

EECS 262a
Advanced Topics in Computer Systems
Lecture 14

VM Migration/Cloudlets
October 11th, 2023

John Kubiawicz
Electrical Engineering and Computer Sciences
University of California, Berkeley

<http://www.eecs.berkeley.edu/~kubitron/cs262>

Today's Papers

- [Live Migration of Virtual Machines](#)
C. Clark, K. Fraser, S. Hand, J. Hansen, E. Jul, C. Limpach, I. Pratt, A. Warfield. Appears in *Proceedings of the 2nd Symposium on Networked Systems Design and Implementation (NSDI)*, 2005
- Together, as a group:
- [The Case for VM-Based Cloudlets in Mobile Computing](#)
Mahadev Satyanarayanan, Paramvir Bahl, Ramon Caceres, Nigel Davies. Appears in *IEEE Journal on Pervasive Computing*, Vol 8, No 4, 2009
- [Transient customization of mobile computing infrastructure](#)
Adan Wolbach, Jan Harkes, Srinivas Chellappa, M. Satyanarayanan. Appears in *Proceedings of the First Workshop on Virtualization in Mobile Computing (MobiVirt)*, 2008
- Today: explore value of leveraging the VMM interface for new properties (migration and edge computing), many others as well including debugging and reliability
- Thoughts?

10/11/2023

Cs262a-F23 Lecture-14

2

Why Migration is Useful

- Load balancing for long-lived jobs (why not short lived?)
- Ease of management: controlled maintenance windows
- Fault tolerance: move job away from flaky (but not yet broken hardware)
- Energy efficiency: rearrange loads to reduce A/C needs
- Data center is the right target

10/11/2023

Cs262a-F23 Lecture-14

3

Benefits of Migrating Virtual Machines Instead of Processes

- Avoids 'residual dependencies'
- Can transfer in-memory state information
- Allows separation of concern between users and operator of a datacenter or cluster

10/11/2023

Cs262a-F23 Lecture-14

4

Background – Process-based Migration

- Typically move the process and leave some support for it back on the original machine
 - E.g., old host handles local disk access, forwards network traffic
 - these are “residual dependencies” – old host must remain up and in use
- Hard to move exactly the right data for a process – which bits of the OS must move?
 - E.g., hard to move TCP state of an active connection for a process

VMM Migration

- Move the whole OS as a unit – don’t need to understand the OS or its state
- Can move apps for which you have no source code (and are not trusted by the owner)
- Can avoid residual dependencies in data center thanks to global names
- Non-live VMM migration is also useful:
 - Migrate your work environment home and back: put the suspended VMM on a USB key or send it over the network
 - Collective project, “Internet suspend and resume”

Goals / Challenges

- Minimize downtime (maximize availability)
- Keep the total migration time manageable
- Avoid disrupting active services by limiting impact of migration on both migratee and local network

VM Memory Migration Options

- Push phase
- Stop-and-copy phase
- Pull phase
 - Not in Xen VM migration paper, but in SnowFlock

Implementation

- Pre-copy migration
 - Bounded iterative push phase
 - » Rounds
 - » Writable Working Set
 - Short stop-and-copy phase
- Be careful to avoid *service degradation*

Live Migration Approach (I)

- Allocate resources at the destination (to ensure it can receive the domain)
- Iteratively copy memory pages to the destination host
 - Service continues to run at this time on the source host
 - Any page that gets written will have to be moved again
 - Iterate until a) only small amount remains, or b) not making much forward progress
 - Can increase bandwidth used for later iterations to reduce the time during which pages are dirtied
- Stop and copy the remaining (dirty) state
 - Service is down during this interval
 - At end of the copy, the source and destination domains are identical and either one could be restarted
 - Once copy is acknowledged, the migration is *committed* in the transactional

