2013 IEEE International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing

CloudAP: Improving the QoS of Mobile Applications with Efficient VM Migration

Yunkai Zhang, Renyu Yang, Tianyu Wo, Chunming Hu, Junbin Kang, Lei Cui
State Key Laboratory of Software Development Environment
Beihang University
Beijing, China
{zhangyk, yangry, woty, hucm, kangjb, cuilei}@act.buaa.edu.cn

***Abstract* — Mobile computing is increasingly growing in terms of massive computation as well as user demand and use of mobile devices. Remote execution techniques enrich the service experience of mobile devices by leveraging the resource pools of computation and storage capabilities of the cloud data center. However, the user experience and quality of service will be severely affected due to the inherent high latency and low bandwidth of a WAN environment. In this paper, we introduce a cloud base station "*CloudAP*" which is a small-scale computing infrastructure close to mobile users with local network access. In addition, we present a two-tier architecture consisting of *CloudAP* and Cloud center and show how to synthesize them to form a general computing environment. Furthermore, we propose a prompt execution environment migration scheme implemented by an efficient whole-system VM migration. It makes the execution environment move following the location of mobile device. Our experimental results demonstrate that the proposed architecture is effective and the execution environment migration scheme is efficient, consisting of up to 10ms and 30s for service downtime and execution environment switch time respectively. These improvements make vital contributions to user experience and QoS in mobile pervasive environment.**

Keywords —Mobile computing, Execution environment, Whole-system VM migration

I. INTRODUCTION

In recent years, smartphones experience an increasingly prevailing trend. It has been reported that the amount of smartphone users worldwide exceeded 1 billion in the third quarter of 2012 and continues to grow sharply. People increasingly use their smartphones for a variety of tasks, such as gaming, navigation, software services etc. However, the intrinsic limitations of mobile devices such as low battery volume or poor resource configuration impede further facilitation of these applications.

The emergence of Mobile computing has resulted in overcoming the challenge of resource restrictions of mobile devices by using Cloud resources. Cloud computing provides elastic resource management and application hosting, compensating for the resource poverty of mobile devices and growing demand of mobile users. In particular, it is the remote execution technique that makes utilizing computer software on mobile devices feasible. By leveraging OS-level virtualization techniques and cloud infrastructures, resource intensive applications can be executed within cloud infrastructure and the results are transmitted to the thin device client. Nevertheless, the WAN latency is still a non-negligible factor and can damage the usability drastically by degrading the system response [3].

To cope with this high latency, Mahadev et al. [2] introduce the concept of *Cloudlets*: trusted, resource rich computers or clusters in the vicinity of mobile users. Mobile users can then rapidly instantiate custom virtual machines (VMs) on the *Cloudlet* where the required software executes. However, the service's continuous execution is not guaranteed due to the lack of integration between *Cloudlets* and Cloud. For example, when a user using software on a mobile device moves to a *Cloudlet* area, the user has to discard his current execution environment before instantiating a new one in the *Cloudlet*. Furthermore, the time required to customize a VM in the *Cloudlet* is relatively long (taking 60-90 seconds). Tim et al. [12] and Rahimi et al. [10] extend the concept of *Cloudlet*, but they have not yet solved the problems introduced by *Cloudlet*. Thus, there is a critical need for a mechanism that focuses on how to keep the service uninterrupted and conserve the execution environment transparently in order to facilitate a more pervasive and seamless access to software service for mobile users.

In this paper, we propose the concept "*CloudAP"* inspired by the access point in communication fields representing a small-scale cloud infrastructure close to the mobile devices and advocate an efficient 2-tier architecture for remote software execution with both Cloud center and *CloudAP* taken into consideration. Furthermore, we present an efficient execution environment migration scheme, which is implemented by whole-system VM migration, to make the execution environment move according to the location shift of mobile devices. We perform experiments to measure the feasibility of the presented architecture and evaluate the efficiency of the execution environment migration scheme. Our experimentations demonstrate that the "*CloudAP* + Cloud center" architecture can improve the Quality of Service (QoS) of software service significantly especially in the interaction delay. Moreover, the switch time of execution environment can be limited to 24-30 seconds, while the service downtime only lasts for roughly 10 milliseconds. In this way, the service interruption time is diminished to an acceptable interval.

In particular, the major contributions of the work in this paper can be summarized as follows:

- The introduction of an efficient software remote execution architecture combining *CloudAP* with Cloud center.

978-0-7695-5088-6/13 $31.00 © 2013 IEEE
DOI 10.1109/HPCC.and.EUC.2013.195

- The design and implementation of an optimized whole-system VM migration approach which fulfills rapid migration and effective reconstruction of user's software execution environment.

The remaining sections are structured as follows: Section 2 presents a use case of *CloudAP* based execution environment; Section 3 presents an overview of the architecture proposed; Section 4 describes the performed approaches and implementations; Section 5 shows the experimental results; Section 6 discusses related work; finally, Section 7 presents the conclusions and discusses future work.

II. CLOUDAP-BASED EXECUTION ENVIRONMENT

In this section, we present a use case to illustrate the idea of *CloudAP* more concretely. The selected use case is based on *Software as a Service* (SaaS) commodity model [1] which is a typical framework for remote software execution.

