Cloudlet-based Multi-lingual Dictionaries

Achanta Vamsi Subhash  vamsi.subhash@iiitb.org
Avinash A  Avinash.A@iiitb.org
M L Sahitya  mlakshmi.sahitya@iiitb.org
Nishanth TS  Nishanth.TS@iiitb.org
Veena Thomas  Veena.Thomas@iiitb.org

Technical Report IIITB-TR-2012-002
April 2012
Abstract

Cloud computing is the current day technology which helps mobile users compute resource hungry applications on a remote cloud and return the result to the mobile. The drawback is that the mobile always depends on the Wireless Area Network (WAN) connectivity to the Internet. WAN delays in the critical path of user interaction can hurt usability by degrading the crispness of system response. Though we may use mobile internet for tasks such as web browsing, it is difficult for continuous bandwidth hogging applications. Cloudlets help solve the problem with the benefits of cloud computing without being WAN-limited. Cloudlets can be thought as a “datacenter in a box”. With the concept of Virtual Machine (VM) based technology called “Dynamic VM Synthesis”, the nearby cloudlet can be used by a mobile user to perform his resource intensive task. A multilingual dictionary with 6 supported languages that does language to language translation is the functionality of the mobile application which we have developed. This application resides in the mobile and when intended can be run on a nearby cloudlet if available. Lack of Internet connectivity and poor bandwidth are not an obstacle for a cloudlet based application. Hence, the application could be of use in scenarios where a cloud based solution is not apt or feasible.

Project URL : https://sourceforge.net/projects/cloudlet6a/files/

©2012 Achanta Vamsi Subhash, Avinash A, M Lakshmi Sahitya, Nishanth TS, Veena Thomas. This material is available under the Creative Commons Attribution-NonCommercial-Share Alike License. See http://creativecommons.org/licenses/by-nc-sa/3.0/ for details.
Acknowledgment

We would like to convey our deep sense of gratitude to Dr. Shrisha Rao, for his valuable suggestions and guidance. It was our immense pleasure working under his guidance.
Contents

1 Introduction .............................................. 6

2 Project Description ...................................... 6

3 Architecture ............................................ 8
   3.1 Dynamic VM Synthesis ................................. 8
   3.2 VM Overlay Creation .................................. 9
   3.3 Binding to Cloudlet Infrastructure ................. 10

4 Implementation Details .................................. 11
   4.1 Cloudlet Virtualization ............................... 11
   4.2 Client Cloudlet Interaction ......................... 13
   4.3 Client Cloudlet Interaction ......................... 13
   4.4 Scheduler ............................................ 14
   4.5 Dataset Generation .................................... 14

5 Results .................................................. 15
   5.1 Screenshots ........................................... 15
   5.2 Performance Metrics ................................. 16
## List of Figures

1. Dynamic VM Synthesis ........................................... 8
2. Architecture ..................................................... 10
3. Multi-Lingual Dictionary Application ............................ 12
4. Android Application ............................................... 15
5. Input to the Application .......................................... 16
6. Output of the Application ......................................... 16
7. Server Side Display .............................................. 17
8. Response time for 3 requests on three VMs ...................... 17
1 Introduction

Mobile devices have brought various functionalities to our finger tips. Mobile devices which are small in size suffer from the problem of resource poverty. This acts as a constraint limiting the class of applications that can be run on these devices.

In order to overcome this drawback and allow the mobile users to use a wide variety of applications various alternate methods to obtain resources have been devised. One of the methods is to connect to a cloudlet. This enables the mobile devices to make use of the resources of the cloudlet in order to run huge applications.

In this vision, mobile users seamlessly utilize nearby computers to obtain the resource benefits of cloud computing[1] without incurring WAN delays, jitter, congestion, and failures. Crisp interactive response for immersive applications is much easier to achieve because of the cloudlet’s proximity.

The focus of our project is a multilingual dictionary application running on a mobile device. This application allows a user to give the input either as text or voice and look up its corresponding meaning. The application makes use of the resources of a nearby cloudlet in order to perform the required parsing and displays the results to the user.

In our implementation the mobile user exploits Virtual Machine (VM) technology to rapidly instantiate the service (multilingual dictionary) on a nearby cloudlet, and then uses that service over a wireless LAN.