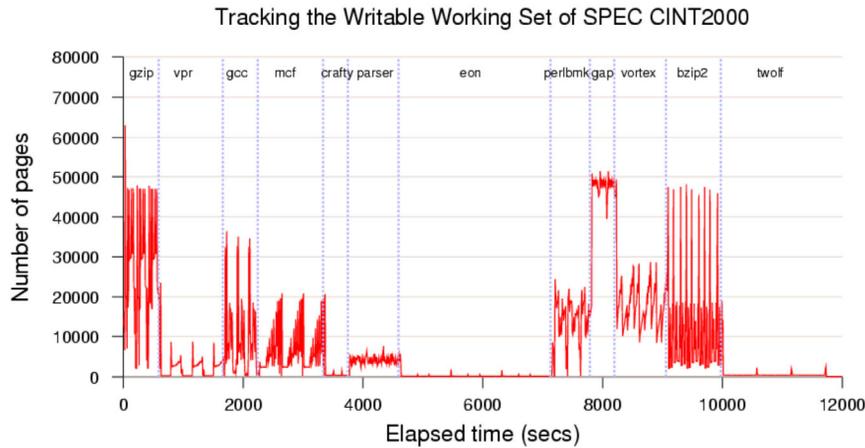
Live Migration Approach (II)

- Update IP address to MAC address translation using “gratuitous ARP” packet
 - Service packets starting coming to the new host
 - May lose some packets, but this could have happened anyway and TCP will recover
- Restart service on the new host
- Delete domain from the source host (no residual dependencies)

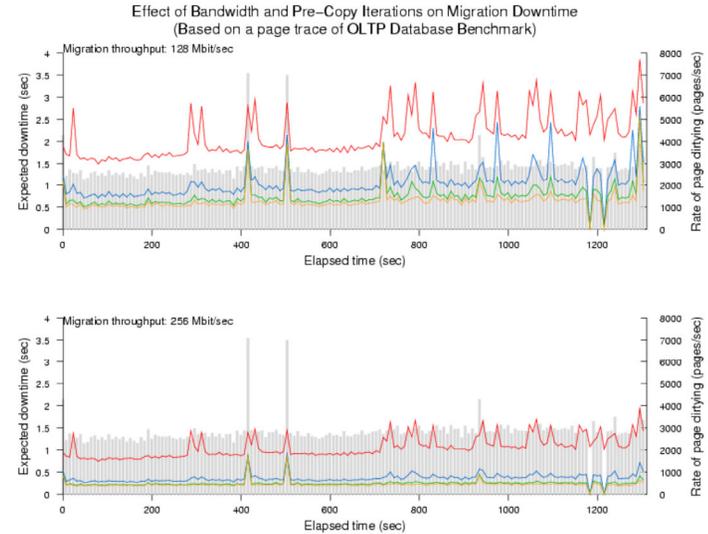
Tracking the Writable Working Set

- Xen inserts shadow pages under the guest OS, populated using guest OS's page tables
- The shadow pages are marked read-only
- If OS tries to write to a page, the resulting page fault is trapped by Xen
- Xen checks the OS's original page table and forwards the appropriate write permission
- If the page is not read-only in the OS's PTE, Xen marks the page as dirty

