Developing Collaborative Applications with Mobile Cloud - A Case Study of Speech Recognition

Yu-Shuo Chang and Shih-Hao Hung
Department of Computer Science and Information Engineering
National Taiwan University, Taipei, 106, Taiwan
asouchang@gmail.com, hungsh@csie.ntu.edu.tw

Abstract

While the combination of cloud computing and mobile computing, termed mobile cloud computing, started to show its effects recently with many seemingly innovative smartphone applications and cloud services surfacing to the market today, we believe that the real potentials of mobile cloud computing is far from been fully explored due to several practical issues. The quality of the mobile networks is not adequate for delivering satisfactory user experiences via close collaboration over mobile networks. A dynamic workload partitioning scheme would help solve this problem, but the distribution of computation and data storage geographically can lead to serious security and privacy concerns, which makes user to take the risk of exposing the data to eavesdroppers in the middle of the network.

Since we have yet to see collaborative mobile cloud applications which could dynamically migrate the workload to efficiently take advantage of the resources in the cloud, in this paper, we present a paradigm to guide the design of the following: the system architecture, the principle for partitioning applications, the method for offloading computation, and the control policy for data access. We argue that the proposed paradigm provides a unified solution to the performance and privacy issues, with a case study, a cloud-assisted speech recognition application, to illustrate our experimental results.

1 Introduction

Recently, cloud computing has changed software infrastructures and business models of Internet services with technologies to provide and manage abundant of resources of computation and data storage over the network at relatively low costs. On the other hand, the popularity of smart devices and mobile networks has substantially changed the way people access computers and network services. While the combination of cloud computing and mobile computing, termed *mobile cloud computing*, started to show its effects with many seemingly innovative smartphone applications and cloud services surfacing to the market today, we believe that the real potentials of mobile cloud computing is far from been fully explored.

First of all, the quality of the *mobile networks* is not adequate for delivering satisfactory user experiences via *close collaboration* between mobile devices and cloud services. It is difficult to guarantee the smoothness of a mobile application as the response time depends on the condition (i.e. latency and bandwidth) of the mobile network. For a pervasive application over a wireless network, performance and functional issues arise when the client and the server communicate over a slow or unreliable network connection. Besides improving the quality of mobile network, the adoption of a dynamic workload partitioning scheme would help solve this problem. Unfortunately, most mobile cloud applications today use fixed partitioning schemes, so the users have mixed experiences.

Secondly, distribution of computation and data storage geographically can lead to serious *security and privacy concerns*. Besides willing to take the risk of exposing the data to eavesdroppers in the middle of the network, the users have to trust the service providers with their personal data. While it has been argued whether adoption of cloud computing offered better or worse security and privacy over traditional ways of keeping the services local, it is obviously an obstacle for many users and businesses to embrace

Journal of Internet Services and Information Security, volume: 1, number: 1, pp. 18-36

cloud-based Internet services. On the other side, it is a heavy burden for a service provider to ensure that user data are absolutely secure, when the user data are frequently replicated and populated all over the cloud infrastructure. For example, *Facebook* has long been criticized for privacy risks.

So far, we have yet to see *collaborative mobile cloud applications* which could dynamically migrate the workload to efficiently take advantage of the resources in the cloud. On the system side, the software infrastructure of today's mobile cloud computing environment is not sophisticated enough to mitigate the problems mentioned above. Application migration schemes, workload offloading mechanisms, efficient distributed file access/sharing, and trust models are among the active research topics for improving the software infrastructure. The related technologies will be discussed in Section 2.

On the application side, it is difficult for developers to come out of innovative collaborative applications due to the lack of experiences, programming tools, paradigms, and examples. Traditional *client-server* models have been successfully used for mobile applications to leverage the services in the cloud, but we hope to do better in the age of mobile cloud computing. First, application partitioning and deployment of services should dynamically adapt to the resources available in the set of mobile devices and the cloud servers (a.k.a. mobile cloud). The users should be able to deploy their own personal cloud services on-demand on the computer nodes that they can trust with personal data. In addition to sharing data conveniently and securely, applications may be migrated or distributed from one computer to another to encourage close collaboration.

In Section 3, we discuss the *kernel-offload paradigm* for collaborative mobile cloud applications. We attempt to come out of a paradigm to guide us in the design of the following: the system architecture, the principle for partitioning applications, the method for offloading computation, and the control policy for data access.

In Section 4, we illustrate the paradigm with a case study. We first show the design of our *cloud-assisted speech recognition (CSR)* service and then we explain the implementation of the CSR based on the collaborative application paradigm. Performance evaluation is critical to the application of a paradigm, so we characterize the performance of the CSR by analyzing the latency, the power cost, and the privacy issues associated with the service. Finally, in Section 5, we conclude this paper and discuss the potential future work.

2 Related Work

It was shown in earlier work that remote execution saved a large amount of power for mobile computers [10, 21, 17]. While there were different approaches proposed, program partitioning for remote execution has been a challenge for researchers. Spectra [12] proposed to monitor the current resource availability and dynamically determine the best remote execution plan for an application. Cyber foraging [7] used surrogates to improve the performance of interactive applications and distributed file systems on mobile clients. MAUI [11] proposed to reduce the programming efforts by automating program partitioning with the combination of code portability, serialization, reflection, and type safety.

Without application partitioning, methods have been developed to migrate the entire application execution to a remote server. Process migration and virtual machine migration have been two common approaches to migrate execution across the network. The ISR system [23] emulated the capabilities of suspend/resume functions in a computer system and migrate the system by storing the snapshot image of a virtual machine in a distributed storage system. Zap [20] introduced a pod (PrOcess Domain) abstraction, which provided a collection of processes with a host-independent virtualized view of the operating system, so as to support a general-purpose process migration functionality, but it did not allow live migration. Live VM migration [16] achieved rapid movement of workloads within clusters and data centers with minimal service downtime by continuously pre-copying page changes in a virtual machine

to another system, but on highly demand of network bandwidth.

Cloudlet[22] is a concept to deploy an in-box cloud in house, which is self-managed and does not need any professional attention. A cloudlet is deployed for few users to share the computing resource via the high speed LAN network, rather than using the distant public cloud on the Internet in order to shorten the latency for real time application. The concept is also adopted in our proposed architecture to make the user can setup their own cloud at home, which answered some privacy issues in some cases, we will discuss later in Section 3.

However, most of the previous works did not specially address the need for smartphone applications to offload workloads in a pervasive environment with limited network bandwidth, and most of them only focus on the pure computation offloading and migration. Nor did they address to offload the applications if there is I/O operations in them, and concern the control/privacy/secure issues surfaced in today's cloud services as extensively we did in our work. Hence, in this paper, we proposed an application architecture for developments to support the offloading process between the smartphone and the cloud.

3 A Paradigm for Collaborative Applications

In this section, we discuss the paradigm for mobile cloud applications. Traditional client-server or dumb-terminal approaches have been used for years, but application developers have not been able to exploit the potential of mobile cloud computing and develop collaborative applications yet. In Section 3.1, we examine the issues from architectural point of view and application developer's point of view. We proposed a paradigm to enable fine-grained and dynamic collaboration between computers. Section 3.2 further mentions the performance issues and discusses system infrastructure and application development efforts needed to enhance the performance of collaborative applications. Section 3.3 addresses the privacy and security issues. Finally, we compare the proposed paradigm with other schemes for the mobile cloud in Section 3.4.