SaaS provides an efficient means for users to execute applications which are beyond the computation ability of their thin mobile devices. They are executed on remote cloud servers while the results are transmitted and displayed on the client by using VNC viewer, XenDesktop, THINC[4], Muse[5] etc. It is extremely beneficial for users to even use a Windows or iOS application on their Android-based devices. For example, the user can continue editing the unfinished document using his tablet PC. However, due to the long distance between users and the cloud data center, a number of challenges still need to be addressed. The most obvious obstacle is the interaction latency in the WAN environment. This latency has a negative impact on the QoS of applications and the thin client in particular is vulnerable to this performance degradation. To bridge the gap between the QoS requirement and existing network latency, *CloudAP* is introduced.

Inspired by the pervasiveness and ease-to-use of base station in communication realm, *CloudAP* can be regarded as a small computing station providing both the software execution and transparent access for nearby users. It can be deployed in public places such as café, classrooms, meeting rooms etc.

As shown in Figure 1, when user A enters into an area covered by the certain *CloudAP*, he can directly run applications in the local *CloudAP* resource pools with a better QoS guarantee in comparison with those running in a remote Cloud center. In addition, users such as User B in a poor network environment can easily move into a nearby *CloudAP* covered area resulting in better service quality. More than that, the execution environment for User B can be maintained constantly via VM live migration between *CloudAP* and the Cloud center dynamically. Even when User B leaves from *CloudAP* area, his execution environment will be transferred back to the Cloud center for further utilization. In this way, the unfinished workloads can be transparently migrated in the background autonomously and then continue executing after migration. This seamless execution environment switch-over and conservation cannot be fulfilled in previous *Cloudlet* architecture.

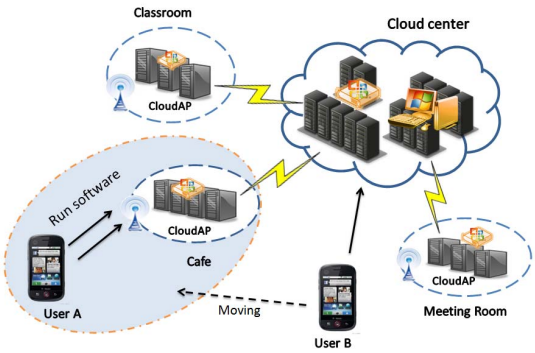

Figure 1. CloudAP Usercase Scenario

To achieve the above scenarios, a two-tier architecture is proposed considering both *CloudAP* and remote public Cloud center. In addition, an efficient VM migration approach is also introduced to support the rapid and uninterrupted switch-over of the software execution environment.

III. SYSTEM ARCHITECTURE OVERVIEW

To address the issue of interactive latency of remote public cloud, *CloudAP* which is close to mobile devices is proposed. It is a small-scale computing infrastructure which can provide software service to the nearby mobile users by a low-latency, one-hop and high-bandwidth wireless network. Moreover, *CloudAPs* are not isolated but constitute an integrated two-tier architecture with the Cloud center.

Two-tier architecture is essential in many aspects. First, users can acquire a better software service near a *CloudAP* and can still concede to the service provided by remote Cloud center even if no *CloudAP* is available nearby. In addition, if a mobile device which runs software switchovers between the *CloudAP* and Cloud center, the execution environment can migrate following user's location seamlessly.

The two-tier architecture proposed is shown in Figure 2. The system is comprised of two main components from each tier: Cloud center and a group of *CloudAPs*. The configuration comparisons can be observed in Table I. Each *CloudAP* is consist of several servers and the amount of devices which can access to one single server depends on the configurations of those servers. Basically, a common server can support around ten VMs simultaneously and each user's execution environment is capsulated into an isolated VM. Additionally, Cloud center can be connected by scores of *CloudAPs* in our scenario.

To support the federation and cooperation between tiers in this architecture, *CloudAP* and Cloud center are connected by high-bandwidth network and the message transmission is

TABLE I. CLOUDAP VS CLOUD CENTER

Options	CloudAP	Cloud Center
Network	LAN	WAN
Sharing	Few users	Large numbers of users
Software Execution Ability	Yes	Yes
Latency	Low	High
Bandwidth	High	Low
Scale	Small	Large

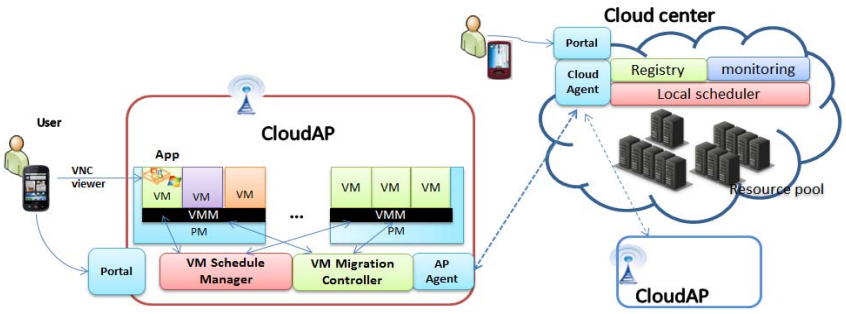

Figure 2. System Architecture Overview

through agent mechanism. That is, the agent is the representative of each tier and it communicates with each other via protocols or interfaces. As *CloudAPs* spread sparsely, the interactions among them are not considered in this paper.