Writable Working Set

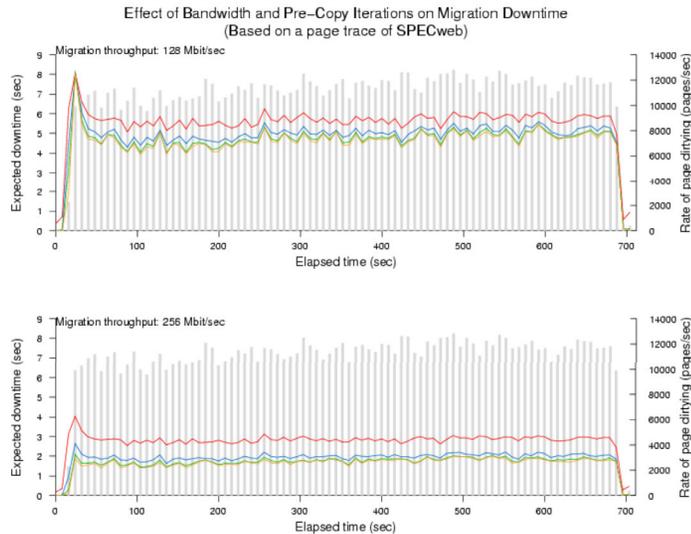


OLTP Database



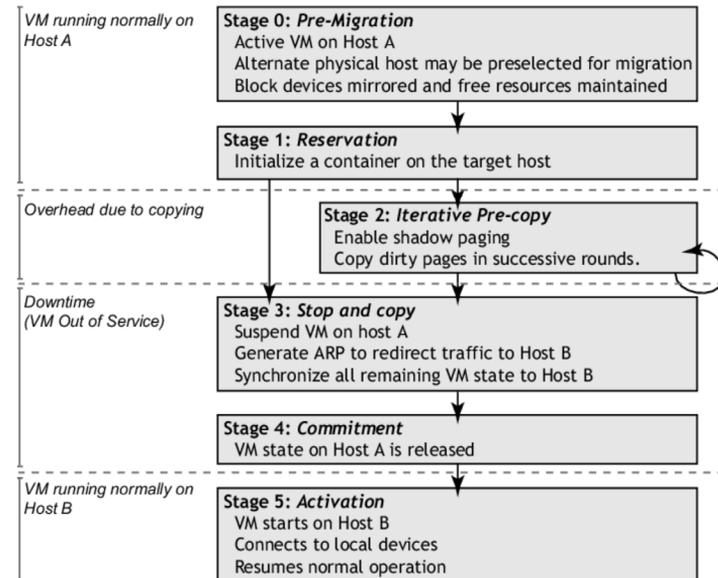
- Compare with stop-and-copy:
 - 32 seconds (128Mbit/sec) or 16seconds (256Mbit/sec)

SPECweb



- Compare with stop-and-copy:
 - 32 seconds (128Mbit/sec) or 16seconds (256Mbit/sec)

Design Overview



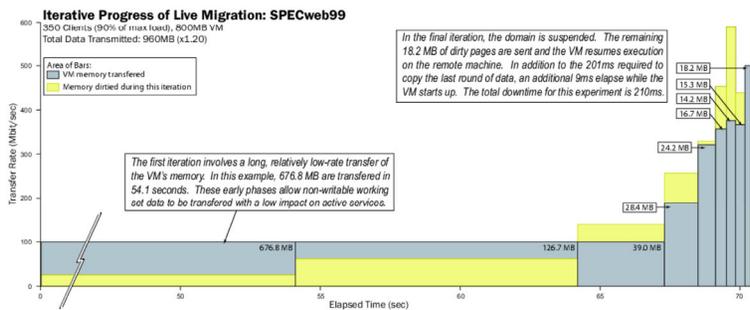
Handling Local Resources

- Open network connections
 - Migrating VM can keep IP and MAC address.
 - Broadcasts ARP new routing information
 - » Some routers might ignore to prevent spoofing
 - » A guest OS aware of migration can avoid this problem
- Local storage
 - Network Attached Storage

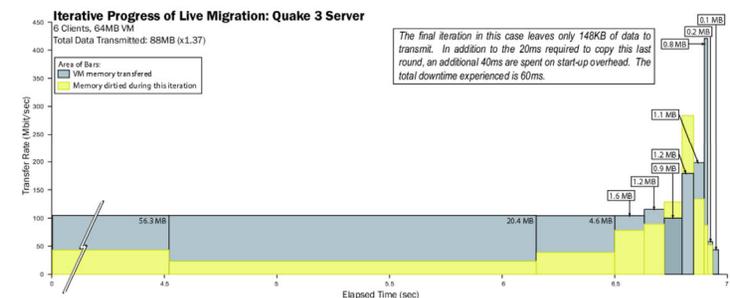
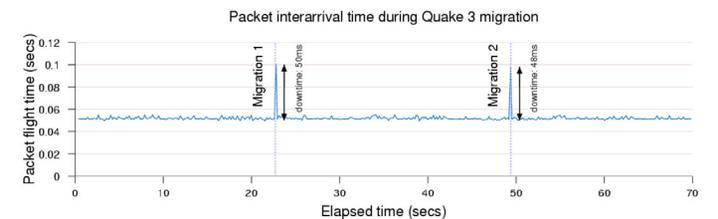
Types of Live Migration

