EECS 262a
Advanced Topics in Computer Systems
Lecture 20

VM Migration/VM Cloning
November 10th, 2014

John Kubiatowicz
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
- SnowFlock: Rapid Virtual Machine Cloning for Cloud Computing
  H. Andrés Lagar-Cavilla, Joseph A. Whitney, Adin Scannell, Philip Patchin, Stephen M. Rumble, Eyal de Lara, Michael Brudno,and M. Satyanarayana. Appears in *Proceedings of the European Professional Society on Computer Systems Conference* (EuroSys), 2009

- Today: explore value of leveraging the VMM interface for new properties (migration and cloning), many others as well including debugging and reliability
- Thoughts?

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

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

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

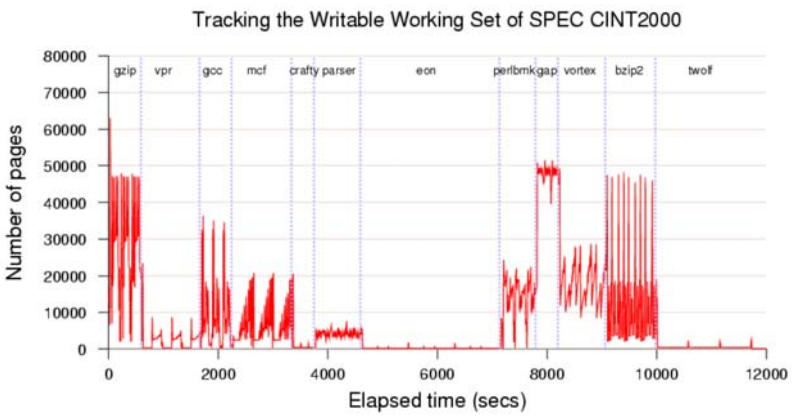
Live Migration Approach (II)

- 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)

