Handoff between Proxies in the Proxy-based Mobile Computing System

Azade Khalaj
Department of Computer Science
Western University
London, Ontario, Canada
Email: akhalaj@csd.uwo.ca

Hanan Lutfiyya
Department of Computer Science
Western University
London, Ontario, Canada
Email: hanan@csd.uwo.ca

***Abstract*—Research activities in the mobile computing field aim to find solutions for achieving the smoother access to remote online resources, such as cloud services, from client application found on mobile devices. The limited capabilities of mobile devices and also the unreliable condition of wireless environment are sources of challenges in achieving the mentioned goal. We have proposed a proxy-based mobile computing system that offloads compute-intensive tasks from mobile devices to proxies. Additionally, to handle the variation in connection quality, the proxies are chosen and changed dynamically to provide a better service quality for the client application. The result of our experiments shows the effectiveness of our approach in choosing the appropriate proxy and switching to a new proxy when it is needed.**

I. Introduction

The use of mobile devices (e.g., smartphones, and tablet PCs) has tremendously increased. Applications targeted at mobile devices have become abundant with applications in various categories e.g., entertainment, health, games, business, social networks.

Many smartphone applications are implemented by communicating with a remote service. The remote service provides a well-defined API that can be used by smartphone applications e.g., weather applications can use of several remote services that collect weather data make this data available through a well-defined API. This works well for applications with relatively little data to be transferred. This model of mobile device development is not suitable for applications that require larger amounts of data to be transferred and/or have a high level of interactiveness with the user. This includes mobile video communications (e.g., Skype, Face-Time, Google-Hangout), gaming applications that require sophisticated rendering and cloud media analytics that can be used to offer more personalized services. Challenges with these types of applications include potentially high latency incurred by the wired network and packet loss of the wireless network. Other limitations include the limited processing power and energy constrains which makes it difficult to support applications with high computing needs e.g., natural language processing.

One approach is to offload tasks from mobile devices to a proxy. A proxy is a powerful machine on the wired network that is responsible for executing specific tasks on behalf of mobile devices. The proxies can also be used to offload tasks and data from the cloud to the proxy. Examples of work that use proxies to offload compute-intensive tasks from mobile devices to a computational infrastructure is introduced in [1]. There are proxy-based approaches used to handle the mobility of mobile devices [2], [3]. An example of augmented reality applications that use proxies in the communication between mobile device and remote resource is presented in [4]. In [5] proxies are used to reduce network latency in video streaming application.

We introduced a proxy-based architecture for mobile computing [6], [7] that proposes to have a set of proxies in distinct locations. The proxies in our proposed system are designed as a dynamic library of generic services that can be used by a broad range of services and client applications. A proxy discovery mechanism is introduced and evaluated in [7]. The evaluation shows the effectiveness of the use of proxies and the proxy discovery mechanism. The brief description of the proposed proxy-based architecture and the proxy discovery mechanism, introduced in [6] and [7], is provided in the section II of this paper.

Multiple mobile devices may use a proxy. The resource demands on a proxy incurred by a mobile device may vary and thus at some point result in an overload. In our earlier work we did not consider that a proxy might become overloaded. To deal with overload, we introduce the handoff mechanism to mitigate overload. The handoff operation refers to the process of transferring the responsibility of servicing a client to another proxy. This paper introduces a handoff mechanism that is transparent to the client application. The purpose of this paper is not to compare the performance of our proxy-based system with other proxy-based proposals. This paper aims to show the effectiveness of our proposed handoff mechanism in providing favorable operation condition for mobile application, in ever-changing mobile wireless environment.

In the section II, a description of our mobile computing system is provided. The description includes the functionality of components of the system. The handoff mechanism is presented in section III. The description of the example client application and proxy services that are used by this application are presented in section IV. Experiments to evaluate the effectiveness of the handoff and obtained results are presented in the section V. In section VI the related work is presented.

MOBILWARE 2013, November 11-12, Bologna, Italy

DOI 10.4108/icst.mobilware.2013.254207

Finally, the conclusion and future work is provided in the section VII.

II. Proxy-based Mobile Computing System

There are multiple proxies that are used to offload computation from mobile devices and to provide services, referred to as *proxy services*, to facilitate communication between remote services, including cloud services, and client applications found on the mobile device. Section II-A describes proxies and *proxy services* in more detail.

The address and other information about proxies are maintained by *Proxy Finder Servers* (*PFSs*). A *PFS* is used to support proxy discovery and handoff. The process of finding a proxy is described in the section II-B.

Our proposed proxy-based mobile computing is made of multiple proxies that are used to offload computation from mobile devices and to provide services, referred to as *proxy services*, to facilitate communication between remote services and client applications found on the mobile device. Section II-A describes proxies and *proxy services* in more detail.

The application on the mobile device needs to know the address of a proxy offering its required *proxy services*. The address and other information about proxies are kept in servers called *Proxy Finder Servers*(*PFSs*). The client application should have the address of at least one *PFS* and it is the responsibility of the *PFS* to find the appropriate proxy for a client application. *PFSs* are also responsible for cooperating with the proxy to execute handoff operation, for handling issues related to the mobility of mobile devices and changing conditions of the network connections. The process of finding a proxy is described in section II-B.