2 Project Description

Objective

The project is aimed at developing a cloudlet based multilingual dictionary service with six languages. The user can enter a word in any language and view its corresponding meaning in any other language of his choice. The mobile device is used to enter the word the user wants to look up, the computing is done at the cloudlet and resultant meaning is sent back to the mobile phone.

Gap Analysis

Mobile computing [2] means using portable devices to run stand-alone applications and/or accessing remote applications via wireless networks. Mobile devices can execute a resource intensive application on a distant high per-
formance computer server or compute cluster and support the client user interactions with application over the internet. This may lead to long WAN latencies which are a fundamental obstacle. WAN delays in critical path of user interaction can effect usability by degrading the crispness of system response even for trivial applications. As latency increases interactive response suffers making the immersive tasks jerky or sluggish to the point of distraction.

Cloud computing even with progress in network bandwidth, security, energy efficiency, manageability, latency is unlikely to improve dramatically or may even worsen although bandwidth continues to improve over time. Cloud has both hard and soft states which need to be professionally administered all the time. It requires high maintenance due to its high power consumption. Clouds have centralized ownership like Amazon and Yahoo. They have internet latency or bandwidth. Thousands of users of the same application will be accessing the cloud at the same instance of time.

The current technological solution for poor mobile computational performances is cloud computing. But cloud computing suffers from the problems of WAN latency [3]. A cloudlet is one of the solutions to this problem. M Satyanarayanan et al. [3] refers to cloudlet as a “data center in a box”. The differentiating features of cloud and cloudlet are as follows:

1. A cloudlet has only soft state where as a cloud has both hard and soft states.

2. A cloudlet is self-managed where as a cloud is professionally administered and is available 24×7.

3. Any computer/device can act as a cloudlet if configured but a cloud needs a fixed infrastructure with huge resources.

4. A cloudlet can be owned by local businesses whereas a cloud is owned by centralized vendors like Yahoo!, Amazon.

5. Only a few users may be able to use a cloudlet at a time where as a cloud can supports thousands of users at a time.

If no cloudlet is available nearby, the mobile device can gracefully degrade to a fallback mode that involves a distant cloud or, in the worst case, solely its own resources. Full functionality and performance can return later, when a nearby cloudlet is discovered.
3 Architecture

Rather than relying on a distant cloud, resource poverty of mobile device can be addressed by using a resource rich cloudlet.

*Cloudlets are decentralized and widely dispersed Internet infrastructure whose compute cycles and storage resources can be leveraged by nearby mobile devices [3].*

The multi-lingual dictionary is implemented using a cloudlet allows transient customization[4] of the infrastructure, by making use of hardware VM technology.

3.1 Dynamic VM Synthesis

The VM is delivered to the infrastructure dynamically. The VMs have to be synthesized rapidly, as users who rely on cloudlet services will find it unacceptable if extended delays for service initiation in a new location occurs. The various steps in this process are shown in Figure 1 and are also described below.

![Image of Dynamic VM Synthesis](image)

*Figure 1: Dynamic VM Synthesis*
3.2 VM Overlay Creation

A small VM overlay (dictionary overlay VM) is present in the mobile device. A base VM is present in the cloudlet infrastructure.

2. The dictionary overlay VM from the mobile device is delivered to the cloudlet infrastructure that already possesses the base VM.

3. The cloudlet infrastructure applies the dictionary overlay VM to the base VM in order to derive the ‘launch VM’.

4. The ‘launch VM’ starts execution of the dictionary application in the precise state from which the overlay was derived.

5. Once the user’s dictionary application is completed, the result is sent back to the mobile device.

The creation of the VM overlay is explained in the following section. In the multi-lingual dictionary application, the software in the launch VM is a server that receives the input text (for meaning, input language, output language) from the mobile device, performs the search for meaning and returns the output meaning in the desired language to the mobile device.

3.2 VM Overlay Creation

We use Virtual Box, a hosted Virtual Machine Manager for Linux. BaseVM is a VM in which a minimally configured guest OS has been installed. The process of VM Overlay creation is as follows:

1. The tool launches baseVM, and then executes install-script in the guest OS. The result is a VM that has been configured for use by the mobile device.

