Towards Synchronization of Live Virtual Machines among Mobile Devices

Jeffrey Bickford
AT&T Security Research Center

Ramon Caceres
AT&T Labs-Research

Presenter: Bruce

Outline

- Introduction
- Preliminary Design
- Feasibility Study
- Conclusion and Future Work

Introduction

- People use mobile devices everyday

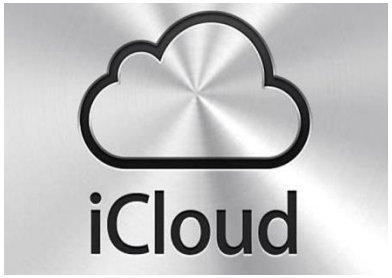
- Users want to switch devices seamlessly

Switch devices seamlessly

Synchronization based on app

- Some synchronization are based on application

  such as iCloud can synchronizes calendar, address book



- This requires separate and specific support to build into each application

Synchronization based on VM

- General synchronization solution based on Virtual Machine is needed.

1. System-level VM can improve security, manageability

2. VM encapsulate both data and computation state for a complete OS

Constrains of mobile environment

- Limited bandwidth, processing speed, storage, energy limitation

- Appropriate policies for how to represent changes and when to send them

Preliminary Design of VMsync

- Maintain a consistent VM image across multiple devices
- Minimize the time needed for change (penalty time)
- Minimize the total amount for data transferred

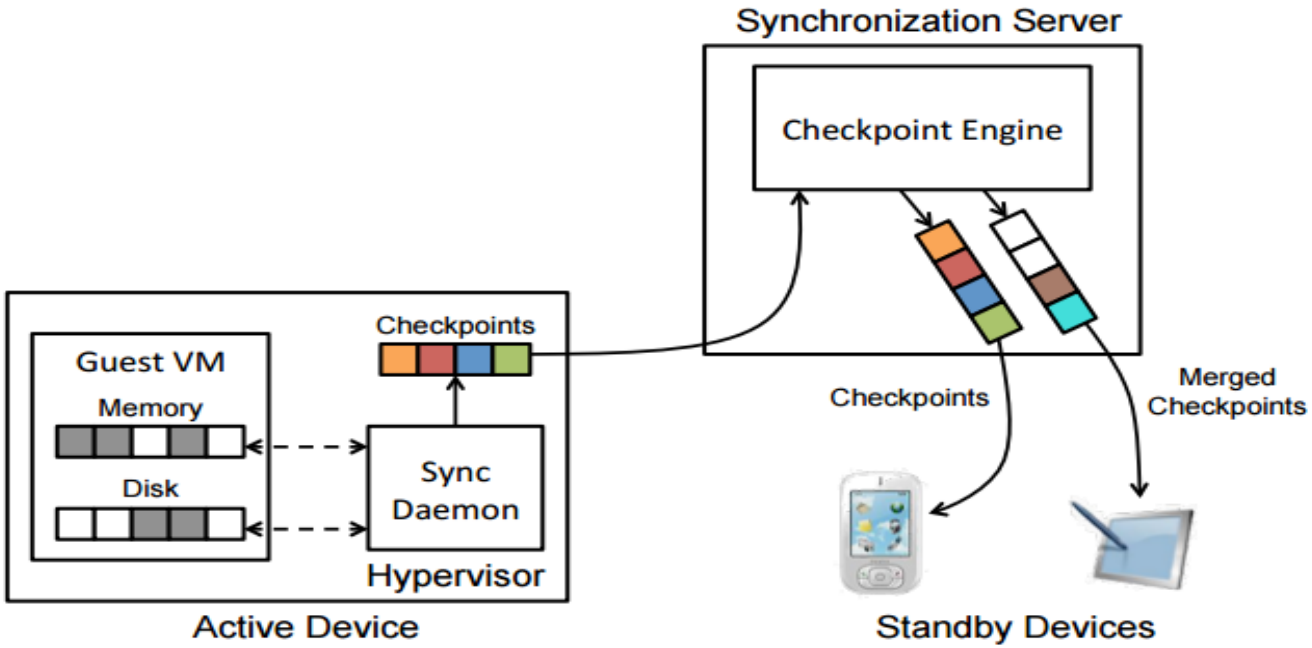


Figure 1: VMsync Architecture

1. Multiple host devices with virtualization support
2. Resource-rich server used as a synchronization point
3. Privileged daemon which monitors the guest VM state