A. Proxy and Proxy Services

Proxies provide *proxy services* that enable the client application on mobile devices to access cloud services and resources smoothly. To achieve this goal, *proxy services* are used to handle challenges existing in the mobile and wireless environment. For example, to deal with frequent disconnections a *proxy service* can keep track of how much of the data has been transmitted so far. *Proxy services* can be used to offload tasks from the client application or request that a task from the cloud service be placed on the proxy. The latter is to decrease communication delay between the client application and the cloud service. *Proxy services* should be independent from the remote services and client applications. This independence increases the reusability so that each *proxy service* can be used by various client applications to access various remote services.

B. Proxy Discovery

PFSs are used to find a proxy upon the request of a client application. It is possible to have multiple *PFSs* that keep the information about available proxies. By having multiple *PFSs* the load on the *PFSs* is distributed among all of them and the system will maintain the scalability needed in the case of an increase in the number of proxies and mobile applications that intend to use proxies. Having multiple *PFSs* also improves the reliability of the system; if a *PFS* fails, because of any reason, there are other *PFSs* that can provide services for the client applications.

To be accessible by client applications, when a proxy first comes on-line, it registers itself with one or more *PFSs*. Client applications need to query a *PFS* to discover proxies. There are multiple mechanisms that can be used to provide client applications with *PFS* addresses. For example, *PFS* addresses could come from an ISP provider or a telecom company.

The proxy registers at several *PFSs* by providing the information about itself. The information provided by the proxy at registration should include all the information that is needed by the *PFS* to find the best proxy for a client application. The information includes, for example, the list of *proxy services* available on the proxy, the access information of the available *proxy services*, i.e. their URIs, the information that can be used by the *PFS* to determine the trustability of the proxy, the usage cost model that the proxy uses to charge its clients, etc. The information about proxies saved at *PFSs* can be updated later by proxies or by the *PFS*. For example, if the list of available *proxy services* at a proxy changes, the proxy informs the *PFSs* at which it is registered.

The process of proxy discovery is depicted in figure 1. The client application starts the operation of proxy finding by sending a request to a *PFS* (step 1). It is possible for the client application to send the request to multiple *PFSs* to speed up the proxy finding operation, by accepting the first proxy found by one of the *PFSs*. The request sent to the *PFS* should include the information that a *PFS* needs to find a proxy. The information might include the IP address of the mobile device, which is needed to find the network latency between the mobile device and the proxy or the remote service, the list of *proxy services* that are needed for the functionality of the client application, the information of any cloud services that will be accessed by the client application, such as the download URI of the cloud service, the cost model of the proxy, trustability information of the proxy, etc.

In its request to the PFS, the client application provides a set of factors and a weight (weights sum to one) for each factor indicating the importance for the factor choosing a proxy. Factors to be used in proxy selection include network delay, trustworthiness, etc; The weights are specific to a client application.

Upon receiving the request for a client application, the *PFS* chooses the best proxies with the information included in the request (step 2). The algorithm used by the *PFS* to choose a list of best proxies is presented in the section II-B1.

The *PFS* sends the client application request to all the proxies in the list, in this case, proxy 1 and proxy 2 (step 3). The proxy that receives the request from the *PFS*, decides whether to accept or reject the request (step 4). The proxy makes the decision based on its available resources. If the proxy does not have enough computing resources to service the new client application, it might reject the request. In the example shown in the figure 1, proxy 1 has decided to reject

the request, while proxy 2 has accepted the request (step 5). An alternative approach is to have proxies send resource usage information to the *PFSs* that they are registered with. This allows a *PFS* to select a proxy without contacting the proxy. The problem with this approach is that there may be a large number of proxies registered with a *PFS* that would require the *PFS* to handle many periodic messages from proxies. This load could increase the response time of the *PFS*.

Upon receiving the first acceptance from a contacted proxy, the *PFS* informs the mobile application about the information of the accepting proxy, including the IP address of proxy, URIs of the *proxy services* on the proxy that mobile application intends to call, etc. (step 6). The acceptance message sent by the chosen proxy also contains a client ID, issued by the proxy for the mobile application. The issued client ID is unique throughout the system and is used by the proxy and *proxy services* to handle the state information related to this mobile client application.

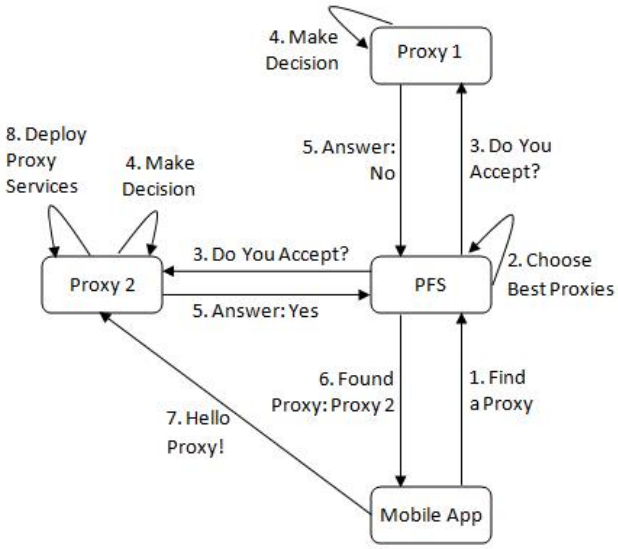

Fig. 1. Finding a Proxy

The client application, after receiving the information of chosen proxy, contacts the proxy by sending a *Hello Proxy* message that includes the client ID issued by the proxy (step 7). The proxy, upon receiving the *Hello Proxy* message from a client application, uses the client ID included in the message to retrieve the request of the client application. After retrieving the request, the proxy starts to prepare the required *proxy services* (step 8). The preparation includes the download of missing *proxy services* from an online repository of *proxy services* or copying them from a local repository and deploying all required *proxy services*. After successful deployment of all *proxy services*, the proxy sends an acknowledgment in response of the *Hello Proxy* message, to the client application. After receiving the acknowledgment from the proxy, the client application knows that the *proxy services* on the proxy are ready to use and the client application can start to access *proxy services* when needed.