In Cloud center, users can use the software service from the *Portal*. *Registry* is employed to registry *CloudAP* agent and to maintain the status of each running *CloudAP* agents.

The *CloudAP* infrastructure consists of four main components: *Portal*, which is the interface provided for users to customize the software services; *VM schedule manager*, which invokes specified scheduling strategy when VM placement occurs; *VM Migration Controller*, which controls the actions to migrate selected VM and execution environment between *CloudAP* and Cloud center; *AP Agent*, which communicates with agent in Cloud center and instructs other components to fulfill their functionalities.

Portal provides the web interface, which can be visited by users through LAN network. All available softwares provided by the service provider are visible on the Portal, and users can rapidly instantiate customized service software in the *CloudAP* from the portal.

AP Agent manages the entire communication and protocols with the Cloud center. The geographical location of mobile device near a *CloudAP* can be captured by the *AP agent*. Once there is a message indicating a certain device's location switchover between the *CloudAP* and Cloud center, the agent will parse the message and send operation instructions to *VM migration controller*, migrating the corresponding execution environment. In addition, AP agent negotiates with the Cloud center exchanging information such as status of user's execution environment, resource utilization of servers etc.

VM schedule Manager is the resource scheduler of this small-scale computing infrastructure. Load-balancing and resource utilization efficiency can be achieved by the VM schedule Manager.

VM migration controller is a core component which performs actual actions while a customer changes his geographical location and VM migration is carried on subsequently. Once an execution environment migration is triggered by the *AP agent*, the VM packaging the entire execution environment should be migrated between the *CloudAP* and Cloud center. This effective migration process should be supervised by the controller until the holistic execution environment is duplicated into local servers.

Since previously well-studied solutions to VM consolidation problems [7-9] and our previous work [6, 19] could be adopted in the VM scheduling and software remote execution, we assume that certain strategies have been deployed in these components. In this paper we will therefore mainly focus on the enhancement mechanisms in software execution environment migration to address the challenges we are facing.

IV. Design and Implementation

In the scenario, the software execution environment is entirely capsulated within a VM. For this reason, if we migrate the VM between Cloud center and *CloudAP*, the execution environment will be migrated subsequently. Because the Cloud center and *CloudAP* are in fact two isolated datacenters, there is no shared disk storage existed between them. Consequently, an efficient whole-system VM live migration mechanism should be designed and implemented in order to maintain the user's holistic execution environment.

A. Design

Whole-system migration enables the migration of the whole-system state of a VM including its CPU state, memory, disk storage data as well as the network state from the source to the destination machine. The migration of network state can be handled by VPN [20] or IP tunneling [21] techniques and the majority of previous literatures focus on the mechanisms to migrate memory and local disk states.

Luo et al. [11] propose a three-phase migration (TPM) mechanism to migrate the VM in three stages: pre-copy, freeze-and-copy and post-copy. In the pre-copy phase, both the memory and disk storage data are iteratively transferred to the destination. In the post-copy phase, the dirty block is synchronized according to the block-bitmap which is transfer in the free-and-copy phase. The approach proposed by Bradford et al. [13] is similar but lacks in the disk post-copy time. In this approach, the synchronization of dirty disk is parallel to the memory migration and all the read operations of VM in the destination are blocked until the end of dirty disk synchronization.

All these approaches don't work well in the *CloudAP* scenario because of the relatively long VM switch time (the

duration from when the migration initiates to when the VM runs in the destination). A minimal VM switch time indicates that the user can promptly switch the execution environment into a better *CloudAP* environment where lower latency and higher bandwidth will facilitate the promotion of QoS. During the memory pre-copy phase, the dirty memory is transferred iteratively until the number of dirty memory is within a low threshold. The multiple iterations of dirty memory pre-copy may take long time for the write-intensive application. With respect to disk, the huge amount of disk data has to be transmitted which results in long-term pre-copy.

To shorten the VM switch time during migration, we present an innovative method in which the disk is post-copied and the memory is migrated through a hybrid pre/post copy approach. Furthermore, we adopt the COW (copy-on-write) file system to set up the VM disk image. COW file system comprises base image and cow image. The base image is the VM template which is able to run standard OS and always remains read only. Any data added or changes made by the running VM will be written to the cow file. The base image which is always large can be stored in both *CloudAP* and Cloud Center well-planned in advance. Therefore, we just need to post-copy the image file to the destination.

Due to the high frequency of access to memory pages, pure post-copy will lead to frequent memory page fault and the degradation of VM performance. A hybrid memory migration approach including both pre-copy and post-copy is introduced to mitigate the performance decline. Specifically, memory is pre-copied in a single round and the remaining dirty memory pages are post-copied after the VM running in the destination.