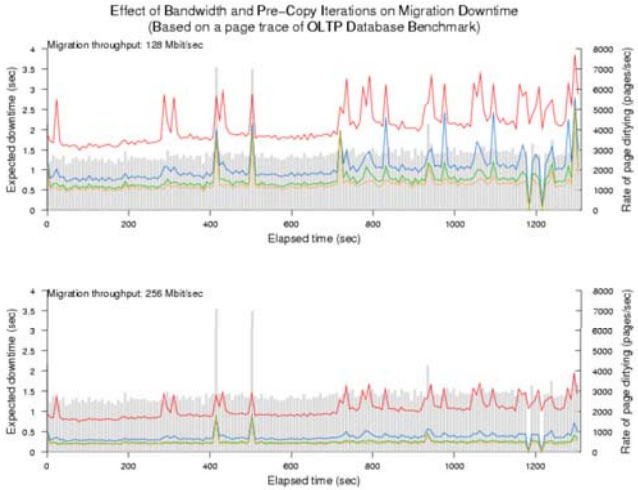
Tracking the Writable Working Set

- 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

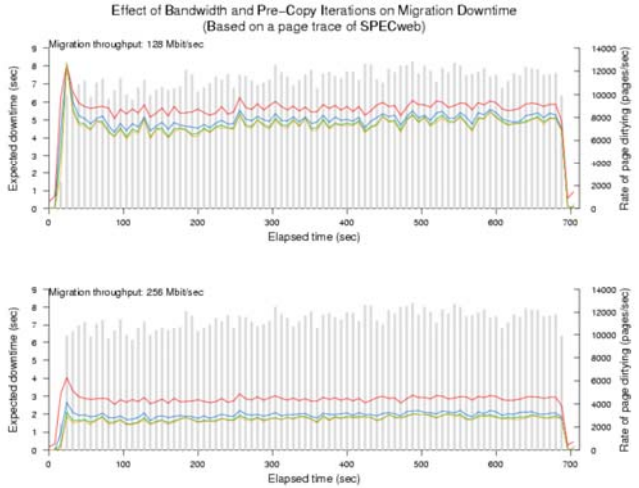


OLTP Database



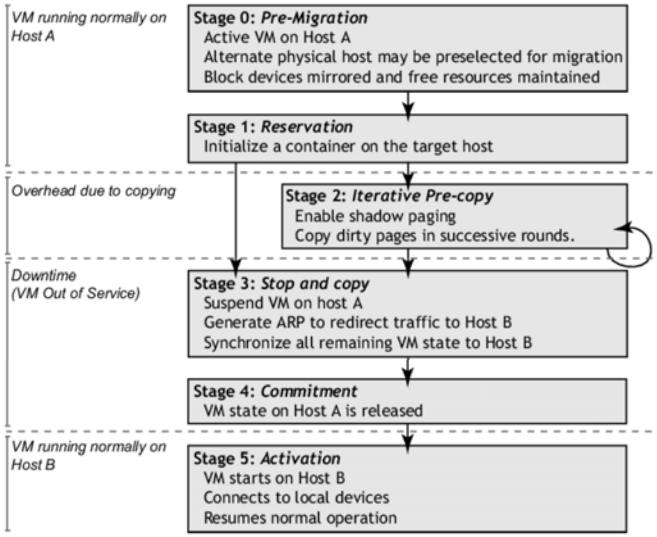
- Compare with stop-and-copy:
  - 32 seconds (128Mbit/sec) or 16seconds (256Mbit/sec)

SPECweb



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

Design Overview



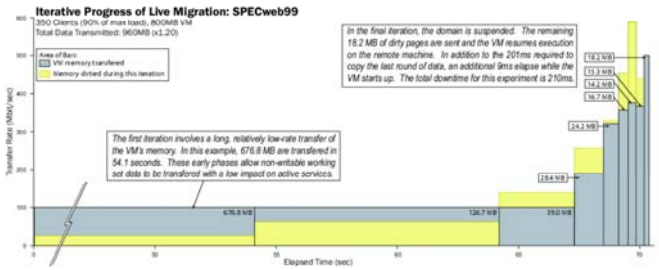
Handling Local Resources

- 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

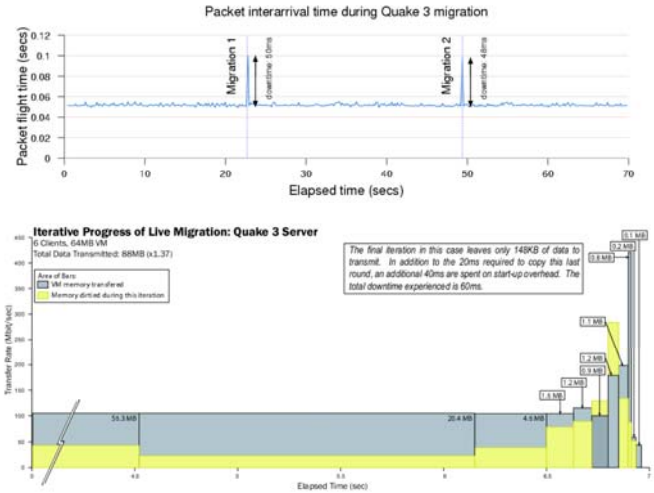
Types of Live Migration

- 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

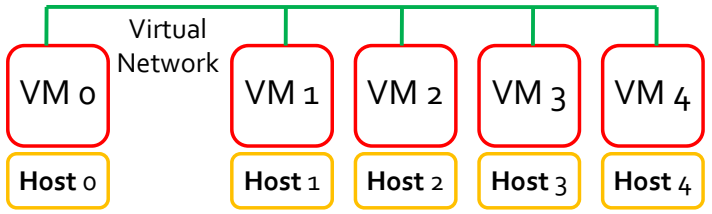
- True "Utility Computing"
  - Illusion of infinite machines
  - Many, many users
  - Many, many applications
  - Virtualization is key
- Need to scale bursty, dynamic applications
  - Graphics render
  - DNA search
  - Quant finance
  - …

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: 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
if child:
work on i[th] slice of data

Parallel Computation

if more load:
fork extra workers
if load is low:
dealloc excess workers

Load-balancing Server

trusted code
fork
if child:
untrusted code

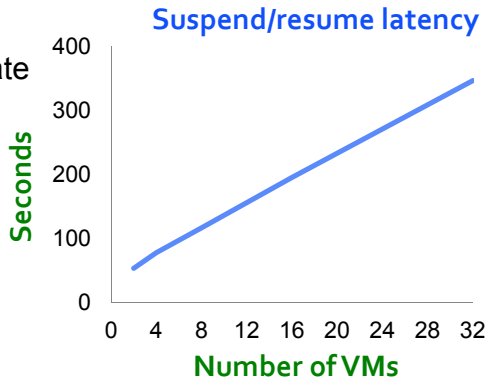
Sandboxing

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

Opportunistic Computation

VM Fork Challenge – Same as Migration!