1) How the PFS Chooses Best Proxies: A set of factors are used by the *PFS* to find the best proxies for a client application. The *PFS*, upon receiving a proxy discovery request, finds a list of best proxies based on these factors. The factors can be either related to the information that is included in the proxy registration request, for example the list of available *proxy services*, or can be taken from other resources. We categorize factors into two categories. The first category includes factors that their values are, more or less static, and change less often. The second category contains factors that their values dynamically change. Examples of factors in the first category, that are rather static, include the list of available *proxy services* on the proxy, the usage cost model of the proxy, the trustability information of the proxy, etc. Examples of factors in the second category, are the network latency and the processing load on the proxy. The network latency is dependent on the network condition and changes frequently. Likewise, the load on the proxy depends on the applications running on the proxy and might be changing over time.

These two categories of factors are treated differently by the *PFS* at the phase of choosing the best proxies. The values of the first category, can be gathered at the registration phase of the proxy at the *PFS*. The values of those factors can either be provided by the proxy, or be queried from relevant resources. For example, the usage cost model of the proxy can be included in the registration request sent by the proxy, while the trustability information about the proxy should be provided by a third party. Although, the values of these rather static factors might change during the time. As a result, the *PFS* should provide facilities that allow updating of these values. It implies that, for example, if the list of available *proxy services* on a proxy changes, the proxy is able to contact the *PFS* and update this information. It is also probable that the *PFS* contacts other resources periodically, to get the updated values for some factors. For example, the *PFS* can contact the third party periodically, to receive the updated information about the trustability of proxies.

Because values of factors in the second category change more often, the *PFS* needs to inquire about relevant resources and get the up-to-date values every time, before choosing the best proxies. For example, the *PFS* needs to contact proxies to find the spontaneous network latency or the processing load on the proxies.

As mentioned previously, in the section II-B, the *PFS* also considers the preferences of the client application. The client application indicates the importance of each factor by assigning a weight to each factor. The factors that are of the interest of the client application can be a subset of the factors that their values are kept by the *PFS*.

Two factors are taken into account in the prototype implementation of our system: i- available *proxy services* on the proxy, ii- the proximity of the proxy to the mobile device and to the cloud service, that is to be used by the mobile application. If the client application functionality does not involve any call to a cloud service, only the proximity between

the mobile device and the proxy will be measured.

The proximity between components over a network can be evaluated by the geographical distance between components, or can be measured using the Round Trip Time (RTT) between components or the hop count between them. In our prototype implementation we use the RTT. The RTT is defined as the time required for a signal to travel from the source node to the destination node and back. Smaller RTT between two components means less communication delay between those components. The RTT between two nodes changes over time because of the change in the load over the route between nodes. As a result, the RTT can be an indicator of the spontaneous condition of the route between components and gives a more accurate information about the route. Unlikely, a large or small values for the hop count or the geographical distance does not necessarily indicates a decline or improve in the condition of route between components. To find the RTT to the mobile device and the cloud service, the proxy uses ping calls.

Every proxy has a local repository of *proxy services*' archive files. A proxy is said to have a *proxy service*, if the archive file of that *proxy service* exists in the local repository of proxy. It is also possible for the proxy to download and deploy a *proxy service* from an online repository, provided that the proxy believes to have enough resources.

An appropriate proxy is a proxy that has a higher number of required *proxy services* and has a smaller RTT to the mobile device and the cloud service. The more *proxy services* a proxy has, the better since this saves time in downloading from elsewhere. The smaller RTT will result in lower communication delay between entities in the system. The tasks done by the *PFS* to find best proxies are shown in the algorithm 1.

```
   Data: ProxiesList
   Result: SortedProxiesList
1  for each Proxy in ProxiesList do
2  |  Proxy.FindRTT();
3  end
4  RTTScaledProxiesList = ScaleRTTs(ProxiesList);
5  PSsPercentageList = CalcPSsPercentage(ProxiesList);
6  SortedProxiesList = SortProxies(PSsPercentageList,
   RTTScaledProxiesList, ClientPreference);
```

Algorithm 1: PFS Chooses the Best Proxies

At the beginning, the *PFS* contacts all proxies that are registered with the *PFS* and asks them to find the RTT values (lines 1-3). After receiving the response from all proxies, the *PFS* calculates two values for each proxy: i- scaled RTT, from 0 to 100, based on RTT values reported by the proxies (line 4), ii- percentages of required *proxy services* that are available on the proxies (line 5). As mentioned earlier, the *PFS* already knows which *proxy services* are available on which proxy. This information is provided by the proxy during the registration phase at the *PFS*.