As demonstrated in Figure 3, our system operates in three stages: memory pre-copy, freeze-and-copy and post-copy. The whole memory of VM is copied to the destination in pre-copy way. The VM service is unavailable only in freeze-and-copy phase. Whole disk storage data and dirty memory pages are synchronized in the post-copy stage.

In the first phase, whole memory pages are transferred to the destination with a single round. As the VM is still running, the memory will be modified by the running VM. A dirty-bitmap is used to record these dirty pages. In the following freeze-and-copy phase, the migrated VM is suspended on the source machine and CPU states are transferred to the destination. Additionally, the dirty-bitmap need to be transferred for the synchronization of inconsistent memory states in the next phase. Afterwards, the migrated VM is resumed on the destination in the post-copy phase. Due to the inconsistency of memory states, a dirty page pulling request occurs once a dirty page in the dirty-bitmap is accessed to. Meanwhile, the source pushes dirty memory pages to the destination according to the dirty-bitmap. Similarly, as the whole cow disk image data is still in the source, all the sectors in the image file should be set dirty in a block-bitmap. The source pushes dirty blocks to the destination in accordance with the block-bitmap while the destination uses the same block-bitmap to pull the dirty blocks requested by the migrated VM.

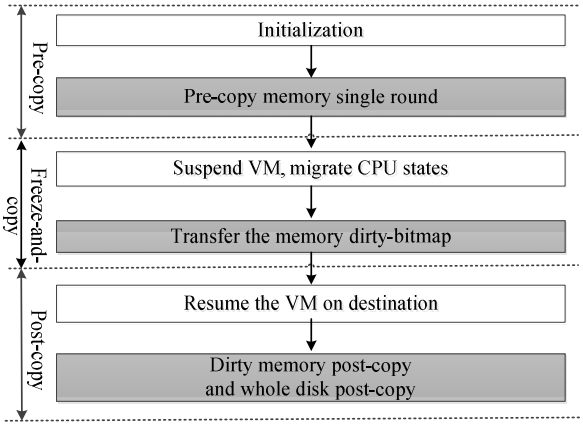

Figure 3. Overview of the migration process

B. *Implementation of whole-system migration*

In this subsection, we describe how the migration of whole-system VM states can be realized and the detailed implementations are presented below. The hybrid memory migration has been implemented on the QEMU-KVM (version kvm-84) in our previous work [22], thus we will mainly introduce the disk post-copy mechanism in this paper.

The simulation of disk I/O is achieved in KVM by its QEMU module. Disk I/O operations of guest VM are eventually transformed to file operations on the disk image files. The work of mapping the VM sector to the file offset of image files is achieved by the virtual block driver. Firstly, the *sector_num* of guest VM disk is converted into the *virtual sector_num* of disk images. Then the *virtual sector_num* of disk image is mapped to the file offset of corresponding image file.

As mentioned in the last subsection, a COW file system is used to set up the VM disk image and only the cow disk image need to be post-copied. At the beginning of post-copy stage, all virtual sectors of cow disk image should be marked dirty in the block-bitmap. Subsequently, the dirty sectors are post-copied by using both PULL and PUSH approaches until all disk states are completely synchronized.

The dirty sector is pulled only when it is being operated by the VM in the destination. The entire I/O requests are intercepted during the post-copy phase. If an I/O request is

```
Define：An I/O request R<FD, OP, BLK_NUM,>, where FD is
the image file, BASE or COW, OP is the operation, WRITE or
READ, BLK_NUM is the operated sector number
1.  An I/O request R<OP, BLK_NUM> is intercepted.
2.  if FD == BASE or bitmap[BLK_NUM] == 0 then
3.      goto 9
4.  else if R.OP == WRITE then
5.      send the syn request for bitmap in source
6.  else if R.OP == READ then
7.      send the pull request and syn the sector
8.  set bitmap[BLK_NUM] = 0
9.  submit or handle the request.
```

Figure 4. Pseudo-code for handling I/O request at destination

received, the destination performs as follows (shown in Figure 4): If the request is operated to the base image rather than the cow image or the operated sector is not a dirty sector, it will be submitted or handled directly. Otherwise, if it is a write request (line 4-5), the source block-bitmap should be synchronized simultaneously. If the request is a read (line 6-7), a pull request for the dirty sector is sent to the source machine. The corresponding bitmap will be updated after the sector is synchronized (line 8).

To shorten the disk post-copy time, the source end should not only receive and handle pull requests but also push the dirty sectors to destination actively. The source pushes the dirty sectors according to block-bitmap which is continuously synchronized with the destination. Furthermore, the read operations to the dirty sectors must be blocked before these sectors are pulled from the source, which will bring about overheads of I/O performance. If most dirty sectors can be pushed in advance rather than pulled when they are being operated, the overheads can be reduced significantly. Fortunately, analysis of disk I/O characteristics [12, 14] can contribute to predicting the disk I/O behavior and push the suitable dirty sectors to the destination intelligently.

