Just-in-Time Provisioning for Cyber Foraging

6/27/2013

Kiryong Ha∗, Padmanabhan Pillai†, Wolfgang Richter∗
Yoshihisa Abe∗, Mahadev Satyanarayanan∗

*Carnegie Mellon University, †Intel Labs

Cloud Offloading

Rich, interactive applications are emerging in mobile context

- Apple's Siri, AR apps..
- Wearable devices push this trend even more!

Cloud offloading

- These applications are too expensive to run on clients alone!
- Offload computation to a back-end server at cloud
- MAUI (Mobisys '10), Odessa (MobiSys '11), COMET (OSDI '12)

Today's cloud is a suboptimal place; high latency and limited bandwidth

Cloudlet as a Nearby Offload Site

Cloudlet: *an nearby offloading site dispersed at the edges of the Internet*

→ ***Let's bring the cloud closer!***

Nokia Siemens Networks & IBM

Nvidia

How to launch a custom back-end server at an arbitrary edge?

Just-in-Time Provisioning

Challenges in provisioning

1. Support widest range of user customization including OS, language, and library
2. Strong isolation between untrusted computations
3. Access control, metering, dynamic resource management, …

A traveler wants to use natural language translation with speaker-trained voice recognition

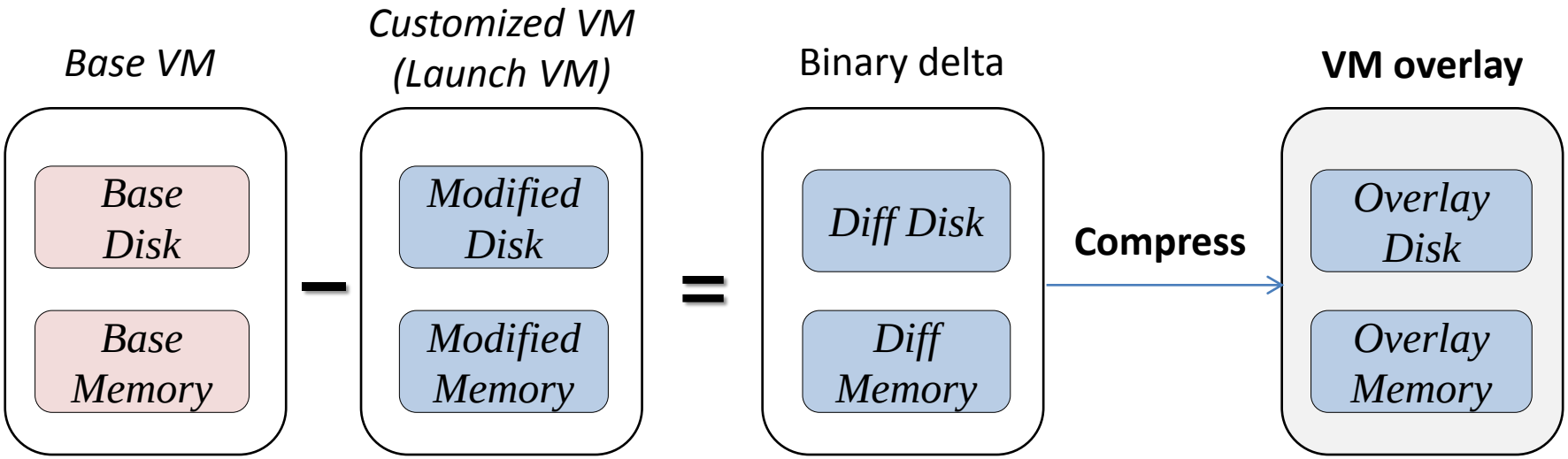
→ *VM (virtual machine) cleanly encapsulates this complexity, but delays provisioning*