The details of scaling the RTT by the ScaleRTTs function is shown in the algorithm 2. The worst RTT, which means the highest, and the best RTT, which means the lowest, are found at the beginning (lines 1 and 2). Assuming that the best RTT is equivalent to 100 and the worst RTT is equivalent to 0, the scaled RTT for other proxies is calculated by the equation in the line 4.

```
   Data: ProxiesList
   Result: RTTScaledProxiesList
1  worstRTT = FindWorstRTT(ProxiesList);
2  bestRTT = FindBestRTT(ProxiesList);
3  for each Proxy in ProxiesList do
4  |  Proxy.ScaledRTT = 100 * (worstRTT - Proxy.RTT) /
   |  (worstRTT - bestRTT);
5  |  RTTScaledProxiesList.add(Proxy);
6  end
```

Algorithm 2: Scale RTT values

By having these two sorted lists, scaled RTTs and percentages of available *proxy services*, the *PFS* finds a number of best proxies based on the preference of the client application (algorithm 1, line 6). The preference of the client application has been included in the request sent by the client application to the *PFS*. The preference of the client application is represented by assigning a weight to each factor. The client application assigns a higher weight to the factor that has more importance for the application. For example, it might be more important for a client application to have a small network delay to the proxy but the client application is willing to wait a longer time at the start for the preparation of a proxy that does not have all the required *proxy services* available and needs to download and install them before starting to serve the client. In this case, the client might assign a weight of 80 to the RTT factor and a weight of 20 to the available *proxy services* on the proxy.

Details and results of experiments that show the effectiveness of our mobile computing system in achieving low delay in communication between mobile clients and services by choosing the most appropriate proxy can be found in [7].

III. Handoff

The handoff operation is used to change the associated proxy of a mobile client application. There are multiple events that can trigger the handoff operation. An event is characterized by a condition on one or more factors that characterize some aspect of application/system behavior. The factors can include those used by the *PFS* to find a proxy.

The thresholds, that define the acceptable values for the factors, can be defined either by the client application or by the proxy. For example, the client would declare a specific threshold for the RTT and require the *PFS* to always associate it with a proxy that has an RTT smaller than the threshold. On the other hand, the proxy might have as its objective that its processing load is always smaller than a specific threshold and if its load exceeds that threshold, the handoff process is triggered. If the threshold value for a factor is defined by the client application, it should be included in the clients request,

during the proxy discovery process. In this way the proxy will be aware of the favorable thresholds of factors for the client application.

The values of factors used in the conditions characterizing events, that are used to trigger the handoff process, are monitored. The monitoring can be done by the proxy itself, or by the client application. In the development of the prototype of our system, we decided to assign the monitoring responsibility to the proxy. In this way, the burden on the mobile device is reduced and the battery power consumption on the mobile device is reduced

The diagram showing the interaction between components of the system, during a handoff operation is depicted in figure 2. When a handoff triggering event is detected by a proxy, the proxy contacts the *PFS* to find a new proxy (step 1). The process of finding a new proxy during the handoff is the same as the process presented in the section II-B, with the exception that the request is sent by a proxy to the *PFS*, rather than by a client application. The proxy retrieves the initial request of client application that includes the list of required *proxy services*, information about the remote service and clients preferences, etc. The proxy sends the retrieved request to the *PFS*. After finding a new proxy, the *PFS* informs the requesting proxy on the address of the new proxy (step 2).

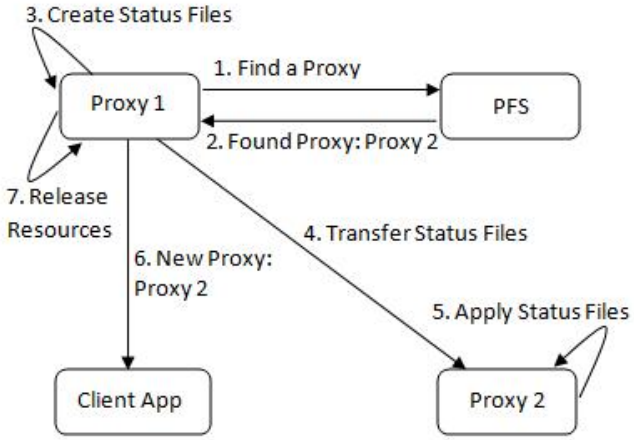

Fig. 2. Handoff Operation

After receiving the address of a new proxy, the current proxy is responsible for saving the state information of *proxy services* used by the client application and transferring the state information to the new proxy. Each *proxy service is* designed in a way to have a method, *saveState*, in its API, that saves the state information needed to create a new instance of the *proxy service* on the new proxy. The implementation of *saveState* is specific to every *proxy service*. The continuation of a *proxy service* on the new proxy should be provided in a way that the client application does not require to redo an already performed action.

To illustrate an example of state information, we consider a *proxy service* whose functionality includes transferring files from the mobile device to the proxy and handling disconnections. Whenever a disconnection happens, the client application queries for the number of already transmitted bytes from the proxy, and resumes the file transmission. The state in this case includes the number of already transmitted bytes from the mobile device to the proxy. The client application inquires the *proxy service* on the new proxy about the number of already transferred bytes. Since this number is transferred to the new proxy, as a part of the state information, the *proxy service* can reply accurately to the clients inquiry.

Besides the state of the *proxy services*, it might be necessary to transfer other data that is needed for the continuation of the *proxy service* from the old proxy to the new proxy. This other data may include databases or files, such as dictionaries or maps. For the example *proxy service*, introduced in the previous paragraph, the other data includes the already transferred part of the file on the old proxy.

After transferring the state information and other required data, the needed *proxy services* on the new proxy load their corresponding state files. The state loading might involve inserting data to databases, creating/copying required files, etc. The activity done during state loading phase, depends on the functionality of a *proxy service*. For example, for the above mentioned example *proxy service*, i.e. the *proxy service* that is responsible for handling disconnections, the state loading involves informing the newly created instance of the *proxy service*, about the location of the transmitted file and the number of already transmitted bytes.