It can be drawn from the analysis of disk I/O characteristics that read operations have a strong locality and continuity. If a read operation occurs to one dirty sector, the backward dirty sectors adjacent to this sector have a high probability to be accessed to. Therefore, once a pull request for one dirty sector is received in the source, we can push the backward neighboring dirty sectors to the destination. Figure 5 lists the pseudo-code for the two components of the source's performance – active push (line 1-9) and pull service (line 1-7) – both of which operate as two concurrent threads. The active push component starts from a pivot sector and pushes backward dirty sectors to the target in each iteration. Once a pull request is received, the pull service component shifts the pivot to the location of the pulled sector and starts to push from that location.

```
Define: The pivot and current is the location of the last pulled
sector and being pushed sector respectively.
1.  ActivePush:
2.      current = pivot = 0;
3.      while there exists dirty sectors do
4.          if the pivot changes then
5.              set current= pivot
6.          if bitmap[current] == 1 then
7.              set bitmap[current] = 0
8.              push this sector data to the destination
9.          current++
Define：An request R<type,sector>, where type shows the pull
for sector or syn for bitmap, the sector is the sector number.
1.  PullService:
2.      A pull/syn request R<type,sector> is received
3.      if bitmap[R.sector] == 1 then
4.          set bitmap[R.sector] = 0
5.      if R.type == pull then
6.          transmit sector data immediately
7.      set pivot = R.sector
```

Figure 5. Pseudo-code for disk post-copy algorithm at source

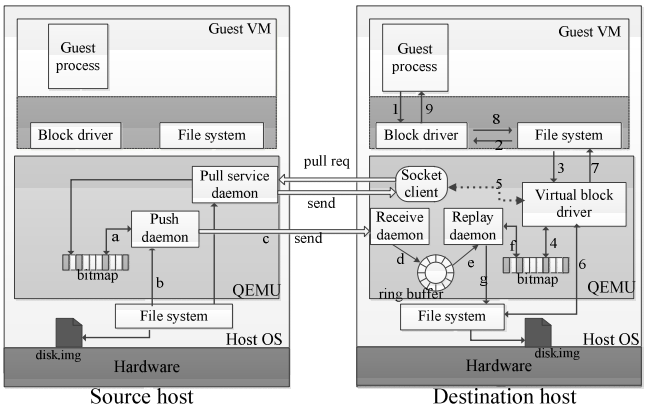

Figure 6. The implementation of disk post-copy

The implementation of system is illustrated in Figure 6. The I/O request of guest process is passed through the file system layer, block layer and virtual block layer in the QEMU (operations 1-3). The virtual block driver firstly checks the bitmap and sends a pull request for dirty sectors which will be received and handled by the pull service daemon in the source (operations 4-6). Thereafter, it submits or handles the I/O to the guest VM. The active push is operated concurrently with the pull operation. The source then fetches the dirty sectors according to the bitmap and sends them to the destination (operations a-c). All these dirty sectors are received by the receive daemon at the destination. Eventually, the replay daemon replays the dirty sector data based on the bitmap (operations e-g).

Moreover, the push efficiency can be further enhanced by leveraging data compression approaches. If we consider the sector to be a continuous array composed of binary bits, our experimentations demonstrate that a long serial bit 0 or 1 is contained in most dirty sector arrays. Therefore, we can compress the dirty sector data before transferring to the destination and decompress it before replaying at the destination. We adopt the RLE (Run-Length Encoding) strategy encoding and compressing the sector data. In this way, a 4KB sector can be compressed to 54-4096 bytes substantially improving the transmission efficiency.

V. EXPERIMENTS AND EVALUATION

In this section, we will discuss and explain the feasibility of the proposed *CloudAP* based two-tier architecture and then assess the efficiency of execution environment switch-over introduced by the advocated approaches.

Before explaining the experiments, the following metrics are presented to measure the effectiveness of execution environment migration:

Downtime is the time interval during which software service is entirely unavailable. Because the software execution environment is entirely encapsulated in the VM, it is the interval from when the VM pauses on the source machine to when it resumes on the destination.

Switch time is the duration from when the migration is initiated to when the VM resumes and starts to run in the destination. When the switch time is over, the VM can execute

in destination although several operations for state synchronization are still needed.

Total migration time is the duration from when the migration starts to when the VM runs independently in the destination. A migration is finished when the states are fully synchronized on the destination and the VM does not rely on the source.

Amount of migrated data is the amount of data transmitted during the whole migration time. It contains the memory state, storage state, CPU state, etc.

A. Experimental Environment

We use two sets of machines to simulate the Cloud center and *CloudAP*. Cloud center is formed using 16 x dell servers with each machine consisting of Intel(R) Core(TM) i7 860@2.80GHz CPU, 4GB memory and 320GB disk. *CloudAP* consists of 3 machines with the same configurations. The QEMU-KVM version is kvm-84. The VM running the software is configured with 512MB of memory and 10GB disk and the format of VM disk is qcow2 while the disk image contains a 10GB base disk image and a 250MB cow image.

The network bandwidth between *CloudAP and* Cloud center is limited to 100Mbps. The mobile device can be connected to the *CloudAP* with Wi-Fi (54 Mbps) and Cloud center through simulated 3G network (average 5Mbps bandwidth and 35 ms latency) respectively. Both the Cloud center and *CloudAP* is capable of running the following software: Abiword, OpenOffice and Firefox. An Android 4.0 tablet device with Exynos 4210 @1.4GHZ is used to execute the software through VNC.

