Available online www.jsaer.com

***Journal of Scientific and Engineering Research*, 2018, 5(1):254-265**

Research Article

ISSN: 2394-2630
CODEN(USA): JSERBR

SyncOS: Cloud-enabled Mobile VM State Manager

Chad DeLoatch[1], Roopak Ingole[2], Aaron Hogue[3]

Manager – Advanced Embedded Software *Corporate Research & Technology Cummins Inc.* Columbus IN, USA
Email: [1]deloatc1@illinois.edu, [2]roopak.ingole@gmail.com, [3]ahogue@computer.org

Abstract OEMs have begun manufacturing multiple portable de- vices for different consumer usages (smartphones, e-Book readers, web browsers), all while running the same OS. Motorola Droid and Xoom run the same Android OS for different purposes, but share the same user interface, ap- plications, and data to some extent. Same is the case with Apple iPhone and iPad (iOS), and the Samsung Galaxy Mobile and Tablet. Each device maintains its own user interface, application collections, and data. It would be beneficial if a complete OS could be shared or synchronized on these devices for a multitude of reasons in which we will discuss in this paper.

We demonstrate a secure mobile VM Suspend & Resume functionality by building *SyncOS*, a kernel using OKL4 as a basic building block for storing our VM state in the cloud. OKL4 is a mobile VMM (hypervisor) pro- viding virtualization for mobile devices. The OKL4 architecture allows for running multiple guest OSes on the mobile devices whilst providing security and isolation in a cross-platform manner. VM Suspend & Resume is used heavily in desktop/server hypervisors and extensively in cloud computing environments. Its usefulness arises for such purposes as OS backups, live VM migration, sandbox testing, automatic failovers, standardized enterprise OS deployments, and secure OS delivery for the private sector.

Keywords Cloud-enabled, VM State Manager

1. Introduction

Mobile operating systems should offer more portability as manufacturers are continually refreshing the hardware chipsets and form factors. Without the ability to keep our OSes synchronized on different devices, we must install the same native applications on each device OS we own. We are proposing the process could be make simpler by providing a means of state management using a snapshotted VM from another mobile device running SyncOS. SyncOS relies on geographically distributed cloud data centers for remote checkpoint image storage to avoid docking your device or using native storage for state transfers. SyncOS will allow you to resume your smartphone environment directly on your tablet, slate, or larger form factor devices, or possibly even to an upgraded smartphone. Al- though in some cases, this state management could be useful for a multitude of other benefits such as sharing your customized environment with a friend or colleague.

Existing VMM products do not address secure check- pointing across mobile device platforms without using native storage (SD Card, USB Drive) and physically wired network connections (USB, Ethernet) for state transfer. Commercial VMMs, such as Hyper-V, XenServer, KVM, vSphere, and Virtual Box rely on State Managers for their guest OSs. Current mobile operating systems only sup- port state management of applications (*tombstoning*) and not checkpointing the entire OS. This is primarily driven from the limitation of local storage on the mobile devices and potential complexities during system resume. Storing VMs to the cloud instead a local SAN or NAS has also been studied, but only as it relates to servers [18, 41]. Using thin clients, such as VNC to access remote machine environments suffers from network latency and jitter [39]. Thin clients also don't provide the same rich interaction a na- tive VM could provide. OpenISR supports VM migration to a

remote server, but not do a geographically distributed cloud data center [37]. ISR also only supports Linux and other monolithic kernels - not the security-minded OKL4.

Mobile devices are resource constrained by network bandwidth and local storage size. Sharing mobile OS state with public storage places a risk for data interception with- out a secure transfer process. Current mobile devices, such as iOS will not enable snapshots of their state without docking directly to a workstation. With the often small device storage support for smartphones, there generally isn't anywhere locally to store the large image a mobile device could potentially consume. Limited network band- width also presents challenges if you want to store guest OS state remotely with smartphone devices reaching 16GB and tablets storage now growing as high as 64GB. Mobile OSs don't currently have an easy way to persist state or snapshot their current workload, as well as resume from a past checkpoint.

SyncOS supports securely addressable VM images us- ing cloud storage (with shared access signatures) and a se- cure microkernel-based guest OS checkpointing applica- tion. Mobile VM management is important for purposes of a remote backup facility, VM image sharing, standard- ized enterprise deployments, secure government VMs [12], synchronized OS across platforms (Android, iOS), as well