3.1 Architecture and Paradigm

Today, an application on a smart mobile device, such as Android, is usually composed by these components: the *user interface (UI)*, the *work flow*, and the *kernel functions*, where each component is described below:

- The UI is in charge of interacting with the user by presenting *output events* to the user and receive *input events* from the user. UI is distinguished from the other input/output (I/O) functions and is especially important to the development of mobile cloud applications because UI must be performed on a mobile device. Modern graphical UI displays graphical events (e.g window), formats information, and presents multimedia (audio/video) contents to the user. UI may receive a variety of input events from the user and the mobile device, such as text, mouse click, mouse movement, voice command, audio input, video stream, location information from the global positioning system (GPS), etc.

- The work flow is a depiction of a sequence of operations for handling the request received from the UI. A work flow may be seen as an abstraction of real work, or a high-level processing logic (control flow and data flow) in the application design. Designing the work flow for a mobile cloud application is different from designing the algorithms in kernel functions, which is why developers usually separate these two components. Global variables are often used for the work flow to keep track of the *global application state*.

- The kernel functions perform the work specified by the work flow in each step. Each kernel function may apply one or multiple algorithms to carry out the computation with one or multiple processors and access the data available on the mobile device or on the cloud. The local variables declared in a kernel function as well as the temporary buffers used by the kernel function compose the *local state* of the kernel function.

In today's mobile applications, the main program running on the mobile device controls the UI and the work flow. The application may perform one kernel function at a time or perform multiple kernel function simultaneously with multiple processes and threads. Multithreaded applications are common for the Android systems. On the other hand, cloud applications often refer to those applications which utilize cloud services to carry out part of the application workload. The partitioning of the application workload is usually *fixed* in a typical cloud application, where the programmer has to partition the applications and work with a service provider to deploy the cloud service in advance. Often, a cloud-based applications simply has the UI running on the mobile device and have the rest of the application handled by a cloud service, which is essentially a *client-server* model.

Moreover, many of today's cloud applications merely use existing services available in the cloud. Thanks to the infrastructure technologies of cloud computing, there are more and more services available to the users for low or no charges. Application developers also benefit from such services as they simply need to follow the API's created by the provider to leverage the services. In practice, some service providers require or encourage the users to register their own accounts before using the services. This has raised privacy issues. Even when registration is not required, the service provider may still record the transactions with information collected by client program. For example, Google Maps APIs [2] allow the programmers to embed the functionality of the Google Maps service into their applications. The use of Google Maps is free, but the location information of the user along with other personal information are exposed to Google unless the user explicitly takes actions to forbid the client program from collecting and sending the information out. Even if we trust Google for now, there are numerous examples for mobile applications like the above, so what can we do besides not to use those services?

We believe that some of the public services are important and should be operated by *trustworthy providers* to reduce the security and privacy risks, but some users would still be hesitate to trust the providers. To further prevent the leak of personal information, *proxy* or *firewall* can be invoked by the user to filter the outbound information or block the entire service. However, it is tedious for the user to configure the filtering mechanism, especially when mobile applications and cloud services are frequently updated.

On the other hand, when a developer decides to offload a kernel function to the cloud, the developer first need to define and create an interface for the service to offload the function, i.e. *offload service*. Then, the developer needs to implement the offload service and packages it as a server program. The developer can work with a service provider to deploy the server program. Cloud computing technologies, such as the *Google App Engine (GAE)* [1], speed up the deployment and management of such services for individual developers, so that we see new cloud-based services coming out everyday. In the case of GAE, in addition to Google, the developers also participate in the operation of the service and have access to the user's personal information. Again, this has raised privacy issues.

Of course, the developer can also distribute the server program as a part of the application to the user, so that the user may install the service on their own. However, how many users have the resources and are capable of installing the service on their own? First, the user needs to find a *trusted computer* on the network to host the service. Finding a computer would be the easy part, but establishing a trusted relationship between two computers are not easy, as security mechanisms are needed to protect the data storage in the remote computer and communications between the two computers. Today, trusted computing technologies [19] are available, but they are not widely used by mobile cloud applications

because most application developers and users are not familiar with such technologies.

Can we elaborate the client-server model to allow better (fine-grain and dynamic) application partitioning for the developers and address the security/privacy issues for the users? For that, we modified a client-server architecture, and its high-level concepts are illustrated as shown in Figure 1:

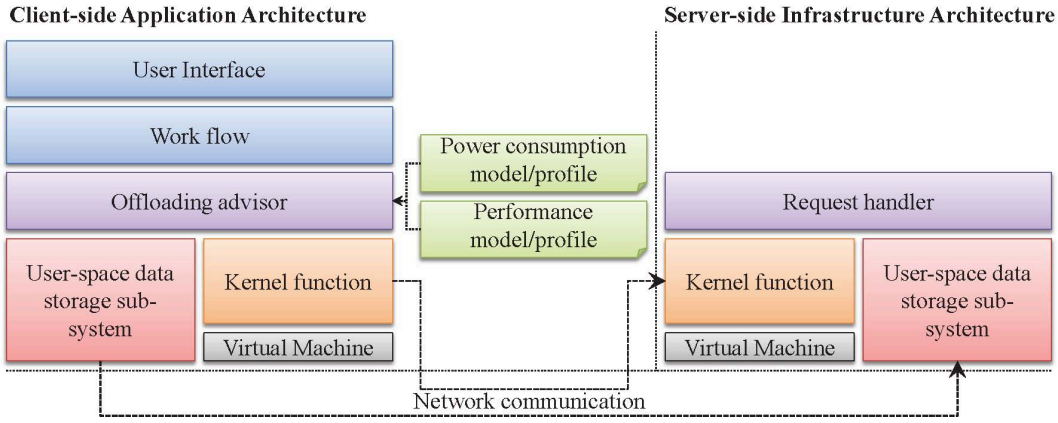

Figure 1: A modified client-server architecture for collaborative applications

- First, the architecture allows the application developer to *mark the kernel functions* in the applications. The kernel functions are executed on a virtual machine in this case to enable cross-platform execution. The recent development of *OpenCL (Open Computing Language)* [24] for multicore systems is in-line with this direction. The OpenCL language requires the programmer to explicitly declare the kernel functions in the application, so that a multicore system may distribute the kernel functions onto separate cores and execute them in parallel. Although we are targeting at collaborative applications across systems, the concepts are similar. In our case, the kernels are considered to built as a Java bytecode or C/C++ bitcode module for portability so that the kernels can be compatible and runnable without any rebuild effort on multiple platforms if there are Java Virtual Machine or Low Level Virtual Machine [18] available on the targeting platforms. The setting up of execution environment and the deployment of the kernels are one-time efforts.

- The *offloading advisor* on the client side monitors the execution of kernel functions and the file accesses in the application. The offloading advisor also serves as the contact window for collaboration between the client and the other systems. There is a control program, the *request handler*, running on a system which is willing to share its resources as a service. There is a notion of a *trusted group*, where the systems in the group trust each other and would share their resources. The request handler monitors the resources on the server, reports the status of the server to the clients, and handles the offloading request from any client.