GOAL : Just-in-time provisioning of a custom VM for offloading

VM Synthesis

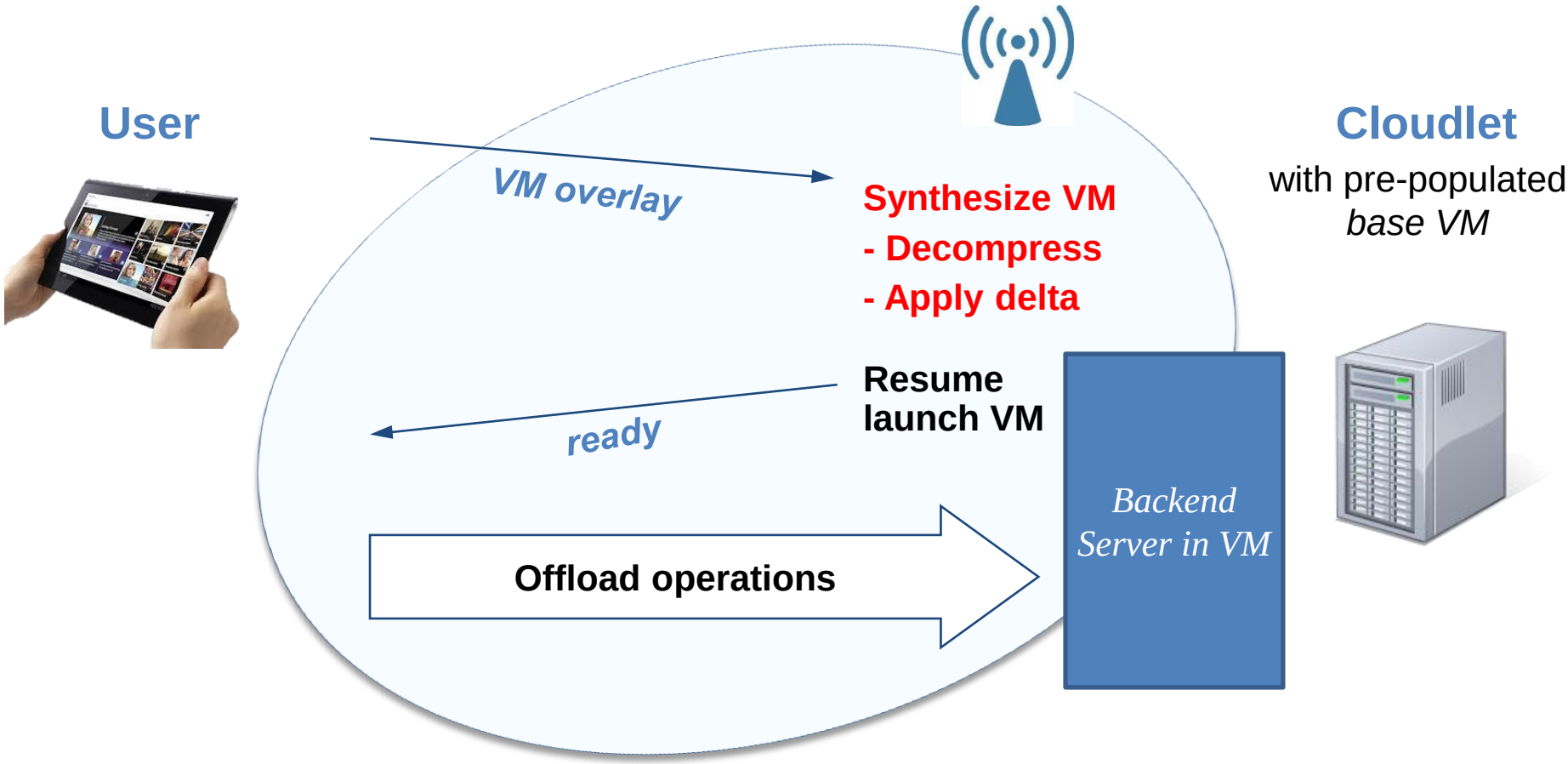
VM Synthesis: dividing a custom VM into two pieces

1) *Base VM*: Vanilla OS that contains kernel and basic libraries
2) *VM overlay*: A binary patch that contains customized parts



VM Synthesis

Steps for VM synthesis



VM Synthesis – Baseline Performance

- Performance measurement with rich, interactive applications
- *Base VM*: *Windows 7* and *Ubuntu 12.04*
  - **8GB** *base disk* and **1GB** *base memory*