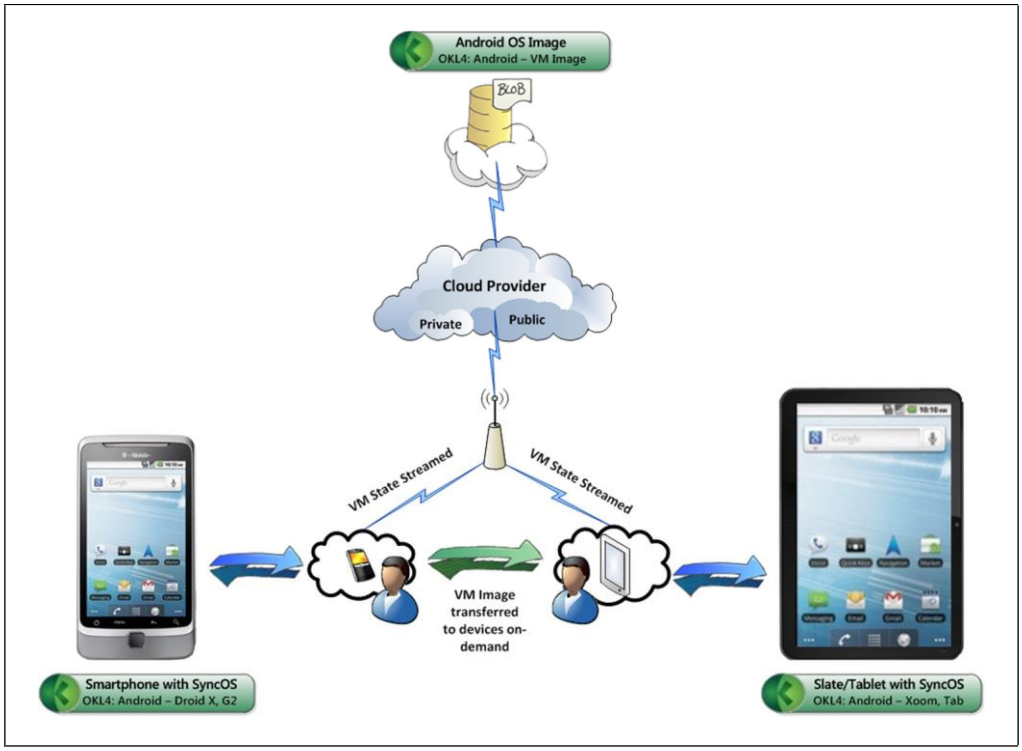

Figure 1: SyncOS - Environment Overview. Migration between devices using wireless communication.

as same platform synchronization (Droid vs. Xoom). Our implementation approach uses OKL4 microvisor to host the guest OS (OK:Android) and SyncOS VMs. We plan to snapshot the guest VM image, and persist the captured snapshot remotely to cloud storage where it can be accessi- ble via a secured URI. At some point later, we can use the same URI to download and resume the checkpointed VM image on another device running SyncOS.

We anticipate that we can capture the state of multiple VM guest OSes running SyncOS such as OKL4:Android and OKL4:Linux. We initially plan to capture state locally and resume from the OKL4 desktop emulator (QEMU). As time permits, we plan to extend this local VM storage model into the cloud by using existing cloud REST APIs for storage and retrieval. Once implemented, we would like to demo SyncOS on real Android-based devices.

The rest of this paper is organized as follows. Section 2 discusses the challenges and design principles of SyncOS. Section 3 describes the SyncOS system design. Section 4 covers the implementation of the SyncOS system. Section 5 discusses the evaluation of SyncOS. Section 6 explains some of the related work done in this area. Section 7 de- scribes the conclusion and some of the future work.

2. Challenges & Design Principles

Our goals for SyncOS are to make migration easier across device boundaries. In this section we will cover some of the challenges in greater depth before diving into the de- tailed system design. Figure 1 above depicts the environ- ment capabilities we wish to achieve.

2.1 Challenges

Capturing state on mobile devices introduces new chal- lenges due to the constraints imposed on battery power, device size, and storage size. Most mobile devices today only run a single native OS. To support mobile OS snap- shots, we will need some way to manage the OS state. The common way for desktops and servers to achieve this is by using a hypervisor layer, or VMM, to manage the Guest OS state. Hypervisor layers will also enable concurrent OS execution - something we expect to be more common in the future.

With concurrent OS execution on the same platform (as VMs), we would like to guarantee secure data and process isolation between each OS instance. Our basic approach to this challenge is to use a microkernel hypervisor to guar- antee that no running thread can access another running threads data without explicit IPC commands. Microker- nels small kernel size also reduce the TCB, ensuring that security vulnerabilities are minimized.

After securing our running VMs above a microkernel, we would like to store the OS snapshot data somewhere it can be shared with another SyncOS-aware device. Since mobile device storage is typically limited, we need a cloud provider that we can persist our VM state to and then re- trieve it using a secure channel. There are no guarantees the device we are transferring state to are line-of-sight or within Bluetooth range to easily communicate with them. By using a remote cloud facility, we introduce network la- tency and increased power consumption, but provide data redundancy and secure storage and retrieval to offset the cost. Our cloud storage approach also avoids Bluetooth pairing, device docking, or infrared state transmissions.

Another important problem occurs if our state backups exist locally on the physical device. If the device becomes lost or stolen, or possibly ruined by hardware failures - there may be no way to retreive state from the local storage. Our cloud model also covers this gap by keeping a history of checkpoints from previous state captures (similar to Mi- crosoft Windows System Restore history) and stores them remotely. Using this approach covers us in the event of device failure or misplacement.

Once we have achieve state storage, we need to sup- port resuming VM state using another device. This can be achieved only if the receiving device knows where the state resides and also has a shared hypervisor core to understand how the VM can be natively resumed.

2.2 Design Principles

2.2.1 OKL4 Microvisor

Open Kernel Labs [35] has developed an open source mo- bile hypervisor called OKL4 Microvisor. OKL4 Microvi- sor is a microkernel [25] and hypervisor that provides a ro- bust, secure and scalable architecture called Secure Hyper- Cell technology shown in Figure 2. This technology is for hosting multiple unprivileged (user) mode isolated envi- ronments (cells) that contain applications, virtual machines and subsystems. OKL4 Microvisor also provides fast IPC communication between environments (cells). OKL4 Mi- crovisor hardware support includes popular processor ar- chitecture ARM, MIPS, and Intel. OKL4 Microvisor cur- rently does not provide a service for managing guest OS states.