1. Only one active VM will be running
2. The active VM will propagate changes of memory and file system state to the synchronization server(checkpoint)

Operation policy of VMsync

- When should the devices be updated with the latest state information
- Should the active device propagate changes periodically or use specific OS events to infer the best time or a mix
- The variety of hardware configuration introduces challenges

Feasibility Study

- Analysis changes to memory and the file system under various mobile workloads
- Goal is to determine how much data would be required to maintain a consistent VM state
- Application: web browsing, video playback, audio playback, audio recording

Experiment environment

- Guest domain

1. Android platform
2. VM: Android-x86 4.0.4
3. 512 MB of memory, 1 virtual CPU

- Host machine

1. Quadcore Intel core i7 860, 2.8 GHz, 12GRAM

(measure the changes of memory and file-system usage, no other computation overheads or other effects)

Measuring the Checkpoint Size

- Dirty Orig: each memory page or disk block that changed from the beginning of the workload is synced
- Diff Orig: the bytes changed from the beginning of the workload
- Dirty Prev: memory of disk block changed from the previous checkpoint
- Diff Prev: the bytes changed from the previous checkpoint

Measuring the Checkpoint Size

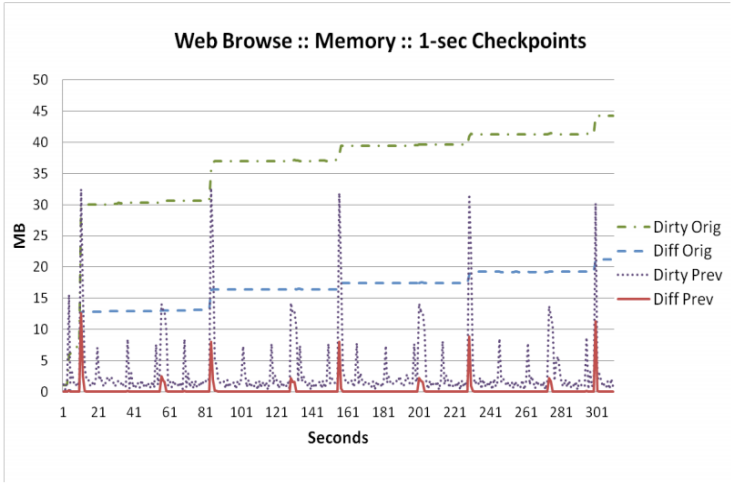


Figure 2: Memory contents change significantly with each new web page loaded.

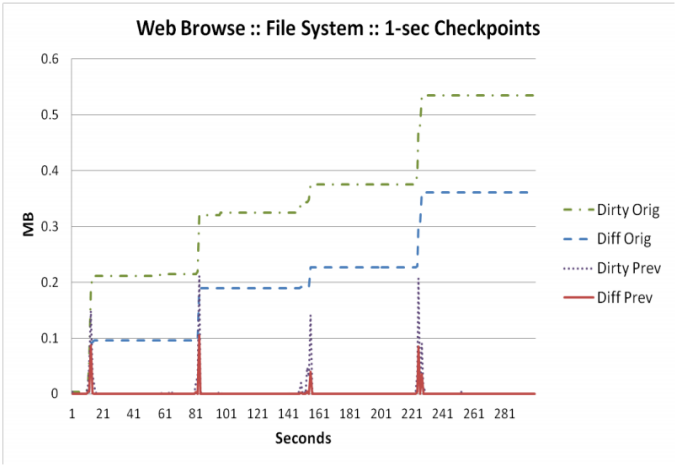


Figure 3: File-system contents change when the web browser caches data.

