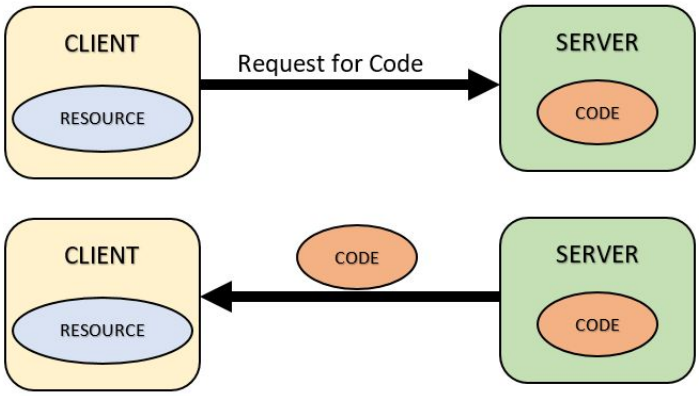
Resources - Can be physical (processing power) or digital (compiler/interpreter).
Client - The user for which the code is intended for. Can also be called the initiator.
Server - The host that either evaluates code received from the client, or pushes code requested by the client.
Code - Self explanatory.

Applications

As stated in the overview, there are two basic reasons for why one would adopt the mobile code architecture. They can formally be defined as following:

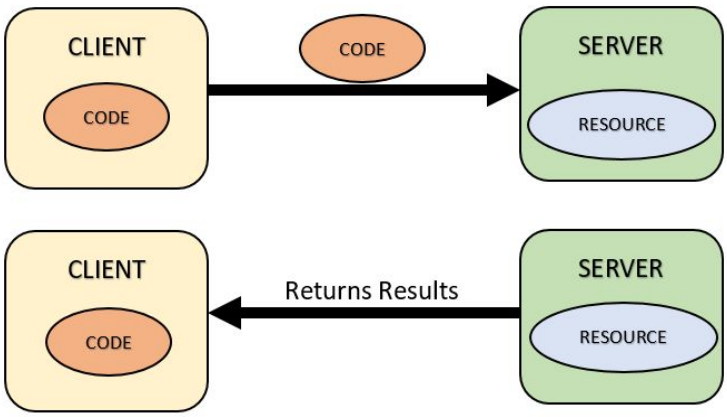
> Code on demand - client has resources, lacking code, does process code
> One major reason for this is simply because it would be infeasible for the application to be stored locally on every machine. For example, webpages on the internet are hosted on their respective servers until a client connects to the website and thus initiates a demand for the code. The server will send the code

for the website to the client's browser, which will then execute the code and display the webpage. This can be extended to sophisticated webapps that have all kinds of scripts and code stored on the server. This allows clients to use applications without being required to download everything locally.



Remote execution - client has code, lacking resources, does not process code

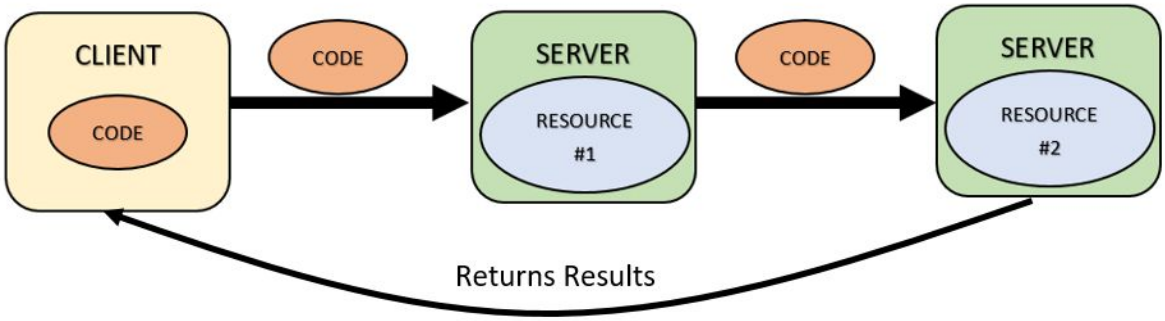
The client could be lacking the actual processing power or may just not have the required interpreter(s) for the code to function. For extremely complex problems, code may need to be executed by a more powerful machine/computing grid in order to be evaluated in a reasonable timeframe. Additionally, it could also be a matter of convenience: for example, you want to compile a LaTeX document but don't have a compiler on your machine. You can use the plethora of online LaTeX compilers to execute the code for you and then download the result file.