2.2.2 Cloud Storage Providers

Cloud Storage Providers exist for both public and private clouds. An enterprise environment may find value in host- ing their own cloud depending on how their devices need to interact, but in most cases we see the economic value of public clouds to be more feasible. We only need an HTTP BLOB storage API to consume since we are strictly deal- ing with VM state data storage and retrieval, but ideally,

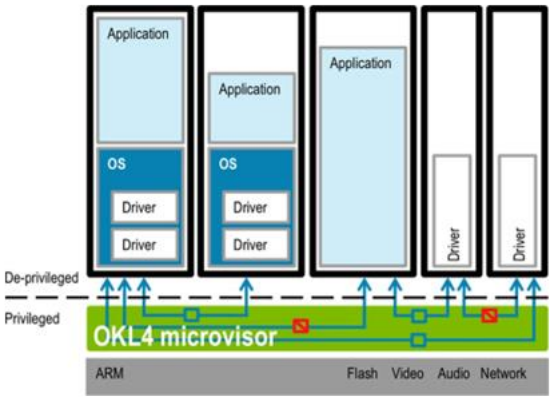

Figure 2: OKL4 Secure HyperCell Technology.

we would have a reusable service layer to handle this in- terfacing. Using Microsoft Azure, we can strictly rely on an HTTP REST-style interface without the need for a com- plex user-level libary embedded within the kernel.

3. SyncOS Design

Our design for SyncOS was to take the OKL4 Microvisor and add two secure cells. The first hypercell contains the guest OS OK:Android and the second contains SyncOS. The OKL4 microvisor already has support for display, stor- age, and networking drivers which we will leverage using synchronous IPC calls. Figure 3 depicts the structure de- scribed. The SyncOS cell is responsible for suspending and resuming the VM state as well as cloud data storage and retrieval.

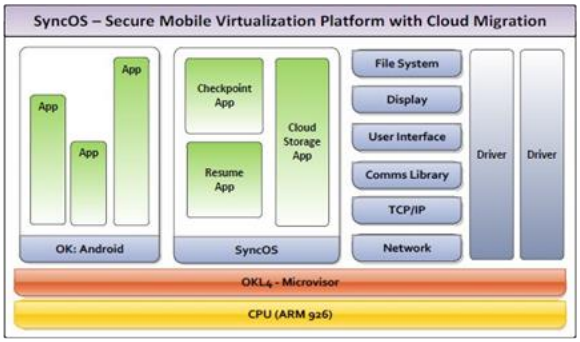

Figure 3: SyncOS - secure checkpoint and resume for portable devices using cloud data storage.

3.1 SyncOS Hypercell

SyncOS will contain two primary independent segments - the Cloud Storage application and Checkpoint/Resume ap plication. SyncOS is essentially a patched OK:Linux ker- nel. We prefered to use OK:Linux for simplicity, and even- tually it will be shrink to reduce the TCB for the unneces- sary components. While ckeckpointing, the cloud storage application will take the image from the kernel and upload it to the server with some security. During the resume, the storage application will download the image from the server and notify it to resume application. The Check- point/Resume application will trigger the checkpointing or restore operation of the guest os on the OKL4 kernel. The kernel is responsible for performing the actual checkpoint- ing and restore operation.

3.2 Guest OS Hypercell

OK:Android was our targeted guest OS used for check- pointing the system state. Our plan is to have a user man- aged application trigger the checkpointing or resume pro- cess. Once the checkpoint request has been initiated by a user, an IPC message will be sent to the SyncOS Check- point application. The Checkpoint application will respond with a confirmation and guest OS shutdown request. The OK:Android kernel state will be captured and simultane- ously trigger the Cloud Storage application for persisting the capture state. Our Cloud Storage application will write locally to disk during our initial development. Once the lo- cal state has

been capture, the cloud storage API will be in- serted and consume the network drivers and TCP/IP stack for managing the guest OS snapshot state remotely.

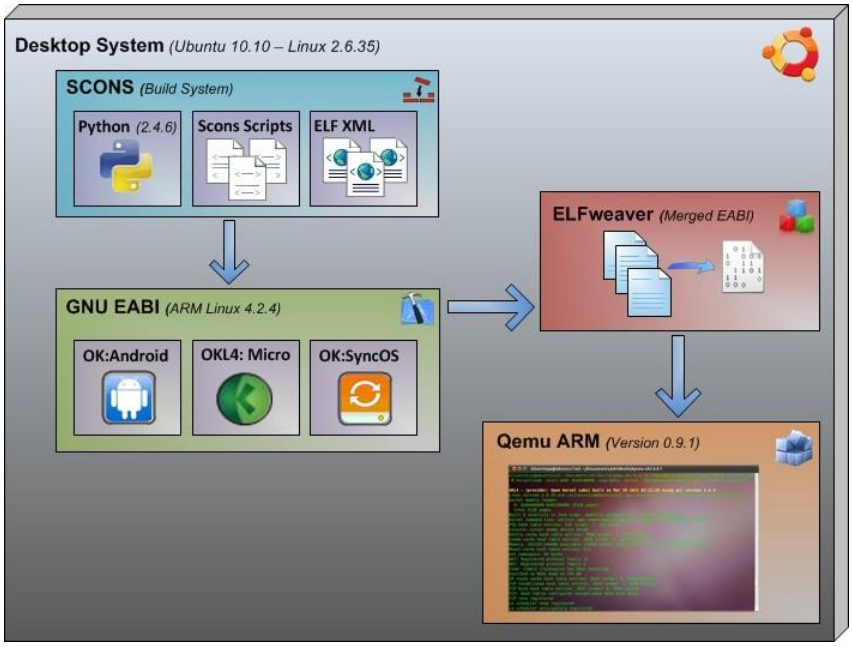

Figure 4: SyncOS - Development Environment. Control flow for building SyncOS ELF image to execution

3.3 OKL4 Kernel

The OKL4 Kernel will implement the two new system calls in order to accept the request for snapshot or restore from the SyncOS application. These two system calls will es- sentially trigger the checkpoint or restore algorithm. The algorithms will be implemented in the kernel space be- cause it is easier to access the guest OS information from kernel. Since the OKL4 doesn't support any direct commu- nication and data sharing between kernel and cell, the cap- tured image has to be stores on some physical media(e.g. Flash). In order to store the captured image, the kernel has to be modified to support the some sort of file system. The device driver for the flash media will be added inside the kernel to read and write the image data.