1. From a news article web CNN
2. The workload sleeps a predetermined seconds(45s) before scrolling down the page as a normal user
3. Then the workload sleeps again(25s) to simulate reading before moving to next page
4. Each time loading a new page, a significant change in both memory and file system

Measuring the Checkpoint Size

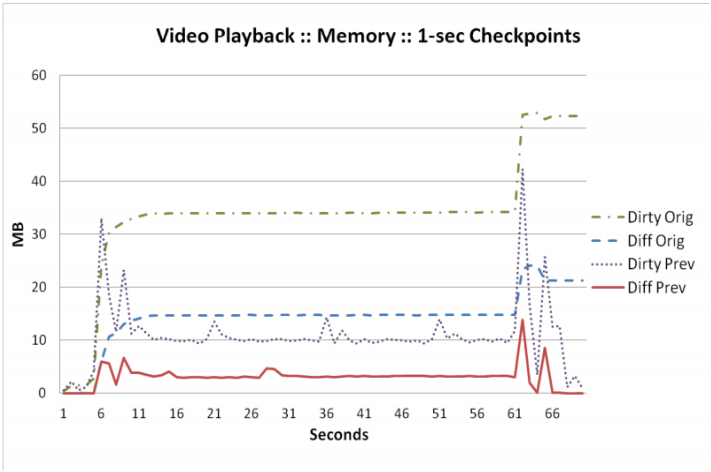


Figure 4: Most changes to memory occur during the initial loading of the video, with some final changes when the application closes.

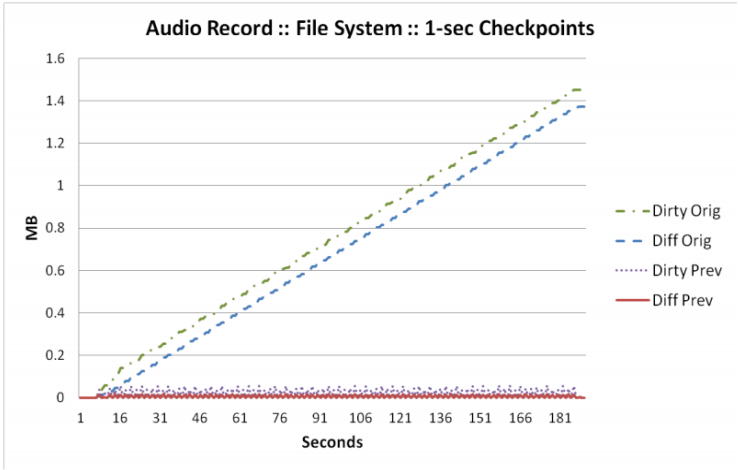


Figure 5: File-system contents change continuously during audio recording.

1. Video Playback: many pages are changed but the change in the system remains steady
The most significant memory change occurs at the beginning of the workload

2. Audio Record: the changes over the workload are continuously increasing
Every checkpoint the changes between the previous state are small, but add to the final total change

Measuring the Total Bytes Require for Sync

- For each workload, we vary the time between checkpoint from 1,5,10,30,60 seconds
- Final Diff: the number of megabytes changed at the end of the workload (remain constant)
- Final Dirty: the total number of megabytes required if sync each dirty page or block (remain constant)
- Incr Diff: total number of bytes changed over incremental checkpoints
- Incr Dirty: total number of complete pages that have changed over incremental checkpoints

Measuring the Total Bytes Require for Sync

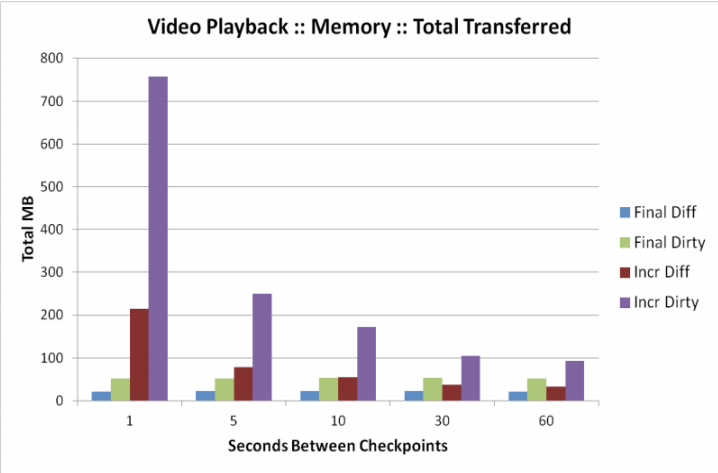


Figure 6: Transferring complete dirty memory pages involves three times more data than necessary.