- The offloading advisor is equipped with a *performance model* and a *power model* to assist the application in deciding if a kernel function may benefit via remote execution on a server in the cloud. To derive a offload plan, the offloading advisor may need to profile the application and store the performance/power consumption profiles from the previous runs. To make an intelligent decision, the offloading advisor also needs to have the following information updated periodically: the model and status of the servers, the locations of files, and the condition of the network. When the offloading advisor decides to offload a kernel function, then it migrates the kernel function to the server with the help from the request handler on the server.

- The accesses to file storage are handled by a *user-space distributed filesystem*. The use of a distributed filesystem allows the application on the client and the kernel functions on the server to access to the same files coherently in the same name space conveniently without modifications. The offloading advisor may query the distributed filesystem to check the location of a file when it tries to make a offloading decision. The performance of the distributed filesystem may benefit from techniques such as caching recently accesses files, prefetching files in advance, or allowing the offloading advisor or the request handler to control the coherence protocol for specific files.

We believe that the modified client-server architecture paves the road and serves as a paradigm for the future collaborative mobile cloud applications. There are many research issues in the architecture, but from application developer's point of view, this is a viable paradigm.

3.2 Performance Issues and Solutions

For a collaborative application which follows the aforementioned paradigm to deliver good performance in light of the obstacles mentioned in Section 1, several practical issues need to be resolved. We will focus on the performance issues in this subsection and leave the privacy and security issues to the next subsection.

Instead of asking the system infrastructure to handle everything automatically, we believe that application developers can contribute significantly in the optimization of application performance by providing a few *hints* to guide the system in offloading kernel functions and synchronizing files.

It is critical to have an efficient mechanism for migrating the kernel functions over the network. Many previous works attempted to migrate the process/thread [23, 20] or the virtual machine [16], but the overhead of migration can be prohibitively high, especially for mobile networks, since these works target at migrating the execution at any time.

First, we believe that the ability to migrate an application execution at any time is desirable, but is unnecessary for developing collaborative applications. Second, it may be overkill and waste of network bandwidth to migrate the entire application to carry out part of the application remotely.

To reduce the network traffic for remote execution, our paradigm requires that the application developer to explicitly mark the kernel functions, so that the system may focus on the offloading of a kernel function at the beginning of the function. The offloading advisor only needs to transfer the input of the kernel function and the global application state associated with the kernel function to the server, which can be far less than the migrating the entire application at the middle of a memory-hungry kernel function.

As far as the file access is concerned, in order to efficiently migrate the data which are needed immediately, the offloading advisor could give a list of files to the distributed filesystem, so that the server can try to prefetch the files in advance. Take an Android application for example. We have designed a set of *heuristics* for the offloading advisor to prioritize and control the synchronization of files in the course of application migration. The application developer may override the arrangement if necessary.

While an Android application is migrated from one machine to another machine, it is necessary to clone its application states and data in the storage system. A brute-force way for synchronizing data between two environments is to maintain identical data for all files in their file systems. For that, the data must be updated as soon as one environment makes changes to any file, which incur network traffics, and the applications need to wait for the completion of any synchronization operation if a strict synchronization protocol [8] is used, which would cause a long delay with a mobile network. To mitigate this problem, we need to look further into the policies for synchronization.

First, we divide the data storage into three categories: system image, system-wide data, and application data. When a virtual environment is initialized, its filesystem is loaded with the system image and application packages that are on a standard operating environment image. Synchronizing system image and application packages happen infrequently and should not be an issue for the virtual environment in the cloud as it may download the image from a high-bandwidth network. System-wide data refers to those files that record system-wide information and/or would affect the operations of the system and applications. Libraries are examples of system data. Modifying a library may affect multiple applications running on the system. When a physical environment makes a change to its system data, it often stops the applications that might be affected and sometimes requires the system to reboot. Similarly, its virtual environment counterpart should follow the same procedure.

Application data refers to the files owned by applications, and the synchronization of application data can be done on per-application basis during the course of application migration. Thus, we apply lazy and on-demand policies [25] to synchronize the data upon the request of an application without keeping application data updated all the time with a coherence protocol. Running a collaborative application that executes simultaneously on multiple devices requires the application developer to design the data communication or synchronization scheme specifically for the application.

3.3 Privacy and Security

For those who would like to protect their personal data, offloading kernel functions is significantly better than executing the entire application remotely, or sending data to a service which is not in control of the users. In our paradigm, security mechanisms can be further developed to reduce the risk of stolen data by distributing the kernel functions over multiple servers and destroying the temporary data used during the remote execution. Besides, unlike traditional service, the offloading service can be shared by multiple applications, so the user no longer needs to register an account for each service.

Still, offloading workload via the network increases security risk as the data propagated over the Internet and stored in the server could be eavesdropped by attackers in the middle and/or on the server. It is important to protect the sensitive user data with an *end-to-end secure communication channel*, such as the virtual private network (VPN), and encrypt the remote files using an encrypted filesystem. Thus, sensitive communications between offloading advisors and request handlers are through the secure channel.

In addition, choosing a trustworthy IaaS provider is critical to deploy a virtual environment. For stronger level of trust, Trusted Platform Module (TPM) could be integrated to enhance the security on the server system, offering hardware mechanisms to store the encryption keys and perform cryptographic operations on sensitive data. The TPM could also be used to store the master encryption keys and perform the encryption procedure to keep the encryption keys away from the eavesdroppers.

Finally, a *trust model* is needed to encourage the exchange of information and computing resources among a large group of people. Many trust models have been developed for peer-to-peer networks and social network communities, e.g. reputation-based trust [13, 26], are worth investigation. It also opens the possibility to connect these activities to derive joint trust models based on resource exchange activities in peer-to-peer networks, relationships in social networks, and collaboration in mobile cloud applications.

3.4 Comparison

In this section, we compare our modified client-server architecture with the following existing offloading approaches:

- The traditional approach refers to the classic design flow for creating an well partitioned applications with customized protocol and servers for offloading the computation. This approach is considered to have the best performance and the lowest offloading overhead because the developer can optimize the protocol with the domain knowledge and the use case, but it takes the most efforts for developer to design the application.

- The *terminal* approach has the mobile device performing the UI and the rest of the application running on the server. As shown in Figure 2, one could easily establish such collaborative scheme using the Virtual Network Computing (VNC) [4] to display the application's UI screen and receive user's input on the mobile device. Unfortunately, the latencies of wide-area mobile networks impact the user experience significantly. It is possible to utilize a higher-level event-based display protocol to improve the performance. While this approach saves the cost of the mobile device, the user needs to find a trustworthy server to perform the application, and the performance heavily depends on the quality of the network.