3.4 Cloud Provider Integration

Utilizing a public cloud provider does present some secu- rity concerns for user privacy. Since user data is being per- sisted remotely, users want to know how their data is being protected from prying eyes and that it cannot be compro- mised. Microsoft Azure provides a content delivery pro- cess that enables us to share blob data using URIs that are tied to SAS (shared access signatures) [32] which can only be used until a specified lease expires access to a given data stream. Essentially, this protection provides a way for users to share their state with friends indefinitely, never, or only for a short period of time (say 2 hours) to limit the op- portunity for attackers to compromise the URI in the event that it is discovered or intercepted.

A typical Azure blob endpoint URI will be formatted as *http://syncos.blob.core.windows.net/captures/oklinux.sync? st=2011-05-10&se=2011-05-11&sr=c&sp=r& si=YWJjZGVmZw&sig=dD80ih...Bh5jf*. In this ex- ample, the st (start time) and se (expiry time) defines how long the lease is valid, while the sig (signature) parameter defines a cryptographic hash of the querystring parameters using a private key. The signature prevents attackers from simply altering the URI lease to extend its validity.

Another concern for users is data interception during transmission. A potential solution is to encrypt the VM state data prior to data transfer and decrypting prior to restoring the VM state. This approach would require some sort of trusted certificate publisher, but could handle data while in transit. HTTP SSL could also provide another layer of protection while communicating with the Azure APIs.

3.5 Design Alternatives

We also looked at other implementation approaches that involve capturing process state from two devices running that same OKL4 merged ELF image. CRAK [8] is a ker- nel module for Linux 2.6 that supports suspend/resume of user-level processes. This project provided inspiration for our state capture and resume. However, it did not capture kernel state and thus did not provide enough capabilities for our needs.

4. Implementation

The OKL4 Microvisor provides the building block for our SyncOS solution. The prototype system will con- tain two paravirtualized OKL4 Guest OSes (OK:Linux and OK:Android).

4.1 OKL4 Environment Setup

The OKL4 development environment was intensive to setup. Some of the build requirements for various workarounds - such as specific GCC requirements (version 3.4) and Python which made installing it on the latest linux distros quite cumbersome. Here we detail the steps we took to create a mock environment for our SyncOS system as described in Figure 4.

We first downloaded the OKL4 source and attempted to build the OKL4 Microvisor for OK:Linux, as we had some issues with compiling OK:Android. We compiled the OKL4 3.0 kernel image and built it using OK-Labs custom GNU toolchain and EABI. Once the ELF image was built, we ran the OK kernel in the QEMU emulator on Ubuntu 10.04 to simulate ARM versatile platform. QEMU handles the cross-compilation of various architectures (x86 and ARM). For the OK:Linux compilation, a specific ker- nel patch had to be applied. The custom environment also required Python 2.4, path configuration changes, and mod- ification of OKL4 build python scripts for your local envi- ronment.

We ran into issues with Ubuntu 10.04 not displaying a login prompt after bootup (possibly a GDM linking issue). We did find a reboot workaround, but we suspect a con- flict with the Python or GNU toolchain/EABI dependen- cies. We had this same issue occur for both physical and virtual Ubuntu environments. The potential workarounds we discovered invovle enabling auto-login for the user ac- count and installing Python using the ”altinstall” option. The combination of these two items seemed to have en- abled a reboot to not permanently disable the demo envi- ronment.

4.2 OKL4 Kernel Modifications

In order to implement the checkpoint and restore algo- rithm, after the research on OKL4 kernel, it was easier to implement those in the OKL4 kernel itself. OKL4 ker- nel itself has the knowledge of all the cells it initialized, it knows the exact memory space allocated for each cell. It is easier to implement the core checkpointing and restore algorithm in kernel and provide the interfaces to the appli- cation to invoke the algorithm.

The application or cells in OKL4 can communicate with the kernel only via a system calls. Inorder to send the re- quest for checkpoint or restore from application we had to add two new system calls, Checkpoint Request and Re- store Request. Whenever SyncOS triggers the Checkpoint System Call, the OKL4 kernel will take a snapshot of the requested guest OS cell. Similarly, when SyncOS triggers the Restore System Call, the OKL4 will record the restore request and trigger the reboot.

The exact functioning of the Checkpoint and Restore operation is explained using the Message Sequence Chart shown in Figure 5 & 6.

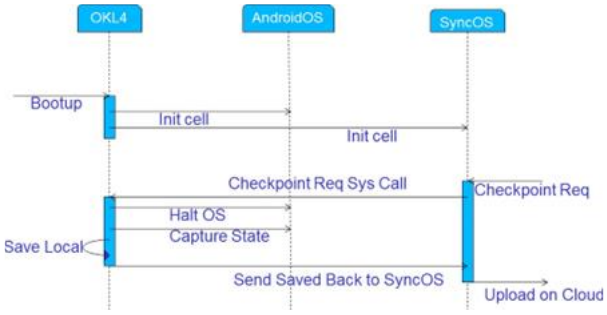

Figure 5: SyncOS - message sequencing for VM bootup and checkpoint operations.

Figure 5 depicts the initial boot process and checkpoint operation. When the device boots, OKL4 is initialized which creates all the cells which are linked at the build time. Along with those cells, our SyncOS cell will also

be initialized. When the user triggers the Checkpoint re- quest to SyncOS, the SyncOS will issue the Checkpoint Request System Call to OKL4 and OKL4 will snapshot the requested OS. Once the snapshot is done, it will then send the image back to SyncOS. Now the SyncOS will upload the image on the server.