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

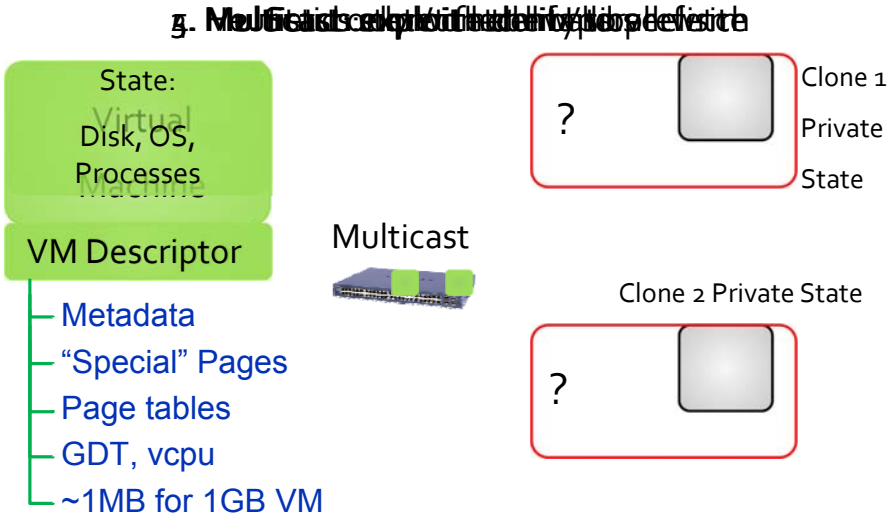


- Same fundamental bottleneck issues as VM Migration – shared I/O resources: host and network