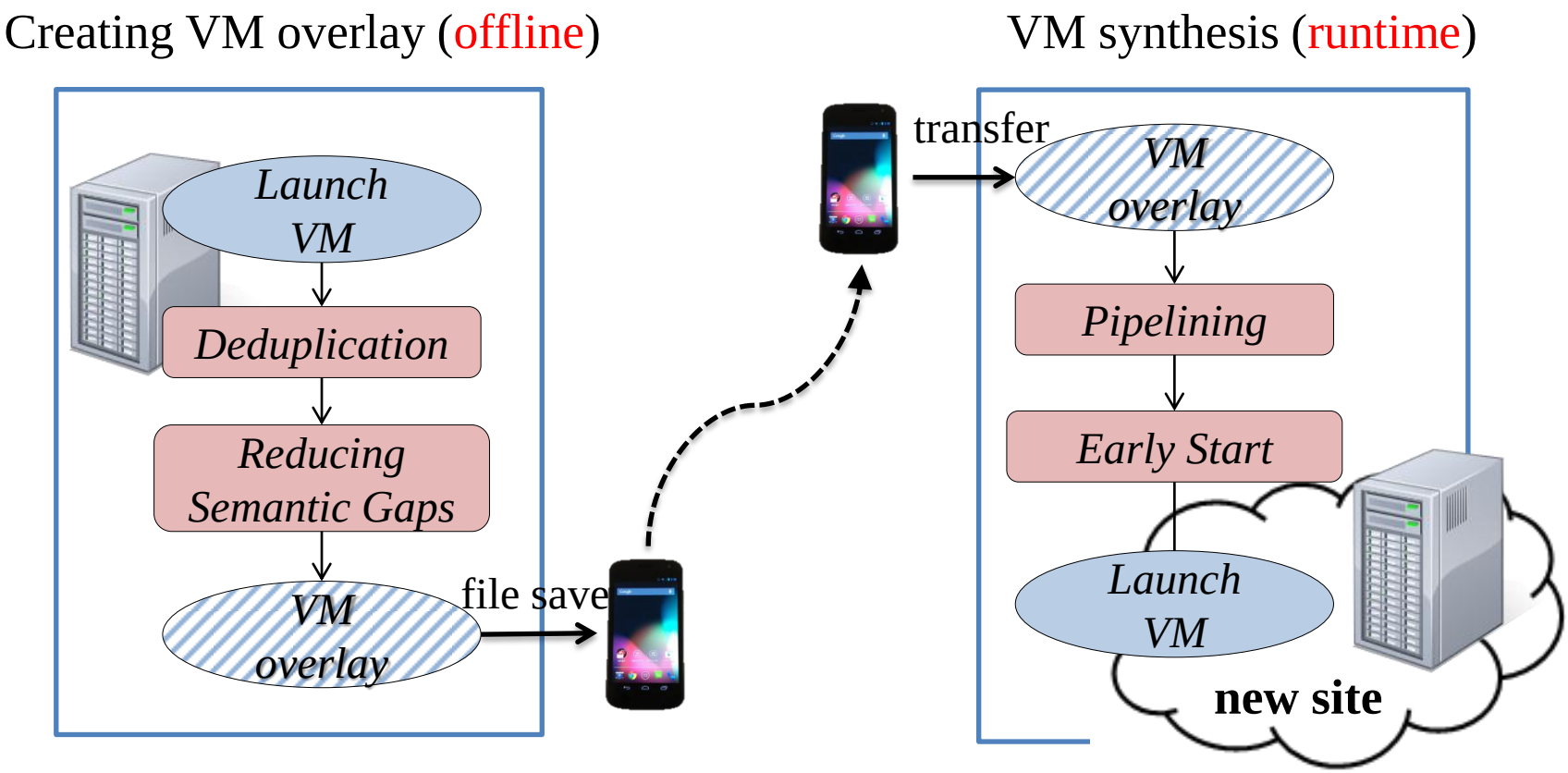
Application	Install size (MB)	Overlay Size		Synthesis time (s)
		Disk (MB)	Memory (MB)	
OBJECT	39.5	92.8	113.3	**62.8**
FACE	8.3	21.8	99.2	**37.0**
SPEECH	64.8	106.2	111.5	**63.0**
AR	97.5	192.3	287.9	**140.2**
FLUID	0.5	1.8	14.1	**7.3**

802.11n WiFi (average 38 Mbps)

Reduce VM synthesis time as little as 10 seconds!

Overview of Optimizations

1. Minimize *VM overlay* size
2. Accelerate VM synthesis



1.1 Deduplication

Approach

- Remove redundancy in the VM overlay
- Sources of redundancy
  1) Between *base VM* and *VM overlay*
     - Shared library copied from base disk
     - Loaded executable binary from base disk
  2) Between *VM overlay's memory* and *disk*
     - Page cache, disk I/O buffer

1.1 Deduplication

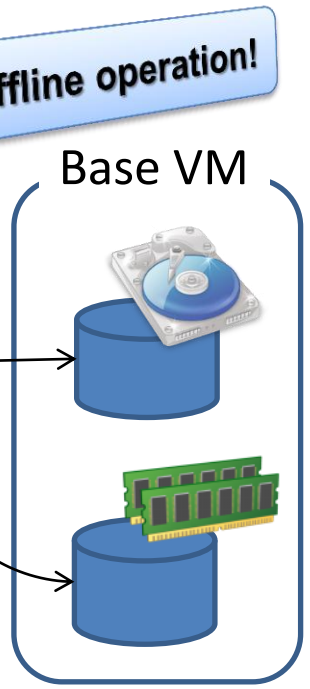
1. Get the list of modified (disk, memory) chunks at the *customized VM*
2. Perform deduplication to reduce this list to a minimum
   - Compare to *1)* ***base disk****,* 2) ***base memory***, 3) other chunks within **itself**
   - Compare between ***modified memory*** *and* ***modified disk***

type	offset	Data
mem	4096	data
mem	16384	data
mem	20480	data
mem	28672	data
mem	36864	data
...		
disk	0	data
disk	16384	data