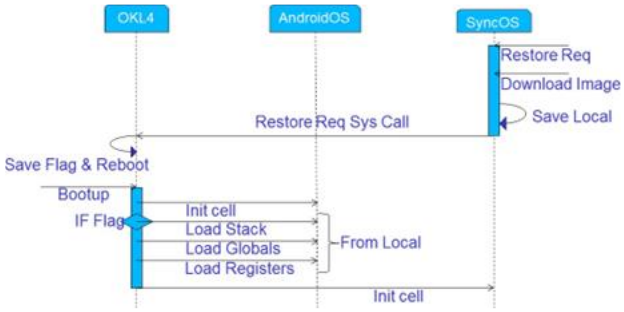

Figure 6: SyncOS - message sequencing for VM restore operation.

Figure 6 explains the restore operation. When the user triggers the Restore Request to SyncOS, the SyncOS will download the image from the server and will save it lo- cally to be shared with the kernel. Once the download is complete, it will issue the Restore Request System Call. The OKL4 kernel will record the restore request for the desired cell (guest OS) and reboot the device. At the time of initializing the cells, OKL4 will check for the restore request and if the restore is requested, it will initialized the stack, registers and globals based on the local image saved earlier. This way the saved image will be restored on the same device or other device.

4.3 Storing Guest OS States To The Cloud

Our implementation goal was modify the capture and re- store daemon processes to use cloud (virtual) storage [18] to access the Guest OS mementos/parcels (file images). We planned to use the Microsoft Azure Storage API web ser- vices to safely interact with user storage accounts.Due to time constraints we were not able to fully implement this portion of the project.

5. Experimental Evaluation

This section describes our evaluation of SyncOS. We dis- cuss our experimental results based upon system expec- tations and typical usage scenarios for SyncOS as dis- cussed in Section 4.1. We also look at the migration approach, storage mechanism, overall performance, and lessons learned.

5.1 Migration Time Estimates

Below are the graphs relating to the downtime of a device which is being captured and then restored to another de- vice using a cloud provider as an intermediary. Depend- ing upon device state size, which varies by user, we chart various state sizes for comparison of downtime. We used average cellular upload and download speeds available on the web [38]. Each chart depicts the assumptions relating to network speed.

5.1.1 Migration Time - Stop & Copy Suspend

The VM capture size varies based upon kernel state size and user data. Our demonstration environment averaged a capture size of 10MB for OK:Linux without any user apps (*using BusyBox utility*). The capture time is based on upload speed to cloud provider which is determined by wireless connection type (WiFi - 802.11 or Cellular 3G). As cellular speeds increase, the capture time should shrink proportionately. Since our base state size was 10MB, we plot increasing state sizes 5x, 10x, and 20x the base to present some likely scenarios in Figure 7 to demonstrate real-world usage.

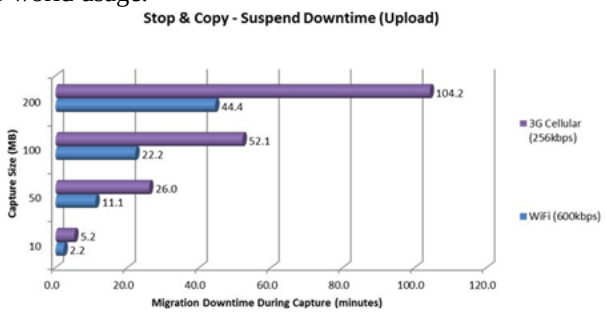

Figure 7: SyncOS - Capturing VM state, expected down- time for a stop-and-copy migration approach.

5.1.2 Migration Time - Stop & Copy Resume

The VM restore time is based on download speed to cloud provider and system reboot time which depends on device boot configuration and hardware. Using our emulator as a baseline since we haven't deployed to an actual device, we are assuming an average boot time of 10 seconds based upon the QEMU emulator environment we created in Sec- tion 4.1. In Figure 8, we share our approximate restoration time which should typically be faster given that download speeds often exceed upload speeds on the network. We also use the 5x, 10x, and 20x multipliers to show how state size affects the restore time of the VM state across device boundaries.

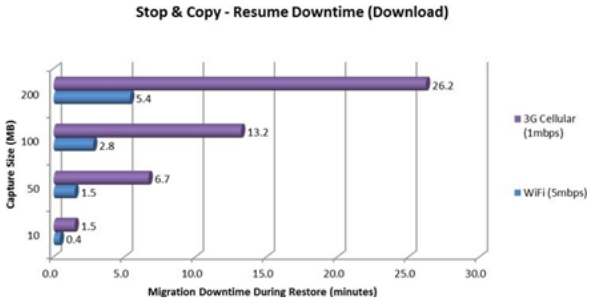

Figure 8: SyncOS - Restoring VM state, expected down- time for a stop-and-copy migration approach.

5.1.3 Migration Time - Full Stop & Copy

In Figure 9 we combine both VM Checkpoint & Restore costs to provide a central view of the downtime involved when a user triggers the device to device state transfer. De- pending on how SyncOS is utilized, the entire migration time may or may not be a factor. If the user only wishes to provide a fault-tolerant backup of their device state into the cloud, we don't incur the resume downtime overhead, only the suspend costs.

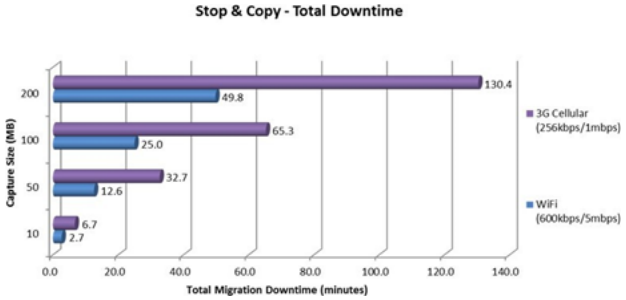

Figure 9: SyncOS - full VM state transfer, expected down- time for a stop-and-copy migration approach.