2. The tool launches the desired application, and brings it to a state that is ready for user interaction. This VM is called launchVM.

3. The launchVM is suspended. It can be resumed rapidly at runtime without delays of guest reboot or application launch.

4. After creating launchVM, xdelta differences its memory and disk images with those of baseVM to obtain the VM overlay.

5. The VM overlay is then compressed and encrypted.
3.3 Binding to Cloudlet Infrastructure

The controller of the transient binding between mobile device and cloudlet is a user level process. Figure 2 depicts the architecture of the binding process. The process of binding to the cloudlet infrastructure is as follows:

1. A TCP connection is established between the device and the cloudlet.
2. The connection is then used for the remaining communication, which involves sending the input and receiving the output (correct meaning).
3. After the connection is established and the mobile device sends the input and the VM overlay, the cloudlet executes a script.
4. This script applies the overlay to the baseVM.
5. The suspended VM is then launched, and is ready to provide services to the mobile device.
6. Once the suspended VM is in the running state, the dictionary application is executed and the output is sent back to the mobile device.
4 Implementation Details

The dictionary application (Figure 3) runs on the mobile device. The application takes the following input from the user:

- Input language: the language in which the word is specified.
- Output language: the language in which the meaning has to be displayed.
- Word: the word whose meaning has to be retrieved. The user can either give a speech input or a text input.

If the input is in the form of speech, the voice is stored in a .wav file and is sent to the cloudlet for converting into text. The converted text is sent back to the user for confirmation. If the user accepts, then the word is taken as the input for the dictionary application. The cloudlet parses the data of the input language dictionary, looking for the specific input word and retrieves the meaning in the output language specified. Once the meaning (output) is retrieved, it is sent back to the mobile device. The mobile device in turn displays the meaning to the user. The mobile application contains either the patch file generated using xdelta as described in section [4.1], or a URL to the location from where the patch file could be downloaded. The patch file contains the dictionary dataset. The patch file, when applied to a clean VM, loads the VM to a state which when resumed contains the loaded dictionary application. The input is then fed to the application inside the VM and the corresponding output (meaning of the word) is then sent back to the base OS and then to the corresponding mobile user.

4.1 Cloudlet Virtualization

The Guest OS used is Ubuntu LTS which is installed inside Virtualbox. The following steps have to be followed to setup the cloudlet:

1. The Guest OS in its original state is a clean Ubuntu installation configured with compilers. It does not have any installed applications supported by the cloudlet.

2. A snapshot (called init) of the VM in its clean state is taken using the VBoxManage snapshot command [5].

3. The Dictionary application (containing the dictionary program and the 30 datasets and a shell script to run the program) is installed in the
4.1 Cloudlet Virtualization

**Figure 3: Multi-Lingual Dictionary Application**

Guest OS and a snapshot (called end) of the Guest OS with the installed application is taken using the VBoxManage snapshot command.

4. The difference between the two snapshots init and end is taken. This is done using the xdelta delta command [6]. This difference is called a patch file.

5. A .sav file is generated from the patch file using the xdelta command. This .sav file is the difference between the patch and the init snapshot.

6. The .sav file created in the previous step is applied to the Guest OS using the VBoxManage adopt state command [5].

7. Now, the Guest OS inside the VM contains the Dictionary application.

8. The shell script contained in the Guest OS (part of the Dictionary application) is executed in the Guest OS using the VBoxManage guest-control command. The output returned is returned back to the base OS and then to the client.

9. After the script is executed the Guest OS VM is powered off using the VBoxManage controlvm command.
4.2 Client Cloudlet Interaction

The interaction between the client and cloudlet is the main task, it is performed as follows:

1. The client requests to connect to the cloudlet. The cloudlet replies with a response. The client then establishes a connection with the cloudlet.

2. Each client has a copy of the patch file. After a client has connected to the cloudlet, the cloudlet checks if it already has a copy of the patch file, if it does then the client does not have to send the patch file, else the client sends the patch file.

3. The patch file is placed in a folder which is shared between the Base OS and the Guest OS. The client also sends an input file which contains the input language, output language and the search word.