<Modified chunks>

type	offset	reference	Data or pointer
mem	4096	unique	data
mem	16384	self	
mem	20480	Base disk	
mem	28672	Base memory	
mem	36864	unique	data
...			
disk	0	unique	data
disk	16384	overlay mem	

<Overlay chunks>



1.2 Reducing Semantic Gaps

VM is a strong black box

- It ensures isolation between the host, the guest, and other guests
- But, VMM cannot interpret high-level information of memory and disk

Example: Download 100 MB file over network and delete it

- Ideally, it should result in no increase in VM overlay size
- However, VMM will see **200 MB of modifications**:
  - 100 MB of changed disk state
  - 100 MB of changed memory state (in-memory I/O buffer cache)

→ Let's include only the state that actually matters to the guest OS

1.2 Reducing Semantic Gaps – Disk

Disk semantic gap between VMM and Guest OS

- File deletion operations only mark blocks as deleted, without discarding the contents
- VMM can't distinguish between deleted and valid contents

Implementation: TRIM support

- ATA standard originally designed to improve SSD's overwrite performance
- Allows an OS to inform a SSD which blocks of data are no longer in use

TRIM support at QEMU

- Modify QEMU's IDE emulation to enable TRIM
- **Guest OS agnostic:** Linux (kernel 2.6.28), Mac OS X (June 2011), Window 7

```
time:1349399899.473041, sector_number:5244928, sector_size:16
time:1349399899.473046, sector_number:5375998, sector_size:3394
...
```

1.2 Reducing Semantic Gaps – Memory

Memory semantic gap between VMM and Guest OS

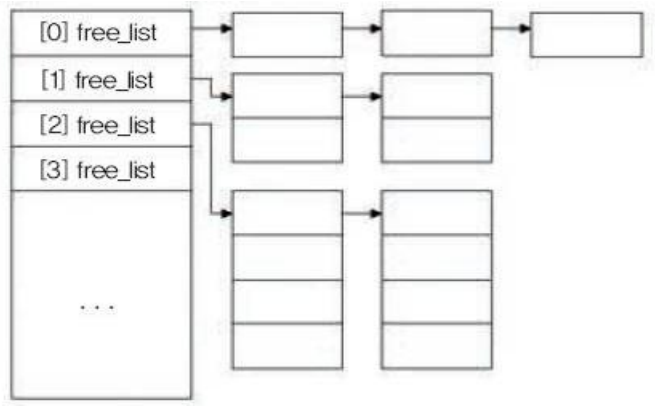
- Released memory is moved to the OS's free page list, but is still filled with garbage
- VMM can't distinguish between valid memory and garbage data

Approach

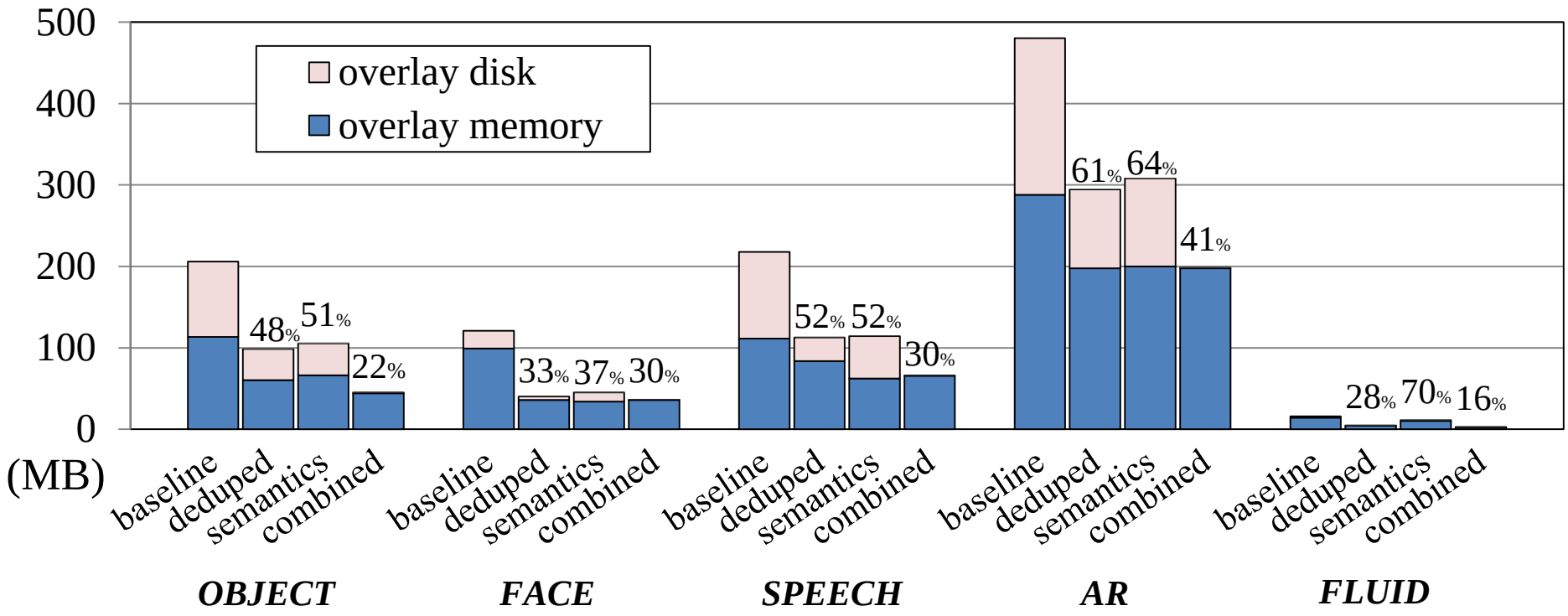
- No way to communicate free page information between the guest and VMM
  → scan memory snapshot