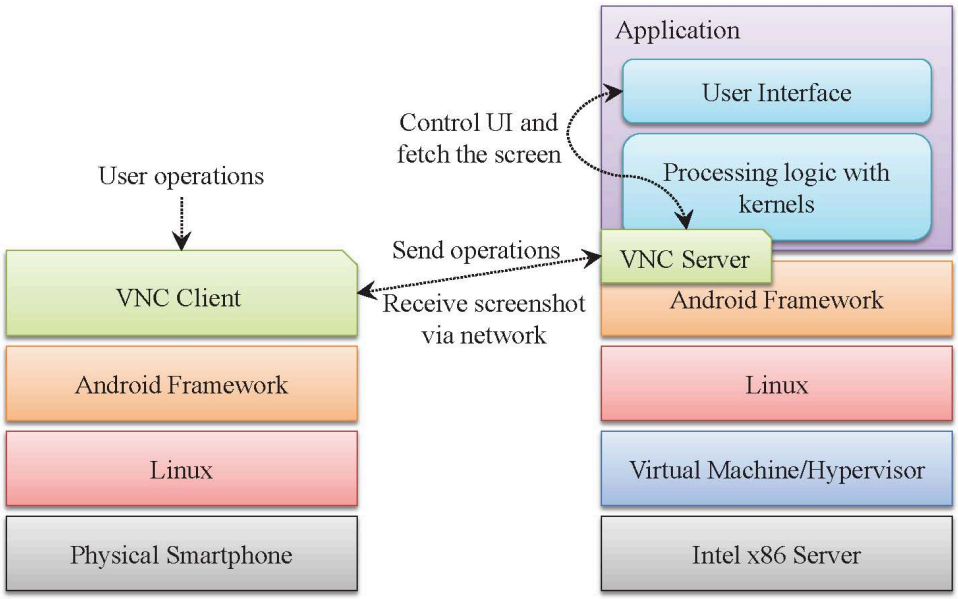

Figure 2: The terminal approach

- The *suspend/resume* migration approach [15] aims to dynamically migrate the whole process with its state from the smartphone phone to the cloud. At the server side, a virtualized environment, a.k.a. *virtual phone*, is used to support remote execution of Android applications on x86-based server, as shown in 3. The user may deploy a virtual phone and then migrate applications over to the virtual phone. For migrating an application, the framework first suspends the current application on the smartphone, which forces the application to save its state, and then transfers the state to resume the application in the virtual phone. The size of the state varies, but can be minimized by the application developer. The main advantage is that the developers do not have to re-design the application, and it requires one-time deployment cost. However, the software infrastructure is more complex for the server, and it has to integrate additional mechanisms to deal with interactive applications.

In summary, each approach has its advantages and disadvantages. While most of today's mobile cloud applications belong to the traditional and terminal categories, we believe the suspend/resume ap-

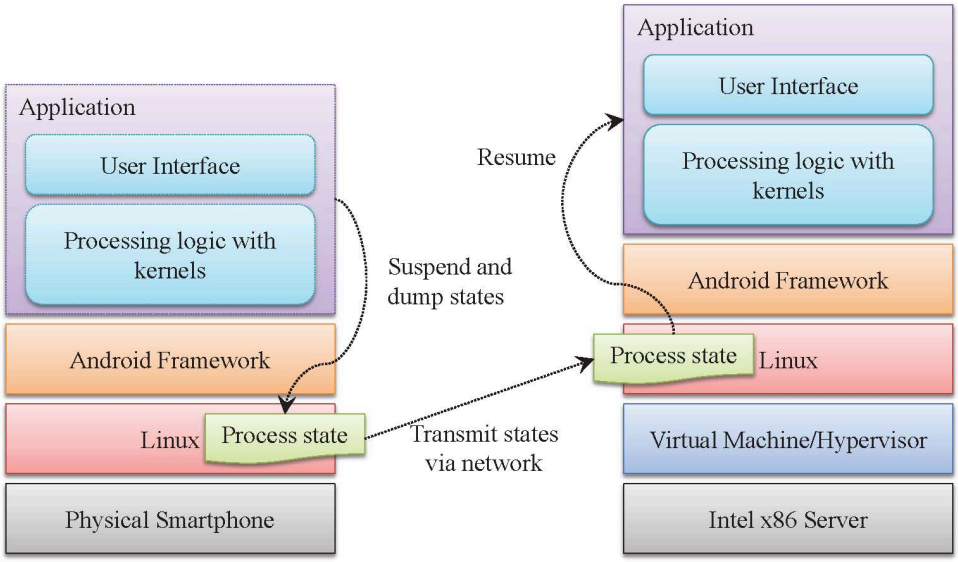

Figure 3: The suspend/resume migration approach

proach and the paradigm proposed in this paper should enable new classes of mobile cloud applications.

4 Case Study: Speech Recognition

As a case study for the proposed architecture and programming paradigm, we use a *voice dialer* application to demonstrate how this mobile application could be offloaded onto the cloud. The Android-based voice dialer allows the user to dial the phone number to a friend by speaking the friend's name to the smartphone. The application recognizes the speech from the user first, searches the phone book for the corresponded phone number, and dial the phone number. An automatic speech recognition (ASR) engine is used as the backend for processing the voice signal and translate it to text.

In order to save the computational cost of the voice dialer for the smartphone and improve its performance and accuracy, we make it a collaborative application based on our proposed architecture and paradigm. Section 4.1 introduces the domain knowledge for this application. Section 4.2 discusses the privacy concerns of this application when the ASR engine is offloaded. Section 4.3 analyzes the performance and the energy consumption of this collaborative application.

4.1 Domain Knowledge

ASR is a technology which converts the phrases or words spoken by human into text. As a mature technology, ASR has become an alternative input method on many mobile devices, complementing the other input methods operated by hands.Although the technology has been developed for years, the accuracy of ASR is still a major issue which prohibits it from being used as the primary input method for the users.

The accuracy of ASR depends heavily on the core technology adopted by the recognition engine, which can be classified into three types based on the dependence of the speaker, with different backend *hidden Markov models* (HMM) for building the recognition engines in each type:

- *Speaker-independent ASR (SI-ASR)* system: The HMM for the recognition engine is not speaker-specific in this system as the model is trained based on the voices from numerous speakers. The

system is most convenient to the end users, since they need not to participate in the initial training process. However, the performance of the system is the worst of the three systems.

- *Speaker-adaptation ASR (SA-ASR)* system: There is a built-in SI HMM *template* in the system, and the voice samples of a user can be adopted to fine-tune the configuration of the template specifically for the user. To the user, one SA-ASR system with a fine-tuned model may have better accuracy than an SI-ASR system. The major disadvantage is that the user has to provide the system with their voices, which could be a sensitive privacy issue. Nevertheless, without any fine-tuning, the performance of the system is exactly as same as an SI-ASR system.

- *Speaker-dependent ASR (SD-ASR)* system: Each user is required to supply his/her voice for training a HMM specifically to the user. The SD-ASR system takes time (from a few minutes to half an hour) to acquire user's voice and generate a HMM. For a SD-ASR system to support multiple users, it also requires significantly more data storage space to store the HMM for each user. In general, the performance of a SD-ASR system is better than the other two systems [14].

Today, most commodity ASR systems favor SI technologies for the convenience of the users, since they can start using the ASR system without any previous training. While SA/SD-ASR technologies may be used to improve the recognition accuracy, the user experience is often affected because a time-consuming training process is needed for the user to apply this method for each device.

On a mobile device of limited resources, a full-strength ASR engine often takes too much time and/or memory space. While a lightweight ASR engine may be used for recognizing voice commands with a limited vocabulary within seconds on a low-end mobile phone, demanding interactive applications such as dictation or gaming require instant responses. Thus, researchers have attempted to improve the efficiency and accurate of current ASR systems for mobile devices, but sometimes the efficiency improvement hurts the accuracy, which is a tradeoff in this case [9].

Another factor effecting the accuracy of ASR system on smartphone is to handle the background noise. In order to deal with the background noise, it increases the computation complexity and computational resources needed on the mobile device. Thus, it is hard for a mobile device to provide a PC-quality speech recognition service. While many smartphones today offer some sort of voice recognition capabilities, they work on limited application areas, such as voice dial, voice command, and phonebook search.