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

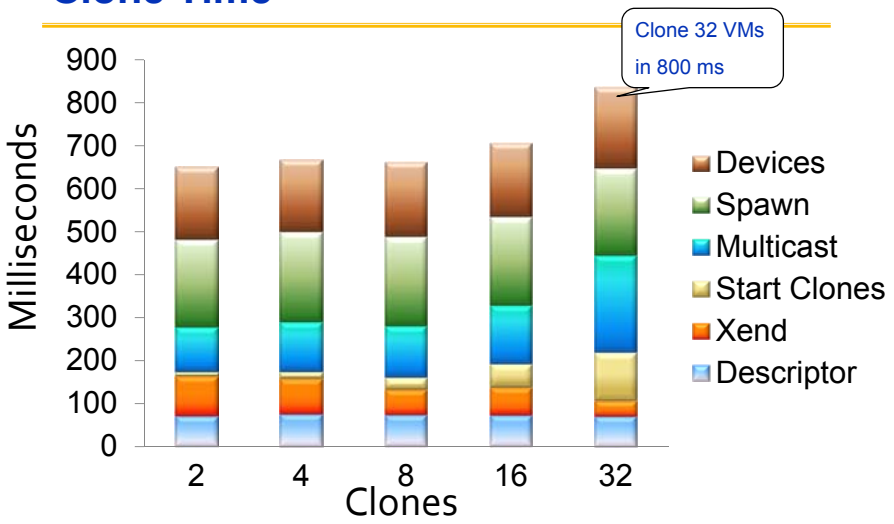
SnowFlock Secret Sauce



Why SnowFlock is Fast

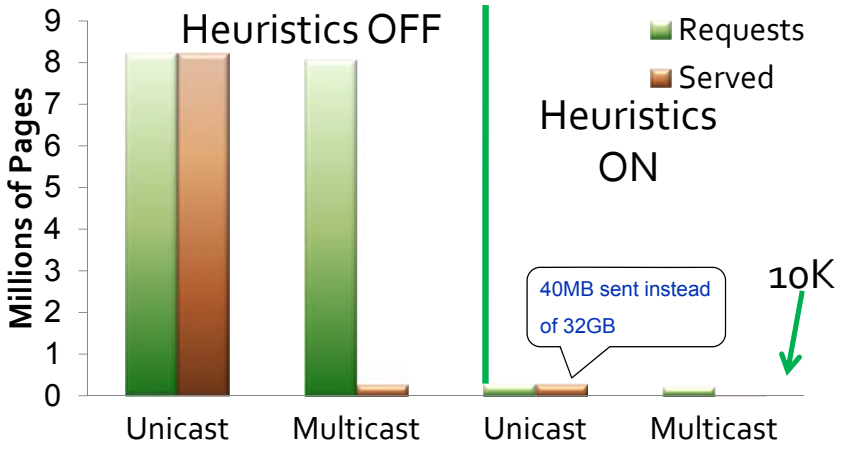
- Start only with the basics
- Send only what you **really need**
- Leverage IP Multicast
  - Network hardware parallelism
  - Shared prefetching: exploit **locality patterns**
- Heuristics
  - Don't send if it will be overwritten
  - Malloc: exploit **clones generating new state**

Clone Time



Scalable Cloning: Roughly Constant

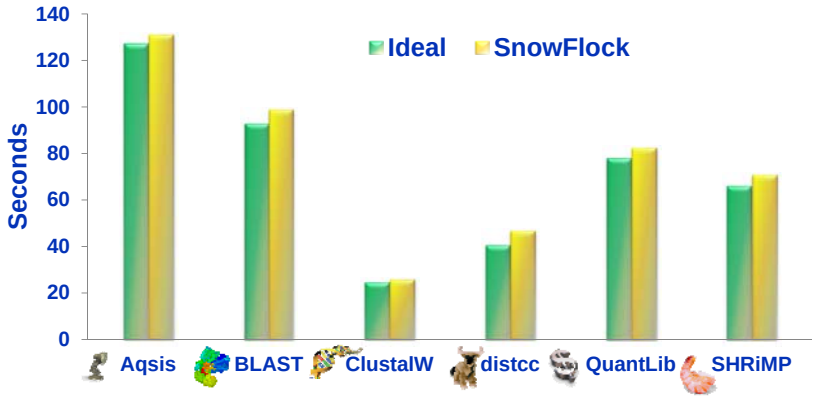
Page Fetching, SHRiMP 32 Clones 1GB



Application Evaluation

- Embarrassingly parallel
  - 32 hosts x 4 processors
- CPU-intensive
- Internet server
  - Respond in seconds
- Bioinformatics
- Quantitative Finance
- Rendering

Application Run Times



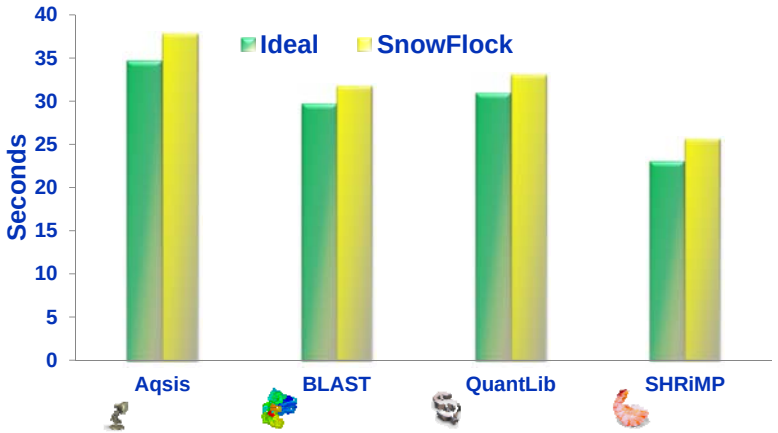
≤ 7% Runtime Overhead

~ 5 seconds

Throwing Everything At It

- Four concurrent sets of VMs
  - BLAST, SHRiMP, QuantLib, Aqsis
- Cycling five times
  - Clone, do task, join
- Shorter tasks
  - Range of 25-40 seconds: interactive service
- Evil allocation

Throwing Everything At It



Fork. Process 128 x 100% CPU. Disappear.

30 Seconds

Summary: SnowFlock In One Slide

- VM fork: natural intuitive semantics
- The cloud bottleneck is the IO
  - Clones need little parent state
  - Generate their own state
  - Exhibit common locality patterns
- No more over-provisioning (pre-alloc, idle VMs, migration, …)
  - Sub-second cloning time
  - Negligible runtime overhead
- Scalable: experiments with 128 processors

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?