The tasks of collecting the state information, transferring it to the new proxy and loading the state information on the new proxy, that are shown as steps 3-5 in the figure 2, are common activities that should be performed during a handoff process by any proxy for each of its *proxy services*, regardless of the functionality of the *proxy service*. In our prototype, an interface is defined that includes a method for each of these common tasks, named *saveState*, *transferState* and *loadState*. All *proxy services* should implement methods of this interface. The design and implementation of the interface methods is on the *proxy service* provider and it depends on the functionality of the *proxy service*. It is possible to have a stateless *proxy service*. In this case, the interface implementation is left empty.

After finishing the state loading on the new proxy, the new proxy is available to serve the client. The client application is to be transparently notified about the change of proxy. This requires that client applications have a component that we refer to as the *bridge*. The *bridge* component is always aware of the address of current proxy. In the case of an occurrence of a handoff operation, the old proxy notifies the *bridge* component. All other parts of the client application, before calling any methods of any *proxy services*, inquire the *bridge* to find the up-to-date proxy address.

Immediately after starting a handoff operation, the proxy stops serving the client application by terminating all instances of *proxy services* that are serving the clients request. A handoff seems like a disconnection to the client application and subsequently the client might try to re-call the *proxy services*.

Proxy services reject all calls from the client application when the handoff process is ongoing. Since the client application includes its clientID to every call to *proxy services*, it is possible to realize the calls from a client application migrating to the new proxy, and drop those calls.

After finishing the setup of the new proxy with the state of required *proxy services*, the old proxy contacts the *bridge* component of client application and provides it with the address of the new proxy (step 6). Hereafter, any call to *proxy services* from client application is directed to the new proxy by the *bridge* component.

In the last step of the handoff operation, the old proxy frees all resources that were assigned to the client, such as databases or files (step 7).

IV. Experimental Setup

To examine the effectiveness of our proposed handoff mechanism, several experiments are carried out. As previously described in the section III, various factors, depending on the preferences of the mobile application, can be used to ignite a handoff process. The factor used in our experiments to start a handoff process, is the RTT between the mobile device and the proxy.

The example application used for our experiments behaves like an augmented reality application. In augmented reality applications the information about the environment is gathered and processed and augmented with extra data and the result is delivered to the users. Examples of augmented reality applications for mobile devices include the translator [8], tourist assistance [9], [10], [11], and mobile learning [12] applications. For example in tourist assistance, the application can be used to capture image or video from a view, process the images or video, retrieve additional information about the view from a database and provide user with the extra information.

Augmented reality applications often require image or audio processing tasks that need computational power not typically found on mobile devices. One approach is to have a client application on the mobile device responsible for gathering the input data, e.g. image, video, or voice, and to offload the processing to a remote server rather than executing it locally on the mobile device, especially if the Wi-Fi connectivity is available [13]. After finishing the processing, the remote server sends the result in an appropriate format, such as image, video, voice or text to the client application which is responsible for showing the result to the user.

A. Example Client Application

An augmented reality-like client application is implemented for our experiments that uses our proxy-based system for experiencing a better performance. Our system tries to find a proxy with smallest RTT to the mobile device. The client application on the mobile device captures images of an object and transfers these images to the augmented reality service hosted on the proxy.

B. Proxy Services Used

The first *proxy service* that can be used by our example client application is the *CodeExec proxy service*. The *CodeExec proxy service* is responsible for downloading and starting a service on a proxy in response to client requests. The client request may include a URI where the service software may be downloaded from. Afterwards, the client is able to communicate with the service available on the proxy machine. The other proxy service used by the client application is the *FileStreamer proxy service*. The *FileStreamer* is used to transfer a stream of files from the client application to the proxy. The *FileSteamer proxy service* provides support for disconnection handling, i.e. the *FileStreamer proxy service* is aware of the possible disconnections and is able to reconnect and continue the communication between the client application and the service without the intervention of the client application or the service. The combination of these two *proxy services*, the *CodeExec* and the *FileStreamer*, enables programmers to create mobile client applications for augmented reality services that need the transmission of extensive amount of input data in the form of a stream of images, or of other formats, from the client and also requires extensive amount of processing on the input data. In the experiments done for this work, the client application uses the *FileStreamer proxy service* to transfer a stream of images from the mobile device to the proxy. Transferred images can be used by an augmented reality service, downloaded by the *CodeExec proxy service* and executed on the proxy. By having this example client application, we evaluate the effectiveness of the handoff mechanism in our system to provide lower network latency when the network latency between the mobile device and the proxy changes.

C. Testbed of Experiments

The mobile device used for our experiments is an Acer Iconia Tab A500 with the Android version 3.2. Two Linux machines are used as proxies. The proxy and *proxy services* functionalities are implemented as Web Services. The Apache Tomcat 7.0 is installed on two machines hosting proxies and *proxy services* as the web server. The Apache Axis2 1.6 is used as the web service engine.

V. Handoff Experiments and Results

To examine the effectiveness of the handoff process we simulate the changes in the RTT between the mobile device and the proxy by having the client application sleep before each write to the connection for the length of simulated RTT. The reason for simulating the RTT is that the mobile device and the machines hosting proxies are connected to the same LAN and the real RTT between them is negligible (usually less than 4 milliseconds).

In the real world implementation of the system, the proxy can monitor the RTT by periodically sending ping messages to the mobile device. In our experiments, the simulated RTT values are generated at the mobile application and are sent to the proxy. When the simulated RTT exceeds a predefined threshold, the proxy starts the process of handoff.