- Managed migration: move the OS without its participation
- Managed migration with some paravirtualization
 - Stun rogue processes that dirty memory too quickly
 - Move unused pages out of the domain so they don't need to be copied
- Self migration: OS participates in the migration (paravirtualization)
 - Harder to get a consistent OS snapshot since the OS is running!

Complex Web Workload: SPECweb99



Low-Latency Server: Quake 3



Summary

- Excellent results on all three goals:
 - Minimize downtime/max availability, manageable total migration time, avoid active service disruption
- Downtimes are very short (60ms for Quake 3 !)
- Impact on service and network are limited and reasonable
- Total migration time is minutes
- Once migration is complete, source domain is completely free

Is this a good paper?

- What were the authors' goals?
- What about the evaluation/metrics?
- Did they convince you that this was a good system/approach?
- Were there any red-flags?
- What mistakes did they make?
- Does the system/approach meet the "Test of Time" challenge?
- How would you review this paper today?

BREAK

Virtualization in the Cloud

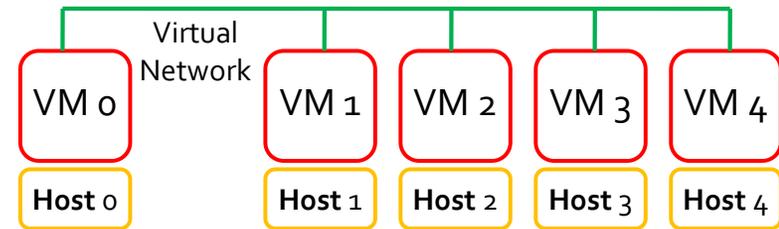
- True "Utility Computing"
 - Illusion of infinite machines
 - Many, many users
 - Many, many applications
 - Virtualization is key
- Need to scale bursty, dynamic applications
 - Graphics render
 - DNA search
 - Quant finance
 - ...

Application Scaling Challenges

- Awkward programming model: “Boot and Push”
 - **Not stateful**: application state transmitted explicitly
- Slow response times due to big VM swap-in
 - **Not swift**: Predict load, pre-allocate, keep idle, consolidate, migrate
 - Choices for full VM swap-in: boot from scratch, live migrate, suspend/resume
- Stateful and Swift equivalent for process?
 - Fork!

SnowFlock (Optional Paper): VM Fork

Stateful swift cloning of VMs



- State inherited up to the point of cloning
- Local modifications are not shared
- Clones make up an impromptu cluster

Fork has Well Understood Semantics

partition data
fork N workers
 Parallel Computation
 if child:
 work on i^{th} slice of data

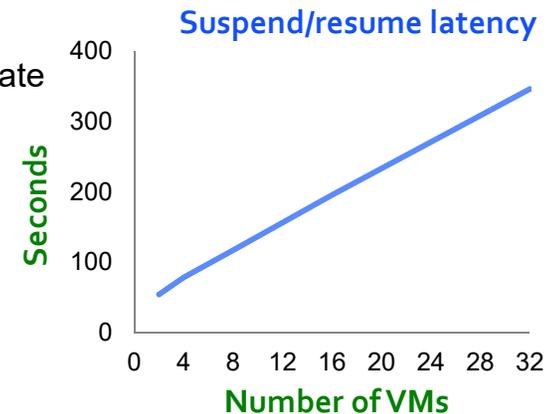
if more load:
fork extra workers
 Load-balancing Server
 if load is low:
 dealloc excess workers

trusted code
fork
 Sandboxing
 if child:
 untrusted code

if cycles available:
fork worker
 Opportunistic Computation
 if child:
 do fraction of long computation

VM Fork Challenge – Same as Migration!