It can be beneficial for a mobile device to offload speech recognition onto a server machine to save execution time and conserve energy [21]. Today, many cloud-based services are offered to with low amortized operation costs [6]. This client-server model has worked quite successfully over the years. A well-know example is *Google Voice Search* [5], which was initially a tool developed by Google Labs for mobile phone to make a Google query. It has been integrated into Google Maps and Google Mobile App for the user to interact with the applications using voice. For Google Voice Search to work in an application, the mobile device records a voice message from the user and sends the recorded message to the Google's cloud service. The speech recognition is done completely on the server side to offload the burden from the mobile device. For the convenience of users, the speech recognition is performed by SI-ASR, so the users do not need to train their models before using the service.

In order to improve the accuracy, SA/SD-ASR system is considered to be adopted for better user experience, if users spend time for training their own model. However, it is a time-consuming effort for users, and we believe they would not be glad to perform it again. In order to address this inconvenience, we designed an ASR service which allows the applications on a mobile device to offload portions of a speech recognition task, model training/generation, and the data storage for storing the SA/SD models the cloud. It also allows multiple devices of the same user to share and populate one stored SA/SD model, so that the user no longer needs to repeat the lengthy training process on each device. This encourages the use of SA/SD-ASR on mobile devices for better recognition accuracy.

4.2 Privacy Concern

Once the users takes the training process, they will be asked to record about 200 phrases for generating the models. Because the procedure of model generation requires lots of storage and is too complex to be done on the smartphone, the process must be offloaded to the computer. Since the voice of the users are sent from the mobile device to the cloud, the users or the company which uses this application might worry about any misuse of their voice by the service provider.

To protect the private information, the user or the company may set up a dedicated server at home or at the company as *home cloud* – a similar concept with cloudlet, which can be deployed with very low cost – and use it to offload the model generation process for new SA/SD models. After finishing model generation, the new models could be stored on both the home cloud and the smartphone. The users may also store encrypted proprietary models on a public cloud, so that the application could retrieve the models from any smartphone for the users, but the public cloud should not perform the ASR functions in this case for privacy concerns.

4.3 Performance Evaluation

To characterize the performance for the voice dialer applications developed with different paradigms, we prototyped the application on the Android smartphone based on both the traditional approach and the proposed kernel-offload paradigm. We are in still the progress of prototyping the application using the terminal approach and the suspend/resume migration approach, so we calculated the latency of the applications with parameters actually measured from the two implemented prototypes. In the traditional approach, the recognizing server was implemented using Java threads and TCP/IP sockets. The model generator was implemented in the *HTK Speech Recognition Toolkit* [3], a portable toolkit developed for voice recognition research and applications. With our kernel-offload approach, we marked the ASR engine as a kernel function and implemented it in Java.

In our traditional version, we did not separate the kernel from the processing logic, and used the TCP/IP sockets to transmit the sampled voice signal from the smartphone to the server. Our kernel-offload version separated the kernel function as a Java runtime library, and transmitted the sampled voice signal via the stream created by `open` calls to distributed filesystem. The kernel function for the ASR engine was deployed automatically on the smartphone and a server, so that the voice dialer can still be used without the server.

For the performance testing, we installed the two voice dialers on the same smartphone, which was equipped with Android 2.2. The smartphone connected to the Internet via the combination of 802.11g network and an ADSL link with 256 Kbps up-link and 2 Mbps down-link. The smartphone had access to a server which has a four-core Intel Core i7 (Nehalem) processor with 4 GB memory. The server machine was located at the National Taiwan University in the same city and was connected to the Internet through the university' s high-speed network.

We ran a benchmark program on the client device to measure the latency of the voice dialers and used profiling tools to analyze the performance and power consumption at the same time. The voice dialer benchmark program used the ASR engine in the server by sending sampled voice signals and the lexicon candidates (the phone book) to the server and waited for the results from the server.

The cost of offloading the kernel function is evaluated by the performance and/or the energy consumption on the mobile device. For the kernel function K runs on the mobile device, $P_{local}(K)$ is defined as the execution time T_{local} from the start to the termination of the function, which is expressed by Equation 1. $P_{offload}(K)$ in Equation 2 denotes the execution time for offloading the kernel function onto the server, where T_{remote} is the execution time of kernel K on the remote server, $S_{rx/tx}$ is the amount of data exchanged between the mobile and the remote server, $B_{network}$ means the bandwidth of the network at

runtime for data transmission, $T_{latency}$ is the latency of the network, and $S_{tcpWindowSize}$ is the window size of TCP. The reduction of execution time G_p due to offloading kernel K is calculated by Equation 3. If $G_p(K) < 1$, then offloading kernel K is beneficial to the performance.

$$P_{local}(K) = T_{local} \tag{1}$$

$$P_{offload}(K) = T_{remote} + \frac{S_{rx/tx}}{B_{network}} + T_{latency} \times \lceil (1 + \frac{S_{rx/tx}}{S_{tcpWindowSize}}) \rceil \tag{2}$$

$$G_p(K) = \frac{P_{offload}(K)}{P_{local}(K)} \tag{3}$$

Energy consumption is another important factor on the mobile devices. The energy consumption for executing kernel K locally is expressed in Equation 4, by multiplying the execution time and the averaged power $W_{CPU-work}$ consumed by the processor during the kernel function. In the case of offloaded execution, the energy consumption is evaluated by Equation 5, which sums the needed energy to exchange the data with the server $W_{network}$, which usually depends on how many traffic via the network, and the energy $W_{CPU-idle}$ consumed by processor on the mobile device when it is idle, waiting for the results. The energy saving rate G_w in Equation 6 denotes the normalized energy consumption of the offloaded kernel to the one running on local. $G_w(K) < 1$ indicates that less energy is needed by using offloading scheme.

$$W_{local}(K) = T_{local} \times W_{CPU-work} \tag{4}$$

$$W_{offload}(K) = T_{remote} \times W_{CPU-idle} + W_{network}(S_{rx/tx}) \tag{5}$$

$$G_w(K) = \frac{W_{offload}(K)}{W_{local}(K)} \tag{6}$$

Since both performance and energy consumption affect the user experience, we may combine these two factors to decide if a kernel function should be offloaded. A model shown in Equation 7 could be used for modeling the cost-benefit of the offloading operation. Since there is sometimes a tradeoff between the power consumption and the performance, the user may adjust the weight ρ to favor energy saving or performance.

$$D(\rho, K) = G_w^{\rho}(K) \times G_p^{(1-\rho)}(K), \text{ where } 0 \leq \rho \in \Re \leq 1 \tag{7}$$

4.3.1 Performance Impact

Both the traditional and the kernel-offload versions of the recognition client are optimized in the processing logic for overlapping the voice recording process and the voice transmission to shorten the latency, as shown in Figure 4. The end detector, which is used to detect if the user begins to speak, was started at first. When the user started to speak, the first segment would be saved in the internal buffer. The buffer was transmitted to the server after it was filled. After the server received the segment, it passed to the ASR engine immediately for processing. After all segments were processed, the server checked the grammar rules and the lexical candidates. Finally, it returned the results to the client. The time used to process the voice segment and to lexicon/grammar check on the server was very short. In practice, the latency is dominated by the *network time*. In our testing environment, the results were shown on the screen almost immediately after the speaker said a 0.5-second-length word. The delay was less than 0.1 second in both versions. If not to offload the computation burden, the results were shown after about 1.5