B. Experimental Results

1) **QoS improvement in two-tier architecture**

To manifest the feasibility of the proposed two-tier remote software execution architecture, we emulate the typical behavior of a user who tries to use Abiword, a popular text editing remote executed software on his own mobile phone. We execute operations such as page scrolling, text input and so forth connecting to Cloud center and *CloudAP* respectively and record each interactive response delay. The delay time is calculated by monitoring and capturing network packets of VNC. As illustrated in Table 2, the average delay time surpasses 0.5 seconds and the value for some operations such as page scrolling and text dragging is even greater than second, far beyond the expectation of users to some extent. By contrast, the delay for most operations in the case of *CloudAP* is rather lower, which only takes less than 0.1s. Obviously, it is acceptable for users in this order of magnitude.

TABLE II USER EXPERIENCE COMPARISON BETWEEN CLOUDAP AND CLOUD CENTER

Operations	Cloud center(s)	CloudAP(s)
Page scrolling	1.08	0.54
Text input	0.32	<0.10
Text selection	0.51	<0.10
Color change	0.53	<0.10
Table insertion	0.52	<0.10
Text dragging	1.04	0.34
Text centering	0.32	<0.10

2) **Efficient execution environment migration**

One achievement of our architecture is that the execution environment can be migrated according to the change in user's location. The execution environment migration is implemented by the VM migration which may cause a temporal downtime and a moment of turbulent service. An efficient execution environment migration should contain two aspects: 1) short software service downtime during which the software service is unavailable; 2) short VM switch time during which the software service is available but inefficient.

We implement the execution environment migration experiment between Cloud center and *CloudAP*. The service downtime and VM switch time are illustrated in Table 3. It is obvious that the service downtime can be limited in 10 milliseconds, which can be nearly neglected. The total VM switch time is limited within 30s which is acceptable for user to sacrifice this time for obtaining a better software service.

TABLE III DOWNTIME AND SWITCH TIME

Workload	Downtime(ms)	Switch time(s)
Abiword	9.5	24.16
Openoffice	10	25.93
Firefox	9.6	29.91

3) Migration performance comparison

To prove that our VM migration scheme is efficient, some comparisons with current migration schemes are made in our experiments. We first realize the TPM[11] and deltas-apply[13] scheme in the hypervisor KVM, and then contrast our proposed approach against them with respect to the downtime, total VM switch time, total migration time as well as the amount of data transferred.

In order to compare the migration downtime with the TPM and deltas-apply scheme, the VM with the same software execution environment is migrated between Cloud center and *CloudAP*. As can be observed from Figure 7, our scheme outperforms the other two. The downtime of all the software can be reduced significantly to only 10 milliseconds in comparison to approximately 2-3s in other schemes. The reason for this phenomenon is that only memory bitmap is transmitted in freeze-and-copy stage using our proposed hybrid memory migration scheme. In contrast, several dirty memory pages and disk states (or block-bitmap) have to be transferred during this stage in the other two schemes, giving rise to relatively long downtime.

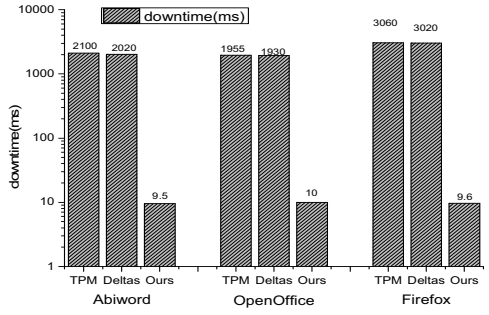

Figure 7. downtime of different schemes

Figure 8 presents the comparison of switch time among different migration strategies and the results indicate that our approach reaches an average 67.29% switch time reduction compared with other schemes. The improvement benefits from not only disk post-copy strategy but also the elimination of iterative dirty memory transfer time caused by pre-copy mode. It is no doubt that this improvement can significantly accelerate the switch over of user's execution environment thereby substantially promoting the user experience.

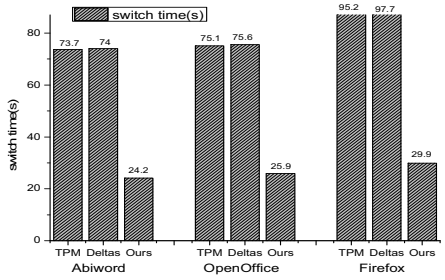

Figure 8. switch time of different schemes

Figure 9 shows the comparison in terms of total migration time among different schemes. With the migration procedure decomposed to three phases, the total migration time of the TPM scheme contains disk pre-copy time, memory migration time and post disk synchronization time. By contrast, the total migration time of deltas-apply is just the sum of disk pre-copy time and memory migration time, due to the fact that the disk synchronization is in parallel to memory migration. Unlike these two above, our approach experiences memory migration time and disk post-copy time without disk pre-copy procedure.