- Transmitting **big** VM State
 - VMs are big:
OS, disk, processes, ...
 - Big means slow
 - Big means not scalable



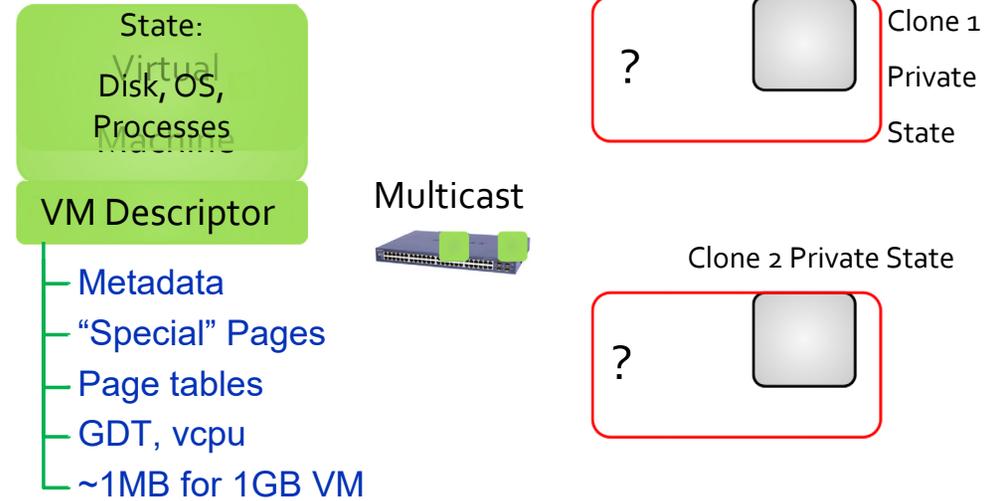
- Same fundamental bottleneck issues as VM Migration – shared I/O resources: host and network

SnowFlock Insights

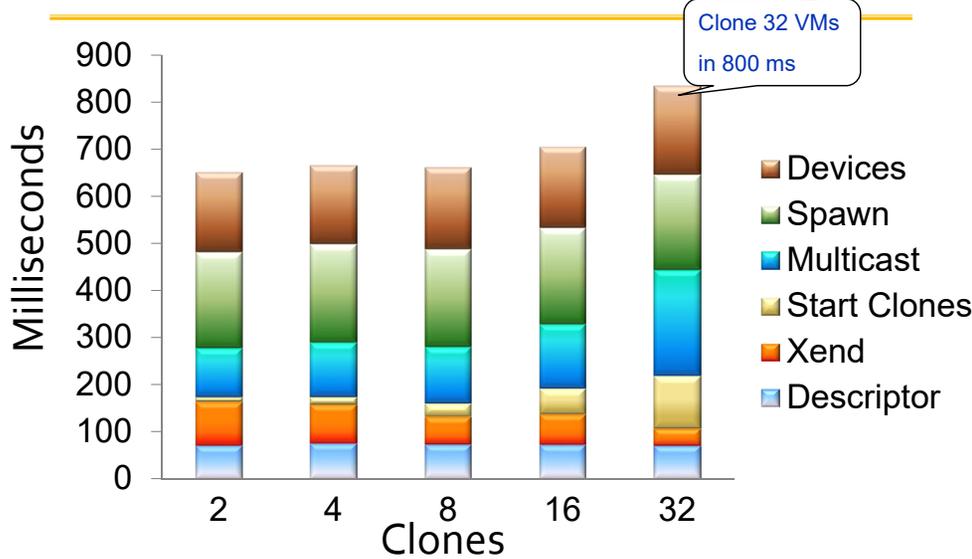
- VMs are BIG: Don't send all the state!
- Clones need little state of the parent
- Clones exhibit common locality patterns
- Clones generate lots of private state