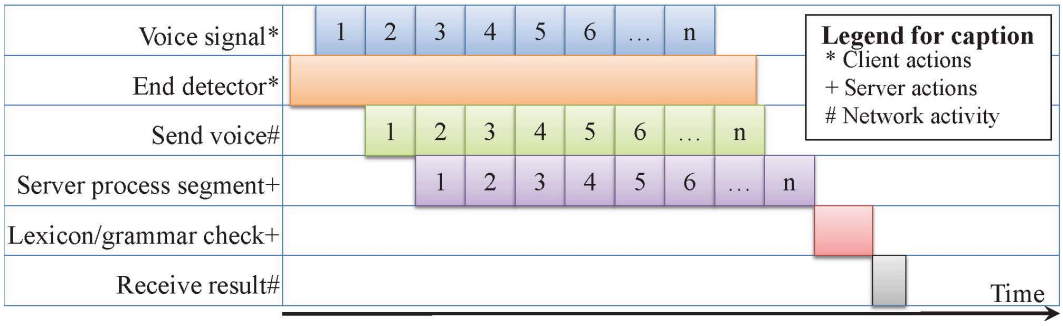

Figure 4: The Gantt chart for the actions on server and client sides

seconds, which is much slower than the offloaded cases. The reducing rate of execution time would be $G_p(K) = 0.1/1.5 = 0.067 < 1$.

For the suspend/resume migration version, we estimate its performance using an averaged 0.5-second (15.7 KB) sampled voice signal, and assume that the application can fully utilize the bandwidth of the uplink. The version has to dump the application state first and transmits the state to the server. The latency for dumping the application state is very short and can be ignored here, since the size of the application state is less than 1 KB, with an average of 249 bytes. Using the averaged size of the application state, we figure that the total time spent on transmission would be $(15.7 + 0.249)/32 + = 0.50$ seconds.

For the terminal version, the size of the uploaded data is as same as the traditional version, as both versions only need to transmit the voice signal. Additionally, the terminal version needs to receive from screen from the server. The latency is dominated by the network, the screen size and the updated area of the screen. We used a VNC client to connect to the virtual phone on the server to profile the network traffic as the user operated the voice dialer on virtual phone for 81 seconds. The statistics of packet size are shown Figure 5. The average packet size is 567.4 bytes, and the packet rate is about 126.3 packets per second in average. Hence, the downstream traffic within 0.5 seconds would be 35.0 KB in total. Thus, the latency could be evaluated by $15.7/32 + 35.0/256 = 0.63$ seconds.

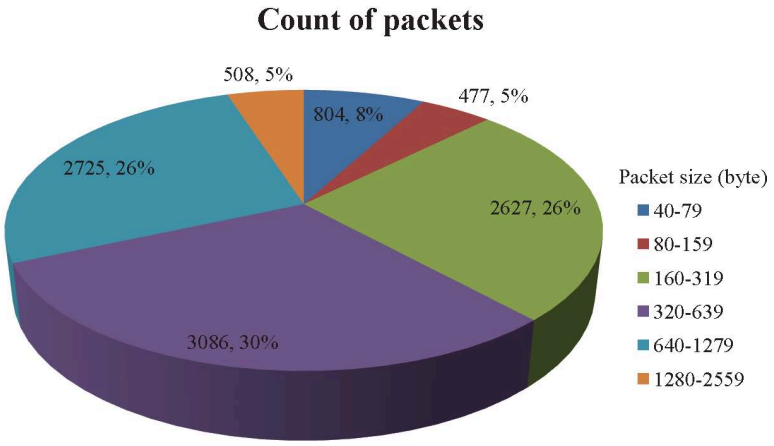

Figure 5: The statistics of the size of the packets transmitted by the VNC server to the VNC client for the voice dialer application

Let replace WiFi network with 3G UMTS cellular network, the average latency is 300 milliseconds, and the average bandwidth is 243 Kbps. The default TCP window size $S_{tcpWindowSize}$ in Android is 65536 bytes (64 KB). Thus, the transmission time for the 15.7 KB sampled voice signal would be $15.7/30.375 +$

$0.3 \times \lceil (1 + 15.7/64) \rceil = 1.17$ seconds. Based on the proposed application paradigm, the transmission could be overlapped by the user's speaking (0.5 second), so the total execution time for offloading kernel $T_{offload}$ would be $1.17 - 0.5 + 0.1 = 0.77$ seconds. For the suspend/resume migration approach, it would be $(15.7 + 0.249)/30.375 + 0.3 \times \lceil (1 + (15.7 + 0.249)/64) \rceil + 0.1 = 1.23$ seconds. And for terminal version, it would be $(15.7 + 35.0)/30.375 + 0.3 \times \lceil (1 + (15.7 + 35.0)/64) \rceil + 0.1 = 2.37$ seconds.

4.3.2 Power Consumption Impact

We characterize the power consumption on the client device when the client device switches from the local ASR engine to the remote ASR engine. Based on the power model of Android smartphone derived in [27], we derived an equation by considering CPU utilization, and WiFi utilization only as Equation 8, where Δ_{cpu} and Δ_{wifi} are the active time (in second) for CPU and WiFi respectively, and the meaning of other variables are shown in Table 1. To apply the equation, we actually measured the parameters listed in the table.

We found that the state of the WiFi network is a major factor which affected the power consumption. There are four states for WiFi: *low transmit*, *high transmit*, *low state (ls)*, and *high state (hs)*, and the state diagram is illustrated as Figure 6. According to the figure, we can find that the WiFi state changes dynamically by the number of network packets. Once the network packets are less than 8, the WiFi would change from high state to low state. If the packets are more than 15, it change from low state to high one. Both of *hs* and *ls* state enter the transmit state, which consumes 1000 mW, for 10-15 microseconds per second if there are data to send, and then return to the original state immediately. Because the time to transmit less than 15 packets is very short in low state, the power consumption when transmitting packets in low state is ignored. The WiFi state is controlled by the operating system at runtime.

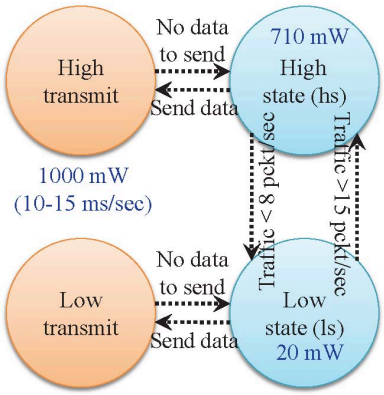

Figure 6: WiFi Power State Diagram

$$\begin{aligned} W(\Delta_{cpu}, \Delta_{wifi}) &= \Delta_{cpu} \times (\delta_{util} \times \beta_{cpu} + \delta_{on} \times \beta_{base}) \\ &\quad + \Delta_{wifi} \times (\delta_{ls} \times \beta_{wfls} + \delta_{hs} \times \beta_{wfhs}) + \delta_{hs} \times f_{wifi}(\delta_{speed}, \Sigma_{packet}) \end{aligned} \tag{8}$$

$$f_{wifi}(\delta_{speed}, \Sigma_{packet}) = \Sigma_{packet} \times (48 - 0.768 \times \delta_{speed}) \tag{9}$$