The factor that is used to measure the performance of our experiments is the number of images transferred from the mobile device to the proxy in 30 seconds. It is assumed that more images received by the cloud services will generate better results.

To evaluate the effectiveness of handoff, two scenarios were designed. In the first scenario, no handoff happens even if the RTT passes the threshold and client continues to transfer files to the same proxy. In the second scenario, the handoff occurs when the RTT exceeds the threshold. In the second scenario we assume the RTT between the new proxy and the mobile device is constant and is equal to the handoff threshold. It is the worst case. In the other cases the RTT between the mobile device and the new proxy would be less than the threshold value.

For the first scenario, the RTT increases linearly according to the equation 1. The variable t represents the time from the beginning of the experiment in seconds which will be between 0 and 30. The variable rtt is the RTT value in milliseconds which is calculated based on the value of t. Equation 1 is made based on the assumption that the RTT increases linearly from 10 millisecond to 100 millisecond during 30 seconds. For the second scenario, the RTT is calculated using the same equation until the rtt value exceeds the handoff threshold and a handoff process is triggered. From that point until the end of experiments (t = 30) the rtt will be equal to the threshold value. To find the influence of the threshold value we examined multiple values for the the threshold which include 14.5, 19, 28, 37, 46 and 64 millisecond. In addition to understand the effect of file size on the effectiveness of handoff, various file sizes were used in the experiments which are 100 KB, 250 KB, 500 KB, 750 KB, 1 MB and 1.5 MB.

$$rtt = 3t + 10 \tag{1}$$

The result of our experiments for the linear increase of RTT is depicted in the figure 3. The experiments for each file size and each threshold value are replicated three times. The chart shows the improvement of handoff cases over no handoff cases. The improvement is calculated as *100 * number of images transferred in with-handoff scenario / number of images transferred in no-handoff scenario*. The confidence intervals for the confidence level of 95% are shown in table I. As shown in the chart, the improvements for the biggest file (1.5 MB) are negative for all threshold values in contrast with the results for other smaller files. This result suggests that the handoff is not beneficial when the file is too large. The other observation is that the improvement values are negative for the largest handoff threshold (64 millisecond). It shows that it is better to start handoff on smaller values for the threshold. There is no significant difference between the result obtained for the file sizes of 100 KB, 250 KB, 500 KB and 750 KB for all thresholds but 64 millisecond. Almost all of them have a positive value between 0 and 20 percent. To find if the performance of the handoff mechanism for smaller file sizes, i.e. smaller than 750 KB, further experiments are needed. Additionally, some values achieved in experiments, such as the value for the file size of 500 KB and handoff threshold of 14.5 ms do not fit in the overall trend for smaller files. More experiments will be done as future work to understand the reason of this observation.

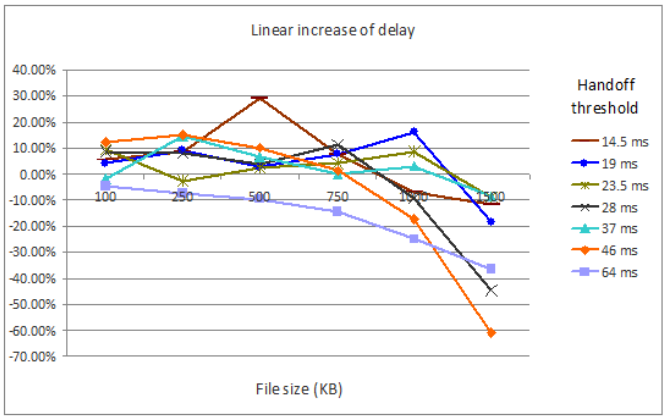

Fig. 3. Improvement of With-Handoff Scenario over No-Handoff Scenario in Case of Linear Increase of RTT

In our experiments we noticed that sometimes the real RTT in the wireless environment increases for a short period of time and then returns to its previous level. This spike increase may cause the start of a handoff process. To understand the effect of handoff, started because of such a spike increase in the RTT, we designed and examined another set of experiments including two scenarios. In this set of experiments, the RTT is equal to the constant value of 10 milliseconds but, to simulate a spike increase, it increases from 10 to 60 milliseconds for a period of 2 seconds in the middle of the experiments, i.e. from 14th second to the 16th second, then returns to 10 millisecond. In the first scenario, i.e. no-handoff scenario, no handoff occurs during the experiment time, while in the second scenario, i.e. with-handoff, when the simulated RTT exceeds the threshold, the handoff process is started. Three threshold values are examined for the second scenario that are 19, 28, and 46 millisecond. Same as the previous set of experiments, the experiments for each file size and each threshold value are replicated three times and the improvement is calculated as *100 * number of images transferred in with-handoff scenario / number of images transferred in no-handoff scenario*. The confidence intervals for the confidence level of 95% are shown in table II.

The result of experiments for these scenarios, i.e with spike increase in the RTT, is shown in the figure 4. As can be seen in the chart, almost all of improvement percentages for files larger than 250 KB are negative. This means that it was not effective to do a handoff when a spike increase in delay occurs. This result suggests that it is necessary to have a mechanism in the system to realize the transient increases in the RTT and to avoid starting the handoff process in those cases.