Implementation

- Insert a small agent at guest OS
  - Get memory address of the kernel data structure that has the free memory list
  - Need guest help : *currently, works only in Linux*
- Extract free memory pages by traversing the data structure



VM Overlay Size

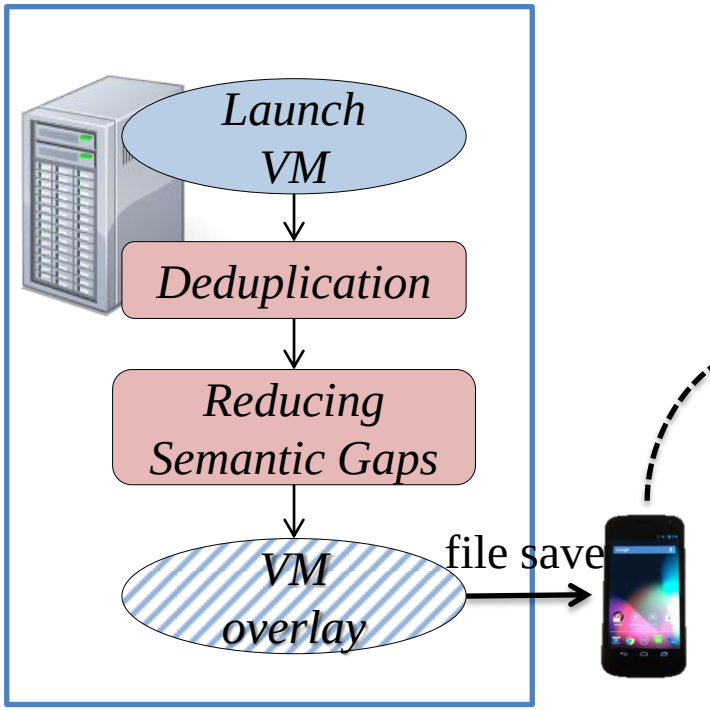


- Deduplication optimization reduces the VM overlay size to 44%
- Using semantic knowledge reduces the VM overlay size to 55%
- Both applied together, overlay size is reduced to **28% of *baseline***