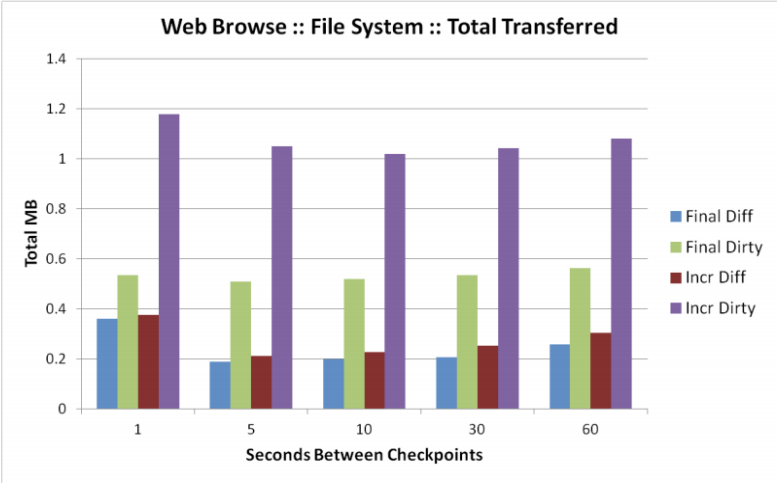


Figure 7: Transferring only the bytes that have changed is also advantageous in the file-system case.

1. Propagating each memory page that has changed would require three times the amount of the data compared to propagating the difference of each page

2. Propagating dirty blocks incurs a cost about five times higher than performing a final diff

Measuring the Total Bytes Require for Sync

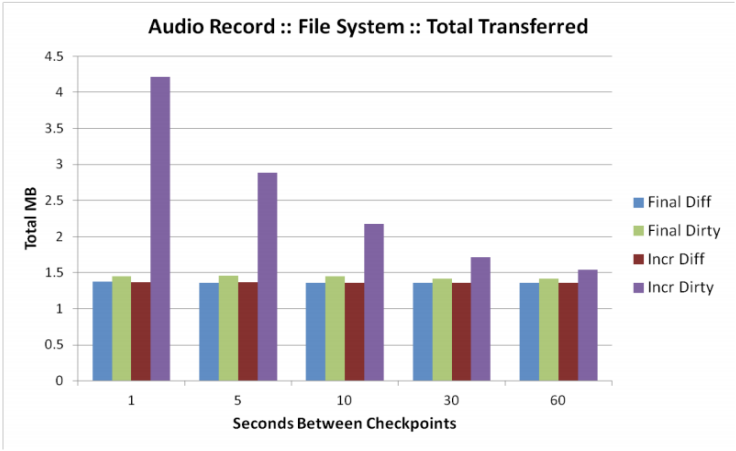


Figure 8: Waiting before propagating dirty file-system blocks approaches the cost of a differencing policy for apps that perform sequential writes.

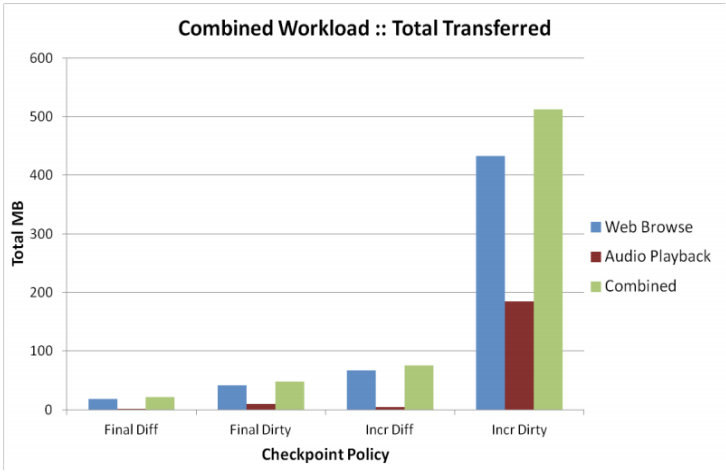


Figure 9: Playing music in the background of a web browsing session has minimal effect on the overall system state.

1. Audio playback continuously modifies a fixed number of memory pages and does not change the file system

2. Compared to memory state, the changes in file system state are fairly minimal, even in a file-system intensive workload such as the audio recording

Conclusion

1. Only propagate dirty pages or blocks will transfer large amount of data and will not be feasible on mobile devices

2. Transfer the difference of previous checkpoint will be feasible

3. Increase the time between checkpoints will save bytes but may require a higher switch latency if a user switch devices during that time

4. VMsync is feasible

Thank you