Based on our observation, the number of transmitted network packets in the whole offloading process was about 9 to 11 for both traditional and our kernel-offload versions, which would not make the WiFi change the state. In other words, the power consumption depended on the current state of WiFi for using

Table 1: HTC Dream power model

Component	Variable	Range	Power coefficient
CPU	δ_{util}	$1-100$	$\beta_{cpu} = 4.34$
	δ_{on}	$1,0$	$\beta_{base} = 121.46$
WiFi	Σ_{packet}	$1-\infty$	n/a
	δ_{speed}	$1-54$	$f_{wifi}(\delta_{speed}, \Sigma_{packet})$
	δ_{ls}	$1,0$	$\beta_{wfls} = 20$
	δ_{hs}	$1,0$	$\beta_{wfhs} = 710$

speech recognition remotely. Thus, two situations under different states should be considered as the best and the worst case. Assuming it took about 0.5 second to speak a name, the end detector would take about 30% CPU utilization, and WiFi would run under 54 Mbps speed. The amount of power consumption in low state and in high state state are shown as Equation 10 and 11. As we mentioned in our latency test, it took about one second of CPU time to recognize a phrase with the local ASR engine on the client device, and the total amount of power consumption in this case could be computed by Equation 12. Compared to the local voice recognition, the offloading process was able to save 19.8% to 80.0% of power consumption for the client device. The maximum and minimum power saving rate would be $G_w(K) = 135.83/681.29 = 0.200 < 1$, and $G_w(K) = 546.11/681.29 = 0.802 < 1$.

$$\begin{aligned} W_{offload-ls}(0.5,0.5) &= 0.5 \times (30 \times 4.34 + 1 \times 121.46) \\ &\quad +0.5 \times (0 \times 710 + 1 \times 20) \\ &= 135.83\text{mW} \end{aligned} \tag{10}$$

$$\begin{aligned} W_{offload-hs}(0.5,0.5) &= 0.5 \times (30 \times 4.34 + 1 \times 121.46) \\ &\quad +0.5 \times (1 \times 710 + 0 \times 20) + 1 \times f_{wifi}(54,10) \\ &= 546.11\text{mW} \end{aligned} \tag{11}$$

$$\begin{aligned} W_{local}(0.5,0) &= 0.5 \times (30 \times 4.34 + 1 \times 121.46) \\ &\quad +1 \times (100 \times 4.34 + 1 \times 121.46) \\ &= 681.29\text{mW} \end{aligned} \tag{12}$$

In this case study under the WiFi network, both $G_p(K) < 1$ and $G_w(K) < 1$, thus $D(\rho, K) < 1$, where $0 \leq \forall\rho \in \Re \leq 1$, which means the speech recognition can always be benefited by offloading the computation kernel – the ASR engine – to remote for speedup and energy saving.

4.3.3 Power Management Policy Impact

In the above case, because the bandwidth of up-link is broad enough for transmitting the voice signal in the period when the user is speaking, the latency from the network transmission are completely overlapped and hidden under this condition. In order to highlight the impact of the network bandwidth and latency, we evaluated the round-trip-time in this case assuming the smartphone go to the Internet via 3G UMTS network, whose latency is 300 milliseconds. An alternative equation is used for evaluated the

performance in this case, which totals the processing time of remote server and the network transmission time, and then takes away the overlap time of user's speaking, as shown in Equation 13.

$$T_{offload-ASR} = 0.1 + \frac{15.7}{B_{network}} + T_{latency} \times (1 + \lceil \frac{15.7}{S_{tcpWindowSize}} \rceil) - min(\frac{15.7}{B_{network}}, 0.5) \quad (13)$$

The power consumption of 3G UMTS network modeled in previous work [27] depends on only the active time only with a consuming rate 570 mW per second. Hence, the power model in the ASR case via 3G network can be expressed by Equation 14.

$$W_{offload-ASR} = 0.5 \times (30 \times 4.34 + 1 \times 121.46) + 570 \times \frac{15.7}{B_{network}} \quad (14)$$

Thus, the offloading evaluation model could be calculated by Equation 15, and the relation between the network bandwidth $B_{network}$ and the weight ρ of power saving is illustrated as Figure 7 according to the given equation, where the TCP window size $S_{tcpWindowSize}$ is 64 KB, the network latency $T_{latency}$ is 300 milliseconds, the network bandwidth $B_{network}$ is evaluated from 1 Kbps to 128 Kbps, and the weight *rho* is from 0 to 1. According to the result, the offloading process is always taken when $\rho > 0.700$ even the bandwidth is only 4 Kbps, which causes extreme high latency about 31 seconds for the results, which we guess this is an unacceptable latency for users which would hurt the user experience. In practice, in order to control the latency would not be lengthened exceeding 20%, we recommended limited user to adjust ρ only from 0 to 0.1.

$$D(\rho, K) = (\frac{W_{offload-ASR}}{W_{local}})^{\rho} \times (\frac{T_{offload-ASR}}{T_{local}})^{(1-\rho)}, \text{where } T_{local} = 1.5 \quad (15)$$

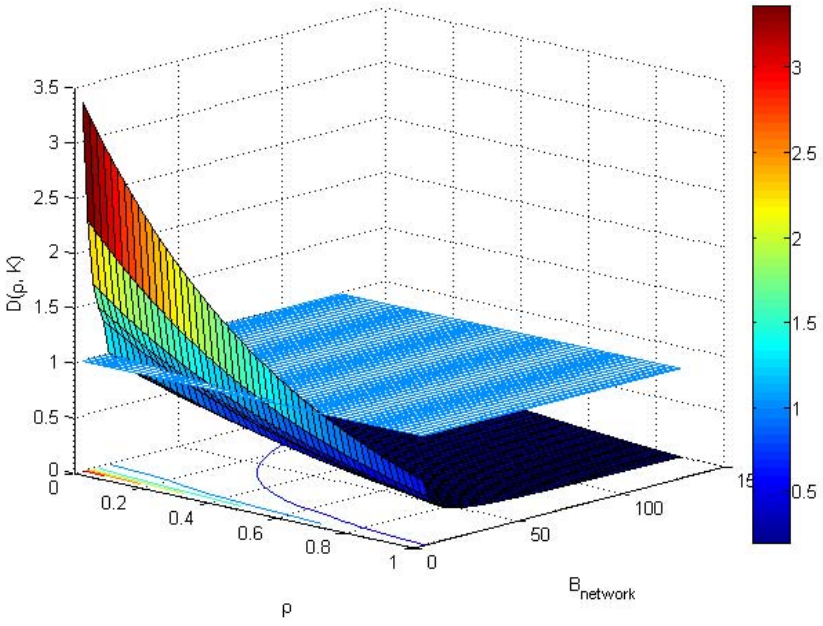

Figure 7: The relation between the weight of power saving and the network bandwidth for the ASR case

5 Conclusion and Future Work

In this paper, we presented an architecture and a paradigm for developing collaborative applications with minor modifications to today's mobile and cloud computing infrastructures. We also provided a performance/power evaluation model to determine whether offloading a kernel function benefits the application. The privacy and security issues are also addressed by the option to set up a personal (home) cloud service for offloading the kernel functions in the mobile application. We believe that the proposed paradigm addresses the above concerns effectively without having to modify existing systems and applications.