Overview of Optimizations

1. ~~Minimize *VM overlay* size~~
2. **Accelerate VM synthesis**

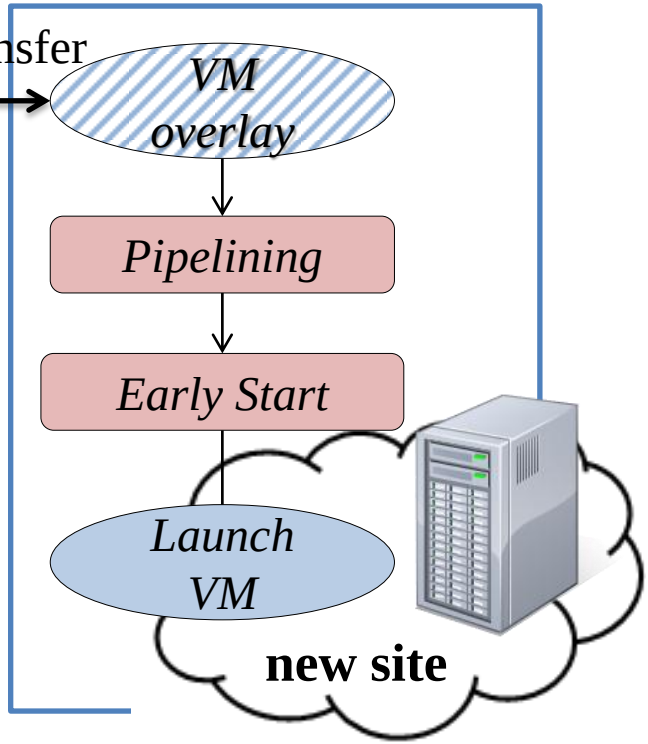
Creating VM overlay (offline)



file save

transfer

VM synthesis (runtime)



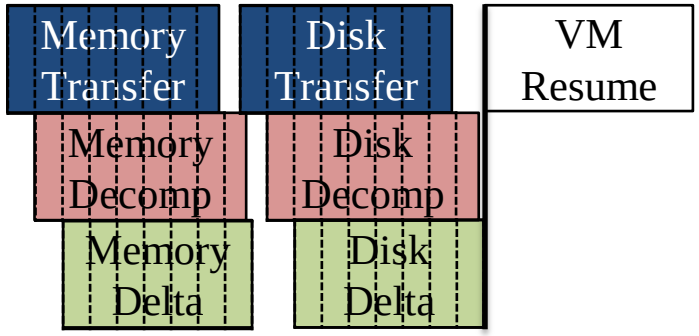
2.1 Pipelining

- Steps for VM synthesis

  ① Transfer VM overlay ② Decompress ③ Apply delta

<Sequential>

<Pipelined>



- Complexities in removing inter-dependencies among blobs

2.2 Early Start

Approach

- From user's perspective, first response time of offloading is most important
- Starting VM even before finishing VM synthesis?

→ Do not wait until VM synthesis finishes, but start offloading immediately and process the request while synthesis is ongoing