4. The input file sent by the client is copied to the Guest OS.

5. A script file is executed which applies the patch to the Guest OS in the clean state and starts the Guest OS. This script executes another script which runs the dictionary program in the Guest OS.

6. The output (meaning of the search word) generated by the program is written into a file called output.txt. This file is copied into the shared folder. The contents of this file are read and the contents are sent back to the client.

4.3 Client Cloudlet Interaction

1. The client requests to connect to the cloudlet. The cloudlet replies with a response. The client then establishes a connection with the cloudlet.

2. Each client has a copy of the patch file. After a client has connected to the cloudlet, the cloudlet checks if it already has a copy of the patch file, if it does then the client does not have to send the patch file, else the client sends the patch file.

3. The patch file is placed in a folder which is shared between the Base OS and the Guest OS. The client also sends an input file which contains the input language, output language and the search word.

4. The input file sent by the client is copied to the Guest OS.
5. A script file is executed which applies the patch to the Guest OS in the clean state and starts the Guest OS. This script executes another script which runs the dictionary program in the Guest OS.

6. The output (meaning of the search word) generated by the program is written into a file called output.txt. This file is copied into the shared folder. The contents of this file are read and the contents are sent back to the client.

### 4.4 Scheduler

The test system used is a Lenovo ThinkPad Edge E420 with an Intel core i5 processor and 4GB of RAM. Ubuntu 12.04 is the OS used and there are three clone VMs in Virtualbox, each running Ubuntu. The main emphasis of scaling the cloudlet is to make it handle more requests from the mobile users.

1. We employ a scheduler at the cloudlet which schedules the VMs in FIFO order.

2. The cloudlet has a socket-server program listening to the client's requests. For each request from the client, the socket-server program creates a thread object and stores it in a ready queue.

3. A thread object in the ready queue is scheduled if one of the VMs in the cloudlet is in power-off state. Else, it is put in wait state.

4. The threads are synchronized to support concurrency. Also, they read and write to separate files.

5. If scheduled, the patch file is applied to the VM and input is fed to the VM after successful launch. The VM writes the output to the corresponding output file.

6. The output is then sent back to the corresponding client by the socket-server program. Scheduling remains the key when multiple requests are to be handled by the system. The requests are handled synchronously by using Threads and maintaining concurrency.

### 4.5 Dataset Generation

The dictionary application input dataset consists of language to language word translation files. We currently support a total of 6 languages: English,
Hindi, Telugu, German, Spanish and French. Hence there are 30(6P2) permutation files. The format of each file is a word in source language, followed by a colon, followed by a word in the target language. The dataset is generated by taking a set of English words (around 1, 13,000) from Startdict dictionary. Then the words are grouped into a collection of 4000 words each. Each group is then given as an input to online Google Translator [7] which translated it to the target language specified. The process is repeated for all the groups and all the six languages.

5 Results

5.1 Screenshots

This is the first screen (Figure 4) of the android application[8]. This screen is displayed when the user starts the application. The user has to click on the Start Application button to go to the next screen.

In the next screen (Figure 5) the user has to enter the language of the word whose meaning he wants to search and the output language in which he wants the meaning to be displayed. On clicking the NEXT button the user is taken to the next screen of the application.

Now we come to the final screen of the android application (Figure 6). In this screen the user has to enter the word whose meaning he wants to search. After entering the word the user clicks the Compute button which triggers the cloudlet application which in turn returns the meaning to the android application. The meaning is displayed on the screen as shown in the figure.
5.2 Performance Metrics

Here we compute the time taken for every virtual machine to run our application and render the result. The Figure 8 graph shows the analysis for three virtual machines used to get the result based on the number of user at that instance.

In our implementation we have used three virtual machines and we have used FIFO (First in First out) scheduling algorithm, the resources of our cloudlet is 4 GB of RAM and Intel Core i5 Processor. Figure 8 graph depicts
that the time taken for 1 request is 14 seconds, for two requests is 17 seconds and three requests is 24 seconds, and if there is a fourth request it should wait in the ready queue until any of the VM is free. So, if 4 requests comes at a time three requests will be allocated a VM and 4th request will be waiting until any of the VM stops execution.

Therefore, for every three queries to be executed it will take approximately 24 seconds.
References