SnowFlock Secret Sauce

g. Multicast cloning with shared private state



Clone Time



Scalable Cloning: Roughly Constant

Why SnowFlock is Fast

- Start only with the basics
- Send only what you **really need**
- Leverage IP Multicast
 - Network hardware parallelism
 - Shared prefetching: exploit **locality patterns**
- Heuristics
 - Don't send if it will be overwritten
 - Malloc: exploit **clones generating new state**

Edge Computing

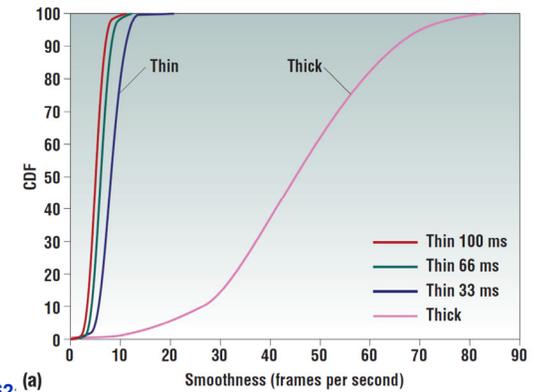
- Wikipedia:

Edge computing is a **distributed computing** paradigm in which computation is largely or completely performed on distributed device **nodes** known as **smart devices** or **edge devices** as opposed to primarily taking place in a centralized **cloud environment**.

- Why compute on the edge?
 - Latency: Importance of human interactivity
 - Privacy: Keep sensitive data local
 - Reliability: Keep computing during network partitions

The Case for VM-Based Cloudlets

- Well-referenced paper in IoT/Edge Computing world (seems like everyone references it!)
- Basic premise: Need edge computing
 - Mobile hardware too weak
 - Somehow extend cloud to the edge so that there is a device close by to mobile devices that need services
- Just a little latency can completely destroy responsiveness of video intrativeness:
 - Compare even 33ms to “Thick” local computation:

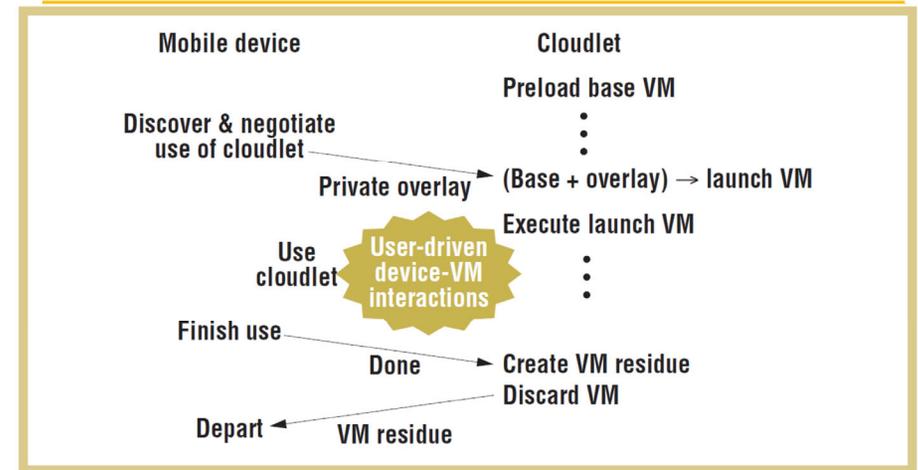


What is a Cloudlet?

- “Proximate computing infrastructure” that can be leveraged by mobile devices
- Or: Fixed infrastructure for transient use by mobile devices
- Some key differences with cloud computing:

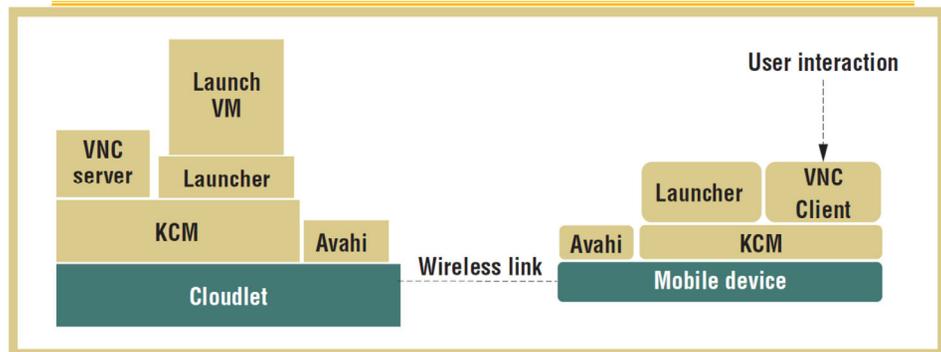
	Cloudlet	Cloud
State	Only soft state	Hard and soft state
Management	Self-managed; little to no professional attention	Professionally administered, 24/7 operator
Environment	“Datacenter in a box” at business premises	Machine room with power conditioning and cooling
Ownership	Decentralized ownership by local business	Centralized ownership by Amazon, Yahoo, etc.
Network	LAN latency/bandwidth	Internet latency/bandwidth
Sharing	Few users at a time	Hundreds to thousands of users at a time

On-the-fly configuration of Cloudlet



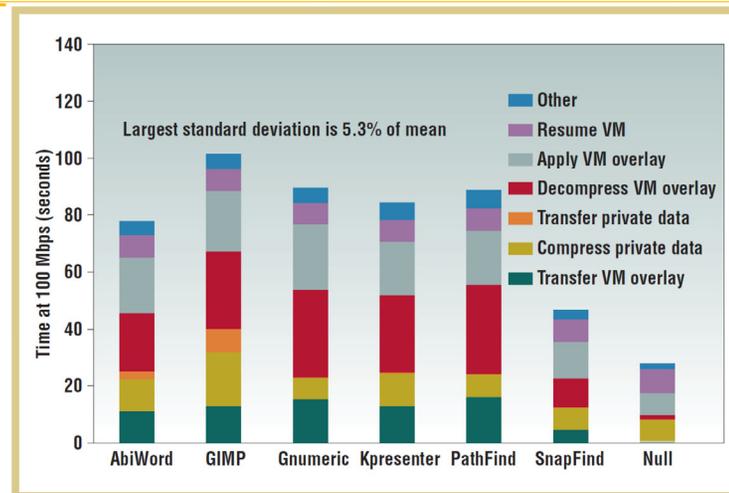
- Create overlay “delta” from base VM
- Launch by sending delta to local Cloudlet server

Prototype: “Kimberly”



- Workshop paper (which was reading #3 for today): “Transient Customization of Mobile Computing Infrastructure”
 - Start with standard (“base”) VM
 - Compute “residue” on top of base
 - Send ID of base and residue from mobile device ⇒ Nearby Cloudlet

Cost of launching VM with 100Mbps net BW



- Largest cost: decompress+apply overlay
- Are these reasonable times? (up to 1.5 minutes...)

Issues with Cloudlet Idea?

- Persistent storage?
 - Send “local storage volume” from mobile device to Cloudlet at startup
 - Return delta back to mobile device at end of computation
 - What is security of information? Is it encrypted? Do you give up encryption keys to Cloudlet?
- How to find and negotiate for computational resources?
 - Need device discovery services. Uses Avahi (or “Zeroconf”). Same protocol as Apple Bonjour.
 - Only works on same subnet or multicast-domain
 - How do you know that edge device has sufficient capabilities?
- Security (continued)
 - Edge devices notoriously insecure
 - Physical security is very tricky – how to prevent someone from walking up to device and dumping its memory to recover keys/data
 - Why do we trust random device in first place?
- What is the economic model? Who would deploy these devices? Why?

Are these good papers?

- What were the authors’ goals?
- What about the evaluation/metrics?
- Did they convince you that this was a good system/approach?
- Were there any red-flags?
- What mistakes did they make?
- Does the system/approach meet the “Test of Time” challenge?
- How would you review these papers today?