5.2 Migration Approach

Our SyncOS snapshot strategy of stop-and-copy provides a quick win as it relates to a proof of concept model. However, it would be nice to minimize any downtime in- curred by the user by performing live or pre-copy migra- tions which we discuss in Section 7.2. The SyncOS archi- tecture is at the mercy of wireless networks, but one of our goals is to make the migration process easy for the users who are on the go and don't want to have to dock their de- vice each time they want to switch contexts. We think our migration strategy will become better adopted as the net- work speeds continue to increase and bandwidth catches up with our on-demand device state sharing.

5.3 Storage Mechanism

Microsoft Azure allows us to store entire state image as a single blob (1TB max file size). Azure Blob Storage also provides redundancies by copying blob data across 3 nodes to embrace failed hardware or software issues. This strat- egy helps prevent corruption if we were only writing the state locally which could not provide this level of data pro- tection. We can also pick our data center region to be ge- ographically local to where we physically are to minimize network latency.

Current Azure APIs require cloud accounts, but using this model we have developed, our SyncOS platform could be sold as an add-on service by cellular providers or soft- ware OS vendors. The cloud provider could easily be swapped in or out to meet the needs of the public sector, private sector, or standard home user.

5.4 Overall Performance

An important questions we have to ask is whether we could potentially reach a usable system state that consumers would actually utilize - given some of the above charts depicting snapshot capture & restore time. We re- alize this research has opened up new ideas and we hope that others can use our findings to continue to open new avenues of research with mobile VMMs and device state sharing. Regardless of what's currently possible, we are looking to the future of mobile computing and what values we can bring without regard to the present infrastructure or hardware limits.

5.5 Lessons Learned

5.5.1 Ubuntu - Logon Reboot Issue (autologin, python alt-install option)

As described in the above section, our development and validation environment is based on Ubuntu v 10.1. In or- der to get the base environment establish we had to resolve the lot of dependencies about the packages on Ubuntu. Be- cause of lack of documentation, we learned and resolve these one by one. At Out of all the dependent packages, the certain version GCC and Python has to be installed. Once we install all the packages, we faced the login issue after reboot. This was caused by some Python conflict. We could overcome this problem by installing the version of Python as alternate package and making the Ubuntu as autologin.

5.5.2 Micro vs. Nano OKL4 Versions (multicell capabilities)

OKL4 can be built with the capabilities of Nano Kernel or Micro Kernel. The choice of kernel to be built is speci- fied at the build time. The Nano Kernel provides the sub- set of the features of Micro Kernel. Since our experiment required the Multicell capability of the kernel in order to support the guest OS and SyncOS at the same time, we learned that we needed the Micro kernel. It took us a while to figure out how to compile the whole kernel with micro kernel capabilities. Once we got the micro kernel compiled and linked, we could create the multicells which helped us to understand the internals of the OKL4

5.5.3 OK:Linux -IRQ issues, Build System Tweaks

Once we got the multicell working, the next thing was to compile the Linux on OKL4. The documentation was not sufficient to get the Linux successfully compiled and run with OKL4 on the ARM target. There were lot of build system tweaks we had to do in order to get Linux compile. Once we got the Linux running on OKL4, the next thing for us was to get two instances of guest OS running on OKL4. For this we choose to run two instances of Linux itself. The original intentions were to have Linux and An- droid OS running as guest OS on OKL4 and use Linux as our SyncOS. In the interest of time we decided to use two instances of Linux for the experiment. In order to build and link two instances of Linux as two different cells, we had to modify the project build setting quiet a lot. With the right modifications and setting, we got the two instances of Linux linked together in one binary. But when we tried to run the image on QEMU, we faced the problem of shared IRQ conflict. Because of lack of time we could not re- solve the conflict and decided to perform the experiment with simple multicell example. With OKL4v3.0 we real- ized that, we really cannot toggle between the two guest OSes even if we could resolve the IRQ issue.

6. Related Work

Current open-source (Xen, KVM) and commercial (Par- allels, VMware, Microsoft) VMMs provide support for state management through a technique called *Live Migra- tion* [7, 29]. Live Migration is the process of transition- ing the state of a virtual machine from one virtual machine monitor (VMM) to another, often between distinct physi- cal machines without halting the guest operating system. Live Migration captures the state of an entire OS and all of its applications as one unit which avoids many of the difficulties faced by migrating finer grained entities such as processes and local resources (file descriptors and net- work resources). The migration is usually managed by a daemon/agent process that runs outside of the VMM.

Self Migration [19, 15] is another state management technique that can be used to migrate an entire OS. The mi- gration is managed by a process that runs inside the source and destination OSes. The NomadBIOS [17] prototype vir- tualization system was built on top of a L4 microkernel and uses pre-copy migration to reduce

downtime. The system keeps the Guest OS running at the originating host while migrating, tracking changes to its address apace and send- ing updates containing the changes to the original image or a number of iterations.
Studies [9] have been done on the cost of migrating to the cloud using the *Pre-Copy* live migration strategy and the non-trival trade-off between minimizing the copy phase duration and maintaining an acceptable QoS (Quality of Service). These studies provide a model for optimizing the live migration Pre-Copy process. The task of migrat- ing pages consumes resources and can degrade further if live migration is performed in-band (using the same net- work bandwidth that is being used by the migration pro- cess). Fixed bandwidth migration is a model where the bandwidth used during the Pre-Copy phase remains con- stant. This model attempts to minimize the total cost of in-band migration and the number of pages that should be copied in the copy phase. The variable bandwidth migra- tion model is used for migration over time and requires an algorithm to decide before each phase home much band- width to use. These bandwidth model algorithms attempt to execute the Pre-Copy live migration strategy with a min- imal cost.
Other research involves using [41] case studies to evalu- ate the cost of live migration performance on modern inter- net applications. These applications contain features such as social networking which are different from traditional static data. These features introduce a specific workload and client/server communication patterns that are difficult to test and evaluate. This research concentrates on quanti- fying the effects on live VM migration.
SyncOS uses some of the standard memory and local resource migration strategies defined by Live and Self Mi- gration to capture and restore guest OS states hosted by the OLK4 Microvisor to cloud storage.