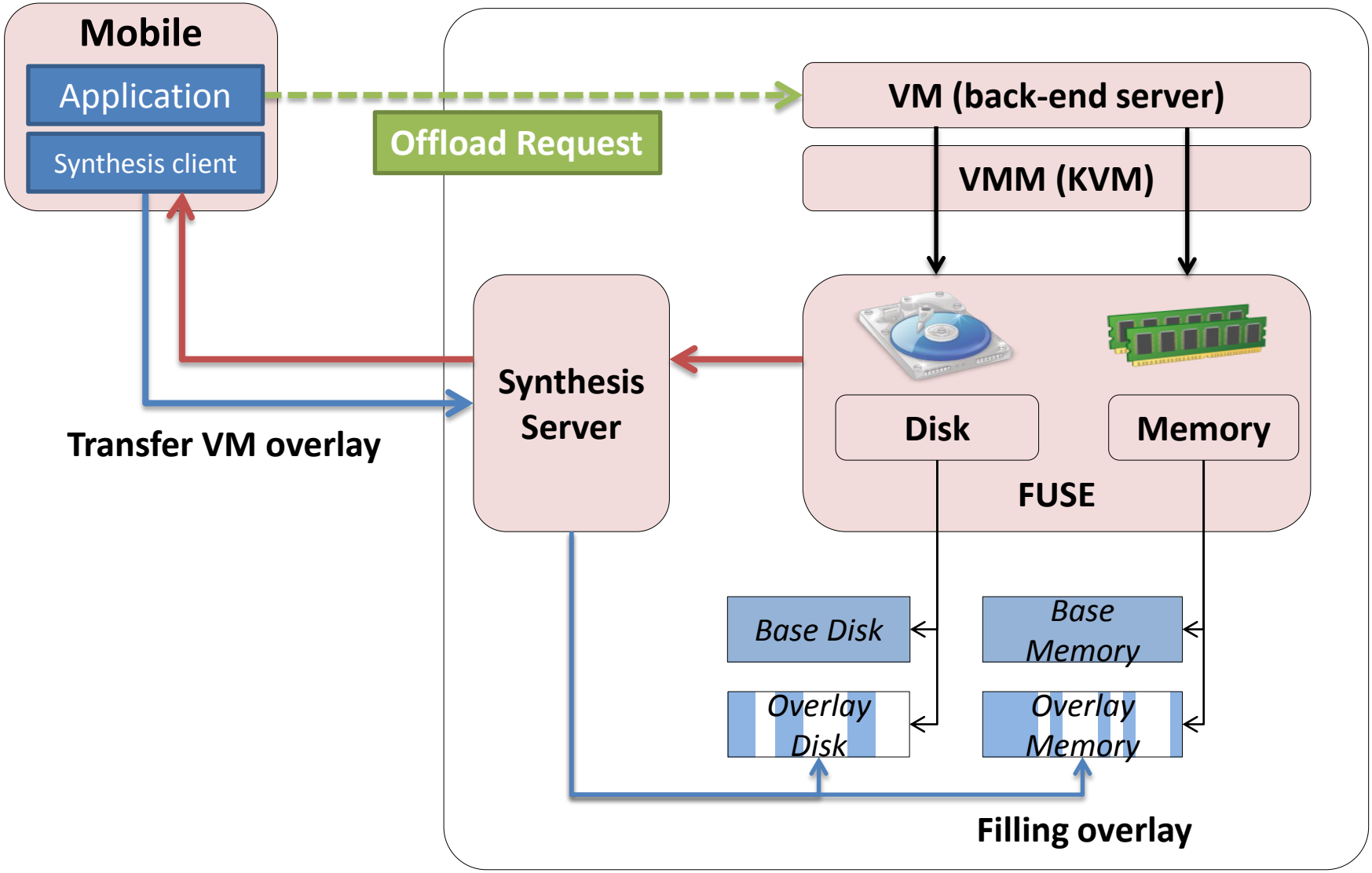
2.2 Early Start

Implementation

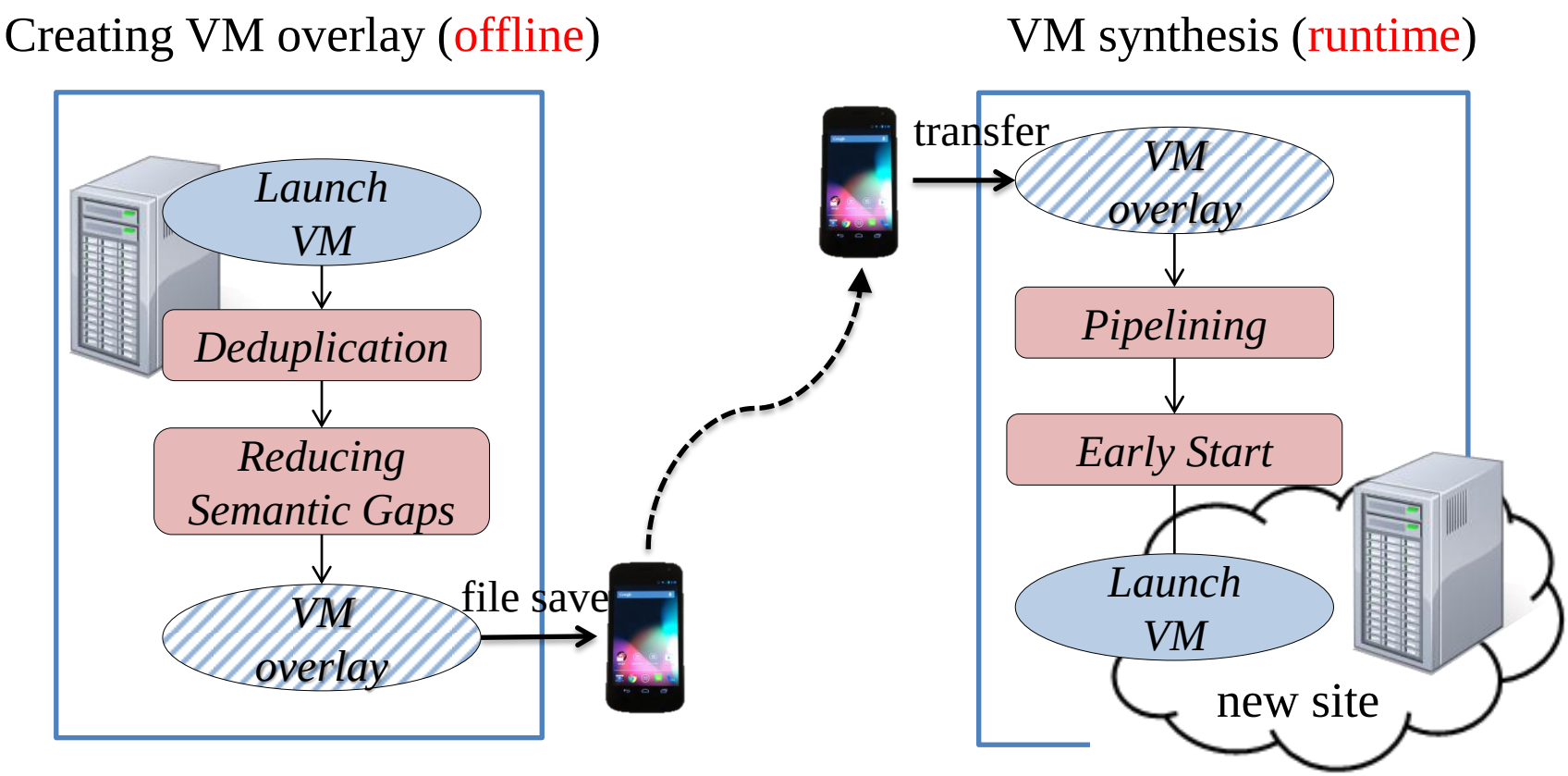
1) **Reorder the chunks in estimated access-order**
2) Break the ordered overlay into **smaller segments for demand fetching**