TABLE I
LINEAR INCREASE OF RTT - CONFIDENCE INTERVALS

File Size (MB)	Handoff Thresholds (MSec)						
	14.5	19	23.5	28	37	46	64
100	5.6 ± 0.04	4.14 ± 0.03	9.5 ± 0.06	8.64 ± 0.09	-1.92 ± 0.05	12.29 ± 0.07	-4.66 ± 0.16
250	8.62 ± 0.03	8.96 ± 0.08	-2.48 ± 0.12	7.95 ± 0.05	14.85 ± 0.10	14.96 ± 0.12	-7.43 ± 0.09
500	29.28 ± 0.24	2.88 ± 0.03	2.40 ± 0.04	3.78 ± 0.05	6.49 ± 0.03	10.07 ± 0.18	-9.65 ± 0.12
750	7.59 ± 0.04	7.60 ± 0.08	4.02 ± 0.06	11.10 ± 0.19	-0.08 ± 0.09	1.35 ± 0.06	-14.58 ± 0.17
1000	-6.60 ± 0.11	16.02 ± 0.17	8.58 ± 0.04	-9.06 ± 0.04	3.19 ± 0.20	-17.41 ± 0.07	-24.96 ± 0.12
1500	-11.61 ± 0.26	-18.21 ± 0.36	-8.92 ± 0.30	-44.73 ± 0.19	-8.55 ± 0.61	-60.72 ± 0.15	-36.52 ± 0.11

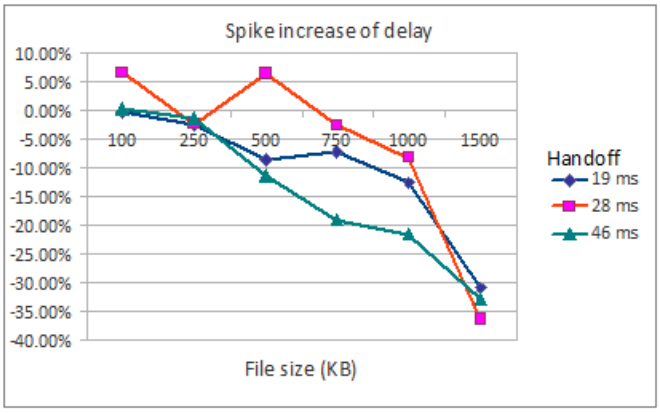

Fig. 4. Improvement of With-Handoff Scenario over No-Handoff Scenario in Case of Spike Increase of RTT

TABLE II
SPIKE INCREASE OF RTT - CONFIDENCE INTERVALS

File Size (MB)	Handoff Thresholds (MSec)		
	19	28	46
100	-0.23 ± 0.01	6.61 ± 0.09	0.34 ± 0.09
250	-2.59 ± 0.03	-2.26 ± 0.10	-1.23 ± 0.04
500	-8.52 ± 0.01	6.42 ± 0.16	-11.30 ± 0.09
750	-7.18 ± 0.04	-2.45 ± 0.14	-19.03 ± 0.15
1000	-12.58 ± 0.02	-8.16 ± 0.15	-21.56 ± 0.15
1500	-30.81 ± 0.10	-36.21 ± 0.03	-32.64 ± 0.05

VI. RELATED WORK

There is considerable literature related to offloading resource intensive tasks from the mobile device to a powerful proxy [1], [3], [10], [14], [15], [16], [17], [18], [19], [20]. Offloading tasks to a remote server adds network latency to the interaction between the client application and the service that is often not acceptable for delay sensitive applications. To mitigate the negative impact of this added delay, several papers have proposed the use of a proxy machine which is closer to the location of the mobile device [21], [15], [5], [1]. The advantage of our proposed infrastructure is that we consider the preferences of the client application to find the best proxy to offload tasks. In the other work, there is no explanation about choosing the host that tasks are offloaded to, from mobile device. In the other work, it is assumed that the proxy, is always the same and its address is hard-coded into the client application.

Additionally, some research has proposed to move the service and proxy along with the mobile movement to keep the communication delay low [2]. In this work the location of the remote service changes adaptively, based on the communication delay between client application and the remote service. Our work considers changes in the communication delay, as well as other factors, such as load on the proxy, usage cost model of the proxy, trust of the proxy, changes in the preferences of the client application, etc.

The handoff ability has been introduced in other communication and networking areas, such as cellular networks and Mobile Ad-hoc Networks (MANETs). The concept of handoff in our work is similar to the concept of handoff in cellular networks. The quality of connection to the access point to the network is monitored. If the quality deteriorates, a new access point with a better connection quality is chosen and the mobile device is handed off to the new access point. However, the cellular network research targets the telecommunication technologies, such as GSM, GPRS, etc. [22], and cannot be used for the accessing the application services.

The use of handoff is also found in MANETs and vehicular networks and several handoff techniques are proposed and implemented [23]. The handoff techniques proposed for MANNETs and vehicular networks mostly are implemented for lower networking layers, e.g. data link, network and transport layers. As a result, to deploy those techniques the alteration of underlying technologies are needed. Our approach is implemented at the application layer with no modification required for the current Internet architecture.

VII. CONCLUSION

The changing in the environment in mobile computing systems leads to the changes in the quality of the connection and the service perceived by the client application. In this paper we presented our proxy-based mobile infrastructure that is able to collect the preferences of the client application, monitor the condition of the environment and apply changes to the service provided for the client application, based on its preferences. The mechanism that is carried on, in response of changes in the conditions, is the handoff of client application between proxies.

The result of our experiments shows that the handoff mechanism provided in our proxy-based system is successful in keeping the favorable condition for the client application. In this case a favorable condition for the client application

is keeping the communication delay below a threshold. Other factors that impact the functionality of the client application and the performance of whole system, such as the trustability of proxy and the load on the proxy, can be used in the process of finding proxy and triggering a handoff operation.