We can observe from Figure 9 that our scheme takes the lowest memory migration time among three different schemes. For example, in case of OpenOffice, it takes about 24.2s to migrate all memory pages, achieving approximately 52.4% and 52.6% reduction against TPM scheme (50.73s) and deltas-apply scheme (51s) respectively. This is because the proposed hybrid memory copy strategy shortens the convergence period of memory pages iterations by leveraging pull copy combined with memory push copy. Moreover, the large amount of disk pre-copy time (approximately 23s) can be totally omitted due to our disk post-copy strategy. Although our scheme takes a slight longer time (roughly 13.6s) during disk post synchronization than the other two schemes, the total migration time is greatly reduced by at most 58.1% in all circumstance.

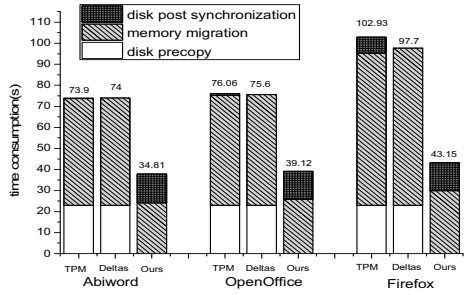

Figure 9. migration time of different schemes

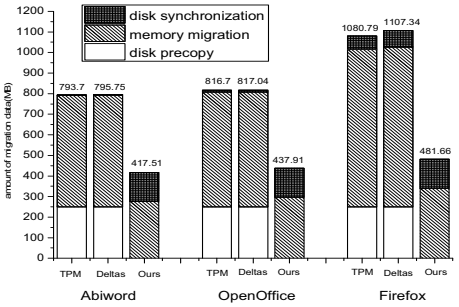

Figure 10. the amount of data transferred of different schemes

Figure 10 illustrates the amount of migration data in different schemes and it is obvious that the migration data can be greatly diminished thereby reducing migration time in our proposed scheme. The amount of memory data transferred in our scheme (276.8MB for Abiword) is much less than the other two (542MB and 544.2MB respectively) due to the fact that most multiple iterations of dirty memory pages are avoided in hybrid memory copy mode. Considering Firefox, the entire migration data is only 481.66M reaching roughly 55.4% and 56.5% decrease compared with TPM and delta-apply scheme respectively. Apparently, the disk data is copied at most one time during the disk post-copy phase. By contrast, all the disk data has to be transmitted during disk pre-copy stage and the dirty sectors should be transfer once again in post-copy stage. Additionally, there is no denying that the data compression approach also plays a significantly important role in decreasing the transmitted data.

VI. RELATED WORK

This section describes and discusses the most relevant related work towards handling problems in remote execution and whole-system migration of virtual machines.

Remote Execution of resource-intensive application has been previously analyzed. Mahadev et al. [2], present a concept called *Cloudlet* representing a trusted, resource-rich computer or cluster of computers that is well-connected to the Internet and is available for use by nearby mobile devices. The idea of *Cloudlet* is derived from the high latency which can exert negative impacts on interactive response time and QoS of mobile users. Verbelen et al. [3] extend the *Cloudlet* by providing a dynamic Ad hoc *Cloudlets* execution environment in which devices can join, share their resources and form a network *Cloudlet*. However, the current approaches only focus on a single *Cloudlet* or resource aggregation and cooperation mechanisms among different *Cloudlets* but neglect the collaboration mechanism and synthetic model between Cloud center and *Cloudlets*. Rahimi et al. [10] introduce a model to increase both performance and scalability of rich mobile applications leveraging a tiered cloud architecture. The authors in this paper focus on solving a resource allocation problem considering multiple QoS factors. Nevertheless, how to migrate the service efficiently is not discussed. In contrast, this paper presents a combined architecture including both Cloud center and *CloudAP* to overcome the performance degradation caused by WAN high latency. We also propose an optimized VM migration approach to support the conservation of holistic execution environment between Cloud center and *CloudAPs*.

Whole-System migration has also been discussed widely within the research community. A simple way to migrate a VM is freeze-and-copy, which first freezes the VM to copy its whole-system state to the destination, and then restarts the VM at the destination. Internet Suspend/Resume [15-18] is a mature project using freeze-and-copy to capture and transfer a whole VM system. It results in a severe downtime due to the large size of memory and storage state. To replicate the bulk data in disk storage with less disruption across WANs, Bradford et al. [13] proposed a block level solution combining pre-copying with write throttling to concurrently apply the write access to both local and remote disk during the migration. However, it may cause long I/O block time and redundant data transfer caused by overwrites. Luo et al. [11] proposed a three-phase migration (TPM) algorithm to reduce the downtime caused by bulk data of disk migration. They suggested using a block-bitmap to track all write access to the local disk during the disk migration. Then, the dirty data are re-sent to the target host according to the block-bitmap in the last phase of migration. The three-phase may increase total migration time and bring about I/O overhead. The largest drawback of above two mechanisms is the long switch time. The post-copy [23] strategy can significantly reduce the VM switch time but may incur the severe performance degradation during migration. Experiments illustrate that our approach achieves a better performance compared with them above.