7. Conclusion & Future Work

Our research into OKL4 has provided many hurdles for us, but also stepping stones into the possibilities of device to device migration without having to be tethered to another machine for state transfer. We did not close in all the gaps in our initial design for SyncOS, but along the way discov- ered interesting challenges and unintended benefits during our implementation. By following others research in ar- eas of VM migration as discussed in Section 6, we applied these same techniques to mobile hypervisor above OKL4, a microkernel used by multiple cell phone vendors (Qual- comm, ST-Ericsson, Motorola, and others [26]).
Using a cloud provider for blob storage we should easily overcome the shortcomings of local data persistence limits while also enabling data protection and fault tolerance. We hope our contributions to VM state management for mo- bile devices continues development work in the field. We would like to thank OK:Labs for their source code, videos, newsgroup, and various resources used to refine our under- standing of OKL4 and aid in our creation of SyncOS.

7.1 Implementation Discoveries

While implementing the checkpointing algorithm based on CRAK [8] in the OKL4 kernel, we realized that, the OKL4 kernel itself doesn't support any kind of file system. So it was not possible to create the image file and store it on to the physical media. Moreover, the kernel doesn't have ac- cess to the device driver of the physical media (e.g. Flash). Lack of file system support within the Kernel lead us to two options, either write the flash device driver to store the image locally or send the image file to guest OS which has support for file systems. We tried to explore the second option, since that is exactly what we needed. We tried to explore the IPC functionality of OKL4 but didn't solve our problem since the IPC is for sharing data between the cells and not from Kernel to Cell. Parallelly we tried to explore the OKL4FS which is built in with OKL4, this also didn't help to solve our problem. OKL4FS provides the support to send large amount of data to be shared between Linux guest OS and other cells. This again doesn't support to share the data from Kernel to Cell. With this discovery and due to lack of time, we decided to keep this issue to be resolved in future.
While implementing the restore algorithm based on CRAK [8] again in the OKL4 kernel, we discovered that, the OKL4 kernel doesn't support the dynamic cell load- ing. Meaning, we cannot create the cells dynamically at the runtime once the OKL4 is booted up. OKL4 creates the cell, which are linked at build time, during boot pro- cess. This limitation made us to make a design choice to have local storage to store at least one guest OS image. This local storage space has to be shared between the ker- nel and cell, so that while checkpointing kernel can write to this shared storage and SyncOS can read it and upload it to server. While restore, the same shared

storage will be used by SyncOS to write the image downloaded from server and kernel will read the data from the shared loca- tion and initialize the cell accordingly during power up.

7.2 Future Works

Future work will focus on implementing additional Live Migration strategies and porting additional OSes to Syn- cOS and OKL4. Live Migration strategies such as *Pre- Copy* can provide performance improvements over *Stop- and-Copy* for mobile device constraints and cloud network latency. Pre-Copy is the process iteratively copying the state of a VM without stopping the execution of the VM being migrated. The migration process may take several iterations and will continuously transfer the dirty (memory pages) or delta state as smaller packets which can be trans- fered easier over a network to the Cloud. We will use the fixed or variable bandwidth migration strategies discussed by Breitgand et al. in his research into the cost of live mi- gration to the cloud [9]. Other SyncOS work will also provide some support for capturing the state of other OKL4 OSes such as OK:Symbian, and OK:Windows Mobile. These OSes will also run in the OKL4 Microvisor using the Secure Hy- perCell Technology. Additional OS support may concen- trate on porting other open source OSes such as BADA to the OKL4 Microvisor/Secure HyperCell Technology plat- form.

References

[1]. Android on OKL4. http://ertos.nicta.com.au/software/androidokl4/.

[2]. P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Har- ris, A. Ho, R. Neugebauery, I. Pratt, and A. Wareld. Xen and the art of virtualization. In *In Proceedings of the Nineteenth ACM Symposium on Operating Sys- tems Principles (Bolton Landing, NY, USA). SOSP '03*, October 2003.

[3]. BootingUp - OKL4. http://wiki.ok- labs.com/BootingUp

[4]. Building and emulating a basic ARM Linux sys- tem. http://elasticsheep.com/2011/01/building-and-emulating-a-basic-arm-linux-system/.

[5]. BuildingAndSimulating OKL4. http://wiki.ok-labs.com/BuildingAndSimulating?highlight=(build)(okl4)

[6]. P. M. Chen and B. D. Noble. When virtual is better than real. In *Proceedings of the 2001 Workshop on Hot Topics in Operating Systems*, May 2001.

[7]. C. Clark, K. Fraser, S. Hand, J. G. Hanseny, E. Jul, C. Limpach, I. Pratt, and A. War. Live Migration of Virtual Machines. University of Cambridge Com- puter Laboratory Department of Computer Science, 15 JJ Thomson Avenue, Cambridge, UK University of Copenhagen, Denmark.

[8]. CRAK - Linux Checkpoint/Restart Kernel Module. http://www.cs.fsu.edu/ baker/devices/projects/ale.