→ Start the VM and begin streaming the segments in order, but also allow out-of-order demand fetches to preempt the original ordering

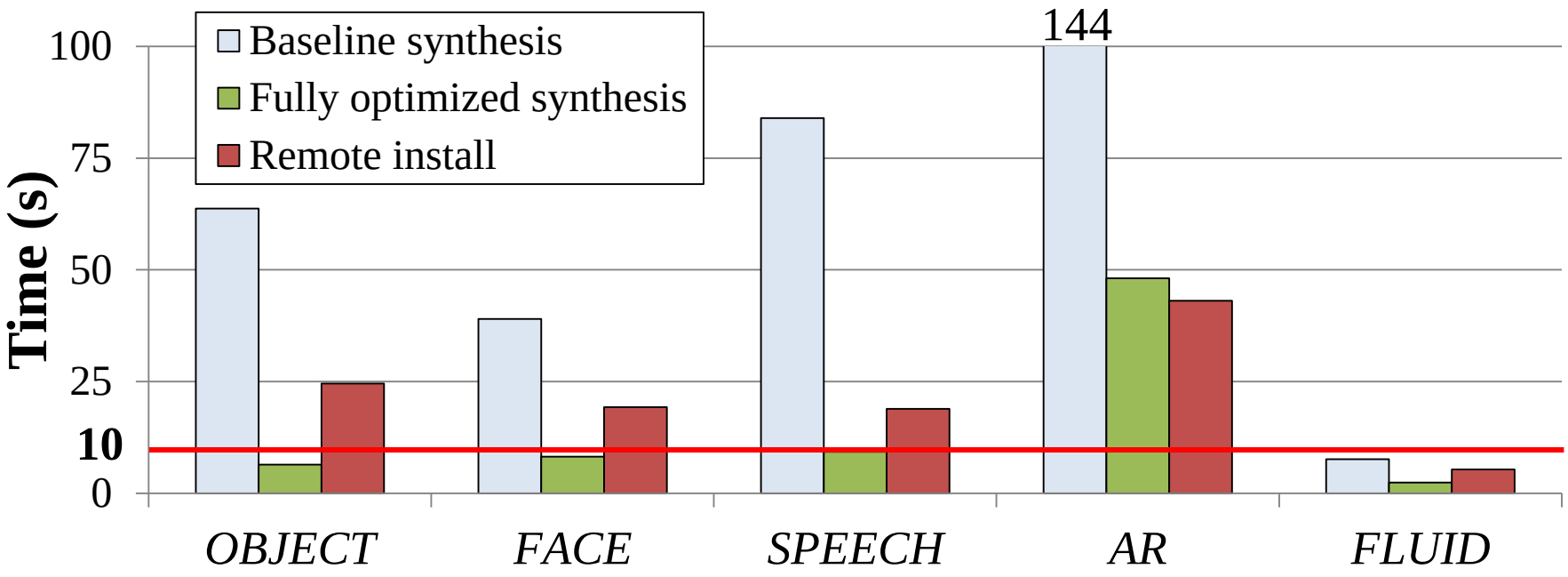
Diagram of Early Start



Review of Optimizations



First-response time compared to *baseline*



* Chunks are ordered with segment size of 1 MB

Time between starting VM synthesis and receiving the first offload result

- It is faster than remote installation maintaining strong guarantees
- Except *AR*, we can get first-response within 10 seconds (up to 8x improvement)

Future work & Conclusion

Future work

- **Open source :** http://github.com/cmusatyalab/elijah-cloudlet
- Integrate with OpenStack (open-source cloud computing platform)

Conclusion

- Cloudlets support **resource-intensive** and **interactive** mobile apps
- Physical dispersion of cloudlets makes their provisioning a challenge
- We have shown how cloudlets can be **rapidly provisioned**