VII. Conclusions and Future Work

In this paper we introduce a two-tier architecture comprised of Cloud center and *CloudAP* and show how they can be cooperated to form a comprehensive mobile computing environment where the QoS of the mobile applications is greatly enhanced. Additionally, we present an effective execution environment migration scheme implemented by an efficient whole-system VM migration to make the execution environment move according to the mobile device's location shift. Specifically, a set of optimizations are proposed to minimize the costs such as down-time span, switch time, total migration time and data transferred etc. during the execution environment migration. Our experimental results show that our proposed scheme can decrease the downtime and execution environment switch time to up to 10 milliseconds and 30 seconds respectively, improving the user experience and QoS dramatically in mobile environments.

As future work, we plan to develop a more comprehensive resource scheduling mechanism between *CloudAP* and Cloud center. In addition, many users accessing to the *CloudAP* simultaneously may cause resource overhead, degrading the service quality. In order to form a more integrated architecture, the software execution environment should not only move following the location of mobile device but also perceive current resource loads in *CloudAPs*.

Acknowledgments

The work in this paper has been supported in part by the National Basic Research Program of China (973) (No. 2011CB302602), China 863 program (No. 2011AA01A202), and National Nature Science Foundation of China (No. 61170294, 91118008, 61272165).

References

[1] B. Gain. (2010, January 1) Cloud Computing & SaaS In 2010 Processor Mag. 12.

[2] Satyanarayanan, Mahadev, et al. "The case for vm-based cloudlets in mobile computing." Pervasive Computing, IEEE 8.4 (2009): 14

[3] Verbelen, Tim, et al. "Cloudlets: Bringing the cloud to the mobile user." Proceedings of the third ACM workshop on Mobile cloud computing and services. ACM, 2012.

[4] Baratto, etc al. "THINC: a virtual display architecture for thin-client computing." ACM SIGOPS Operating Systems Review. Vol. 39. No. 5. ACM, 2005.

[5] Yu, Weiren, et al. "Muse: a multimedia streaming enabled remote interactivity system for mobile devices." Proceedings of the 10th International Conference on Mobile and Ubiquitous Multimedia. ACM, 2011.

[6] I. Solis Moreno, et al. "Improved Energy-Efficiency in Cloud Datacenters with Interference-Aware Virtual Machine Placement," in Proccedings of The IEEE 11th International Symposium on Autonomous Decentralized Systems ISADS, Mexico City, 2013.

[7] Beloglazov, et al.. "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing." Future Generation Computer Systems 28.5 (2012): 755-768.

[8] Hermenier, Fabien, et al. "Entropy: a consolidation manager for clusters." in Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments. ACM, 2009.

[9] Khanna, Gunjan, et al. "Application performance management in virtualized server environments." Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP. IEEE, 2006.

[10] Rahimi, M. Reza, et al. "MAPCloud: mobile applications on an elastic and scalable 2-tier cloud architecture." in Proceedings of the 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing. IEEE Computer Society, 2012.

[11] Luo, Yingwei, et al. "Live and incremental whole-system migration of virtual machines using block-bitmap." 2008 IEEE International Conference on Cluster Computing(Cluster). IEEE, 2008.

[12] Akoush, Sherif, et al. "Activity Based Sector Synchronisation: Efficient Transfer of Disk-State for WAN Live Migration." 2011 IEEE 19th International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS). IEEE, 2011

[13] Bradford, Robert, et al. "Live wide-area migration of virtual machines including local persistent state." in Proceedings of the 3rd international conference on Virtual execution environments(VEE). ACM, 2007.

[14] Shen YL, Xu L. "Efficient disk I/O characteristics analysis method based on virtual machine technology." Journal of Software, 2010,21(4):849-862. http://www.jos.org.cn/1000-9825/3492.htm

[15] Kozuch, M, et al. Internet Suspend/Resume. Fourth IEEE Workshop on Mobile Computing Systems and Applications, 2002.

[16] C. P. Sapuntzakis, et al. "Optimizing the Migration of Virtual Computers. OSDI", 2002.

[17] M. Kozuch, M. Satyanarayanan, T. Bressoud, C. Helfrich, S. Sinnamohideen. Seamless Mobile Computing on Fixed Infrastructure. Computer, July 2004.

[18] R Chandra, N Zeldovich, C Sapuntzakis, MS Lam. The Collective: A Cache-Based System Management Architecture. NSDI '05: 2nd Symposium on Networked Systems Design & Implementation, 2005.

[19] Zhong, Liang etc. al. "A Virtualization-based SaaS Enabling Architecture for Cloud Computing." In Autonomic and Autonomous Systems (ICAS), 2010 Sixth International Conference on, pp. 144-149. IEEE, 2010.

[20] T. Wood , et al. CloudNet: dynamic pooling of cloud resources by live WAN migration of virtual machines. in Proceedings of the 2011 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments(VEE).ACM, 2011

[21] Travostino F et al. Seamless live migration of virtual machines over the MAN/WAN[J]. Future Generation Computer Systems, 2006, 22(8): 901-907

[22] Chen,Yang etc. al. " Live Migration of Virtual Machines Based on Hybrid Memory Copy Approach." Chinese Jouinal of Computers 34.12(2011):2278-2291. http://cjc.ict.ac.cn/qwjs/view.asp?id=3515

[23] Hines etc.al "Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning."in Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments(VEE). ACM, 2009.