There is also a third general application of the mobile code architecture style:

Mobile Agent - client has code, lacking resources, does processes code

In a mobile agent application, the client may not have all the necessary resources to execute the code that it has. The needed resources may be located on a remote server. In order to execute the code, the client will pass it to different hosts with the required resources until the code is finished executing. Mobile agents are useful in applications that must collect information over a network, modify other programs or even report on problems on the network. Due to the autonomous executing nature of mobile agents, they share many traits with worms and computer viruses; as a result, this architecture may be exploited for malicious purposes. An example of when a mobile agent may be used is in price or spec comparisons on shopping websites.



Change Resilience

The mobile code architecture is quite open to changes and updates. Since it is simply the movement of code from one machine to another, any changes made should be cleanly represented without any problems, provided that all resources are up to date. The only potential issue would be change that drastically alters the resource requirements of the application. If the client and/or server can no longer process the code with the resources provided due to the change, then adjustments would need to be made, either to add more resources or to optimize the code.

Negative Behaviour

The biggest issue with mobile code is security. Since code is being sent across the network, there is no guarantee that the code is secure/clean. It is very possible that some servers will look to send malicious code with the intent to infect clients, and vice versa.

NFPs

Mobile code supports:

- Scalability - Since code is being executed remotely, computing power is being shared and thus growing users are easily accommodated.
- Portability - Clients do not need to have all the necessary components stored locally; they can request/push the code needed to run the application.
- Efficiency - Code can be run across multiple machines as required if a single machine does not have enough resources.

Mobile code inhibits:

- Security - Code can come from unknown sources of questionable cleanliness.

Presentation Scenarios

1. Code on Demand
    a. Scenario: You want to make some coffee and you have everything you need to make the coffee (coffee maker, sugar, water, milk) except for the coffee beans. You go to your local grocery store and buy the coffee beans from them. Once you buy the coffee beans you can use all your resources to brew up some coffee.
    b. Explanation: In this scenario you are the client with all of the resources (coffee maker, sugar, water, milk) but you do not have the coffee beans which is the code. By going to the grocery store and buying coffee

beans, you are requesting the server (grocery store) for the code (coffee beans).

2. Remote Evaluation
    a. Scenario: You want to make some coffee but you only have some coffee beans. You ask your neighbour if they can make some coffee for you using your beans with their coffee brewing resources. They prepare you the coffee and give that back to you.
    b. Explanation: In this scenario, you are the client and have the code (coffee beans). You make a request to your neighbour (server) who has the coffee maker, milk, sugar, and water (resources) to make your coffee and return that to you.
3. Mobile Agent
    a. Scenario: You want to make some coffee and you have the coffee beans and some of the resources to make it (coffee maker, water) but lack other resources (milk, sugar). You brew up some black coffee and then ask a neighbour if they can finish up your coffee using their resources. One neighbour has milk and sugar and is able to complete your coffee the way you like it and returns that to you.
    b. Explanation: In this scenario, you are the client and you have the code (coffee beans) but only some of the resources (coffee maker, water). You make a request to your neighbour (server) that has the missing resources (milk, sugar) to help complete your coffee.
4. Inhibits Security
    a. Scenario: You want to make some coffee and you have all the resources to make some coffee (coffee maker, sugar, water, milk) but you do not have the coffee beans. You ask your neighbour if you could use some of their coffee beans and then give you some of theirs. You immediately start brewing the coffee and after tasting it notice that the coffee beans your friend gave you was expired.

b. Explanation: In this scenario, you are the client and you have all the resources (coffee maker, sugar, water, milk) but you do not have the code (coffee beans) to brew the coffee. You make a request to the server (neighbour) for the code (coffee beans) and start making the coffee (executing the code) without realizing that the code (coffee beans) was expired (infected), thus inhibiting security.

Resources

- https://www.lysator.liu.se/~matpe/publish/agark.pdf
- http://searchitoperations.techtarget.com/definition/on-demand-computing
- http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.550.9365&rep=rep1&type=pdf