We are still in progress of developing the software infrastructure to support the proposed paradigm. The speech recognition case study illustrates the paradigm and represents some promising results with our progress so far. From the case study, we see the most time-consuming function in the voice dialer application can be offloaded efficiently to a server in the cloud simply by marking the functions as a kernel function, and the system will automatically offload the kernel function to a server for reducing its response time and the power consumption on the smartphone.

We are currently researching on the trust models for mobile cloud computing, so that mobile devices may find trustworthy computational resources nearby. Meanwhile, we would like to develop more collaborative applications based on the proposed kernel-offload paradigm and integrate techniques to enhance the infrastructure and benefit more applications.

Acknowledgment

This research was supported by Ministry of Economic Affairs in Taiwan under grant: MOEA 99-EC-17-A-01-S1-034, and by National Science Council in Taiwan under grants: NSC 99-2219-E-002-029, NSC 99-2220-E-002-026, and NSC 99-2220-E-002-028.

References

[1] Google App Engine - Google Code. `http://code.google.com/intl/en/appengine/`.

[2] Google Map API Family - Google Code. `http://code.google.com/intl/en/apis/maps/index.html`.

[3] HTK speech recognition toolkit. `http://hbase.apache.org/`.

[4] RealVNC - VNC remote control software. `http://www.realvnc.com/`.

[5] Voice search. `http://www.google.com/mobile/voice-search/`.

[6] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. Above the clouds: A Berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, February 2009.

[7] Rajesh Balan, Jason Flinn, M. Satyanarayanan, Shafeeq Sinnamohideen, and Hen-I Yang. The case for cyber foraging. In *Proc. of the 10th ACM SIGOPS European Workshop (EW'02), Saint-Emilion, France*, pages 87–92. ACM Press, July 2002.

[8] Philip A. Bernstein, Vassos Hadzilacos, and Nathan Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.

[9] Xu Chao, Liu Yi, Yang Yongsheng, P. Fung, and Cao Zhigan. A system for Mandarin short phrase recognition on portable devices. In *Proc. of the 4th International Symposium on Chinese Spoken Language Processing (ISCSLP'04), Hong Kong*, pages 229–232. IEEE, December 2004.

[10] B. G. Chun and P. Maniatis. Augmented smartphone applications through clone cloud execution. In *Proc. of the 12th Workshop on Hot Topics in Operating Systems (HotOS'09), Monte Verità, Switzerland*, pages 8–8, May 2009.

[11] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. MAUI: making smartphones last longer with code offload. In *Proc. of the 8th International Conference on Mobile Systems, Applications, and Services (MobiSys'10), New York, NY, USA*, pages 49–62. ACM Press, June 2010.

[12] Jason Flinn, Dushyanth Narayanan, and M. Satyanarayanan. Self-tuned remote execution for pervasive computing. In *Proc. of the 8th Workshop on Hot Topics in Operating Systems (HotOS'01), Elmau/Oberbayern, Germany*, pages 61–66. IEEE, May 2001.

[13] Scott Garriss, Rámon Cáceres, Stefan Berger, Reiner Sailer, Leendert van Doorn, and Xiaolan Zhang. Trustworthy and personalized computing on public kiosks. In *Proc. of the 6th International Conference on Mobile Systems, Applications, and Services (MobiSys'08), New York, NY, USA*, pages 199–210. ACM Press, June 2008.

[14] X. Huang and K.F. Lee. On speaker-independent, speaker-dependent, and speaker-adaptive speech recognition. *IEEE Transactions on Speech and Audio Processing*, 1(2):150–157, April 1993.

[15] Shih-Hao Hung, Tei-Wei Kuo, Chi-Sheng Shih, Jeng-Peng Shieh, Chen-Peng Lee, Che-Wei Chang, and Jie-Wen Wei. A cloud-based virtualized execution envrionment for mobile applications. *ZTE Communications*, 9(1):15–21, 2011.

[16] Christopher Clark Keir, Christopher Clark, Keir Fraser, Steven H, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *Proc. of the 2nd ACM/USENIX Symposium on Networked Systems Design and Implementation (NSDI'05), Boston, Massachusetts, USA*, pages 273–286. ACM Press, May 2005.

[17] K. Kumar and Yung-Hsiang Lu. Cloud computing for mobile users: Can offloading computation save energy? *Computer*, 43(4):51–56, April 2010.

[18] C. Lattner and V. Adve. LLVM: a compilation framework for lifelong program analysis transformation. In *Proc. of the International Symposium on Code Generation and Optimization (CGO'04), Palo Alto, California, USA*, pages 75–86, March 2004.

[19] Chris Mitchell, editor. *Trusted Computing*. The Institution of Engineering and Technology, 2008.

[20] Steven Osman, Dinesh Subhraveti, Gong Su, and Jason Nieh. The design and implementation of Zap: A system for migrating computing environments. In *Proc. of the 5th Symposium on Operating Systems Design and Implementation, Boston, Massachusetts, USA*, pages 361–376, December 2002.

[21] Alexey Rudenko, Peter Reiher, Gerald J. Popek, and Geoffrey H. Kuenning. Saving portable computer battery power through remote process execution. *ACM SIGMOBILE Mobile Computing and Communications Review*, 2(1):19–26, January 1998.

[22] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Caceres, and Nigel Davies. The case for VM-based cloudlets in mobile computing. *IEEE Pervasive Computing*, 8(4):14–23, October 2009.

[23] Mahadev Satyanarayanan, Benjamin Gilbert, Matt Toups, Niraj Tolia, Ajay Surie, David R. O'Hallaron, Adam Wolbach, Jan Harkes, Adrian Perrig, David J. Farber, Michael A. Kozuch, Casey J. Helfrich, Partho Nath, and H. Andres Lagar-Cavilla. Pervasive personal computing in an Internet suspend/resume system. *IEEE Internet Computing*, 11(2):16–25, 2007.

[24] J.E. Stone, D. Gohara, and Guochun Shi. OpenCL: A parallel programming standard for heterogeneous computing systems. *Computing in Science Engineering*, 12(3):66–73, May-June 2010.

[25] Michael Stonebraker, editor. *Readings in database systems*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd edition, 1994.

[26] Li Xiong and Ling Liu. PeerTrust: supporting reputation-based trust for peer-to-peer electronic communities. *IEEE Transactions on Knowledge and Data Engineering*, 16(7):843–857, July 2004.

[27] Lide Zhang, Birjodh Tiwana, Zhiyun Qian, Zhaoguang Wang, Robert P. Dick, Zhuoqing Morley Mao, and Lei Yang. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *Proc. of the 8th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES/ISSS'10), Scottsdale, Arizona, USA*, pages 105–114. ACM Press, Octoboer 2010.

Shih-Hao Hung joined the Department of Computer Science and Information Engineering at National Taiwan University as an assistant professor in 2005. His research interests include cloud computing, parallel processing, embedded systems, and pervasive applications. He worked for the Performance and Availability Engineering group at Sun Microsystem Inc. in Menlo Park, California ('00-'05) after he completed Post-Doc ('98-'00), Ph.D. ('94-'98) and M.S. ('92-'94) training in University of Michigan, Ann Arbor. He graduated from National Taiwan University with a BS degree in Electrical Engineering in '89.

Yu-Shuo Chang is a graduate student in Department of Computer Science and Information Engineering, National Taiwan University since 2009 fall, and he received the B.S. degree from Department of Computer Science and Information Engineering, National Changhua University of Education at June in the same year. His research interests include cloud computing, grid computing, parallel and high performance computing.