The results shows that since client applications are typically time-sensitive or interactive, and as a result highly sensitive to the condition of the environment that they are operating in, it is necessary to dynamically adapt the services provided by the mobile computing infrastructure based on the changes in the condition to provide a better experience at the client application when using the mobile computing facilities.

REFERENCES

[1] B. Chun, S. Ihm, P. Maniatis, and et al., "Clonecloud: elastic execution between mobile device and cloud," in *Proceedings of the sixth conference on Computer systems*, ser. EuroSys '11, 2011, pp. 301–314.

[2] F. Samimi and P. McKinley, "Dynamis: Dynamic overlay service composition for distributed stream processing." in *SEKE*. Citeseer, 2008, pp. 881–886.

[3] P. Bellavista, A. Corradi, and C. Giannelli, "Mobility-aware management of internet connectivity in always best served wireless scenarios," *Mob. Netw. Appl.*, vol. 14, no. 1, pp. 18–34, feb 2009.

[4] P. Selonen, P. Belimpasakis, Y. You, and et al., "Mixed reality web service platform," *Multimedia Systems*, vol. 18, no. 3, pp. 215–230, 2012.

[5] C. Taylor and J. Pasquale, "Improving video performance in vnc under latency conditions," in *Proceedings of the International Symposium on Collaborative Technologies and Systems (CTS)*, Chicago, May 2010.

[6] A. Khalaj, H. Lutfiyya, and M. Perry, "The proxy-based mobile grid," in *The Third International ICST Conference on MOBILe Wireless MiddleWARE, Operating Systems, and Applications*, June-July 2010.

[7] ——, "A proxy-based mobile computing infrastructure," in *Mobile, Ubiquitous, and Intelligent Computing (MUSIC), 2012 Third FTRA International Conference on*, June 2012, pp. 21 –28.

[8] V. Fragoso, S. Gauglitz, S. Zamora, and et al., "Translatar: A mobile augmented reality translator," in *Applications of Computer Vision (WACV), 2011 IEEE Workshop on*, 2011, pp. 497–502.

[9] P. Belimpasakis, P. Selonen, and Y. You, "Bringing user-generated content from internet services to mobile augmented reality clients," in *Virtual Reality Workshop (CMCVR), 2010 Cloud-Mobile Convergence for*, 2010, pp. 14 –17.

[10] S. Gammeter, A. Gassmann, L. Bossard, and et al., "Server-side object recognition and client-side object tracking for mobile augmented reality," in *IEEE International Workshop on Mobile Vision (CVPR 2010)*, June 2010.

[11] C. Shin, H. Kim, C. Kang, and et al., "Unified context-aware augmented reality application framework for user-driven tour guides," in *Ubiquitous Virtual Reality (ISUVR), 2010 International Symposium on*, july 2010, pp. 52 –55.

[12] M. Specht, S. Ternier, and W. Greller, "Mobile augmented reality for learning: A case study," *Journal of the Research Center for Educational Technology*, vol. 7, no. 1, 2011.

[13] B. Girod, V. Chandrasekhar, D. M. Chen, and et al., "Mobile visual search," *IEEE Signal Processing Magazine, Special Issue on Mobile Media Search*, 2010.

[14] E. Cuervo, A. Balasubramanian, D. Cho, and et al., "Maui: making smartphones last longer with code offload," in *MobiSys'10*, 2010, pp. 49–62.

[15] C. Taylor and J. Pasquale, "Towards a proximal resource-based architecture to support augmented reality applications," in *Workshop on Cloud-Mobile Convergence for Virtual Reality*, Waltham, MA, USA, March 2010.

[16] J. Lee, R. Doerner, J. Luderschmidt, and et al., "Collaboration between tabletop and mobile device." in *ISUVR*. IEEE, 2011, pp. 29–32.

[17] P. Angin, B. Bhargava, and S. Helal, "A mobile-cloud collaborative traffic lights detector for blind navigation," in *Proceedings of the 2010 Eleventh International Conference on Mobile Data Management*, ser. MDM '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 396–401.

[18] M. Kjaergaard, S. Bhattacharya, H. Blunck, and et al., "Energy-efficient trajectory tracking for mobile devices," in *Proceedings of the 9th international conference on Mobile systems, applications, and services*, ser. MobiSys '11. New York, NY, USA: ACM, 2011, pp. 307–320.

[19] K. Kumar, Y. Nimmagadda, and Y. Lu, "Energy conservation for image retrieval on mobile systems," *ACM Trans. Embed. Comput. Syst.*, vol. 11, no. 3, pp. 66:1–66:22, sep 2012.

[20] I. Giurgiu, O. Riva, and G. Alonso, "Dynamic software deployment from clouds to mobile devices," in *Middleware 2012*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, vol. 7662, pp. 394–414.

[21] M. Satyanarayanan, P. Bahl, R. Caceres, and et al., "The case for vm-based cloudlets in mobile computing," *IEEE Pervasive Computing*, vol. 8, pp. 14–23, 2009.

[22] S. Fernandes and A. Karmouch, "Vertical mobility management architectures in wireless networks: A comprehensive survey and future directions," *Communications Surveys Tutorials, IEEE*, vol. 14, no. 1, pp. 45–63, 2012.

[23] K. Zhu, D. Niyato, P. Wang, and et al., "Mobility and handoff management in vehicular networks: a survey," *Wireless Communications and Mobile Computing*, vol. 11, no. 4, pp. 459–476, 2011.