[9]. David Breitgand, Gilad Kutiel, Danny Raz. 'Cost- Aware Live Migration of Services in the Cloud. IBM, Haifa Research Lab, Department of Computer Sci- ence Technion, Israel, 2010

[10]. DebuggingGuide -OKL4. http://wiki.ok- labs.com/DebuggingGuide.

[11]. G. W. Dunlap, S. T. King, S. Cinar, M. A. B. , and P. M. Chen. ReVirt: enabling intrusion analy- sis through virtual-machine logging and replay. In *In Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, 2002.

[12]. Fixmo. http://www.fixmo.com/open-kernel-labs/

[13]. T. Garfinkel and M. Rosenblum. When Virtual is Harder than Real: Security Challenges in Virtual Ma- chine Based Computing Environments. In *10th Work- shop on Hot Topics in Operating Systems (HOTOS- X)*, 2005

[14]. Genode Operating System Framework. http://genode.org

[15]. J. G. Hansen. Ph.D. Thesis "Virtual Machine Mobil- ity with Self-Migration". Department of Computer Science, University of Copenhagen.

[16]. J. G. Hansen, E. Christiansen, and E. Jul. The Laun- dromat Model for Autonomic Cluster Computing. DIKU, Univ. of Copenhagen, DIKU/Microsoft Re- search

[17]. J. G. Hansen and A. K. Henriksen. Nomadic Op- erating Systems. Department of Computer Science, University of Copenhagen

[18]. J. G. Hansen and E. Jul. Lithium: Virtual Machine Storage for the Cloud. VMware Aarhus, Denmark, Bell Laboratories Alcatel-Lucent, Dublin, Ireland

[19]. J. G. Hansen and E. Jul. Self-migration of Operating Systems. Department of Computer Science, Univer- sity of Copenhagen, Copenhagen, Denmark
[20]. G. Heiser. Secure Embedded Systems Need Mi- crokernels. National ICT Australia and University of New South Wales, Sydney, Australia, December 2005
[21]. G. Heiser. Virtualization for Embedded Systems. Chief Technology Officer, Open Kernel Labs, Inc., 2007
[22]. G. Heiser. Your System is Secure? Prove it! NICTA* and University of New South Wales and Open Kernel Labs, Sydney, Australia, 2007
[23]. G. Heiser. The Role of Virtualization in Embedded Systems. Open Kernel Labs and NICTA and Univer- sity of New South Wales, Sydney, Australia, 2008.
[24]. G. Heiser. The Motorola Evoke QA4 A Case Study in Mobile Virtualization. Chief Technology Officer, Open Kernel Labs, Inc., 2009
[25]. G. Heiser, K. Elphinstone, I. Kuz, G. Klein, and S. M. Petters. Towards Trustworthy Computing Systems Taking Microkernels to the Next Level. NICTA and University of New South Wales, Sydney, Australia, 2007
[26]. Illinois Technology Association Spotlight Open Kernel Labs. http://www.illinoistech.org/story.aspx/303907
[27]. S. T. King, G. W. Dunlap, and P. M. Chen. De- bugging operating systems with time-traveling virtual machines, April 2005
[28]. G. Klein, K. Elphinstone, G. Heiser, J. Andronick Cock, P. Derrin, D. Elkaduwe, K. Engelhardt,Kolanski, M. Norrish, T. Sewell, H. Tuch, and Winwood. seL4: Formal Verification of an OS Kernel. NICTA, UNSW, Open Kernel Labs, ANU, 2009
[29]. P. Liu, Z. Yang, X. Song, Y. Zhou, H. Chen, and B. Zang. Heterogeneous Live Migration of Virtual Machines". Parallel Processing Institute, Fudan Uni- versity
[30]. J. Matthews. Virtualization and Componentization in Embedded Systems. Field Application Engineer, Open Kernel Labs, Inc., 2008
[31]. J. Matthews. Android Migration at the Speed of Light. Field Application Engineer, Open Kernel Labs, Inc., 2009
[32]. Microsoft Azure - Creating Shared Access Sig- natures (SAS). http://msdn.microsoft.com/en-us/library/ee395415.aspx.
[33]. OKL4 HTC Dream Project. http://ertos.nicta.com.au/software/okl4htcdream/
[34]. OKL4FS. http://wiki.ok-labs.com/OKL4FS
[35]. Open Kernel Labs, Inc. http://www.ok-labs.com/
[36]. Open Kernel Labs, Inc. 'The Nirvana Phone' Con- cept Specification and Draft Reference Architecture. Open Kernel Labs, Inc., 2009
[37]. OpenISR. http://isr.cmu.edu/
[38]. PC World - 3G Cellular Speeds by City http://www.pcworld.com/zoom?id=167391&page=1&zoomIdx=1
[39]. S. Smaldone, B. Gilbert, N. Bila, L. Iftode, de Lara, and M. Satyanarayanan. Leveraging Smart Phones to Reduce Mobility Footprints, 2009.
[40]. VMware Mobile Virtualization Platform, Virtual Appliances for Mobile Phones. http://www.vmware.com/products/mobile/
[41]. W. Voorsluys, J. Broberg, S. Venugopal, and R. Buyya. Cost of Virtual Machine Live Migration in Clouds: A Performance Evaluation, 2009
[42]. B. Weinberg and L. Pundit. Mobile Handset Tear- down: Designing and Deploying with Mobile Virtu-alization. Open Kernel Labs, Inc., 2009
[43]. What is Mobile Virtualization and Why is it Important? - ReadWriteCloud Zap: Transparent Checkpoint-Restart and Mi- gration Using Operating System Virtualization. http://www.ncl.cs.columbia.edu/research/migrate/.
[44]. Zap: Transparent Checkpoint-Restart and Mi- gration Using Operating System Virtualization. http://www.ncl.cs.columbia.edu/research/migrate/.