Post-Copy Based Live Virtual Machine Migration Using Pre-Paging And Dynamic Self-Ballooning

State University of New York at Binghamton

Michael R. Hines & Kartik Gopalan

Talk Outline

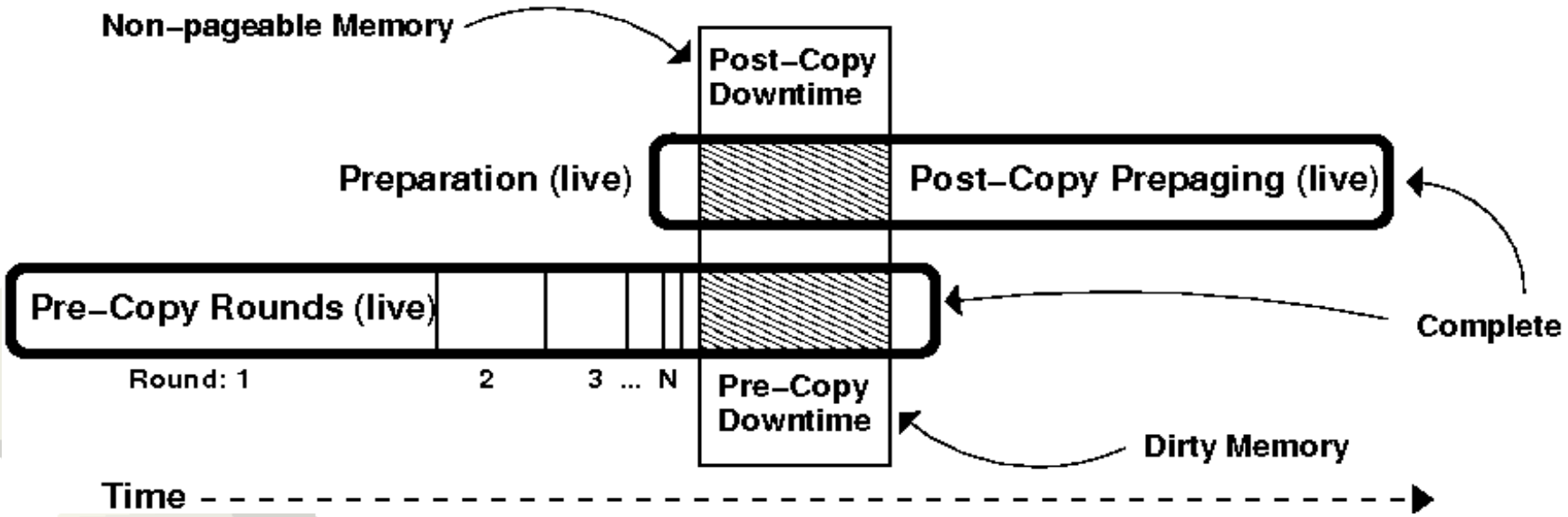
- **Motivation:**
  - Xen Primer & Migration Primer
  - Differences w/ Processes Migration
  - Performance Goals & Metrics
- **Solutions:**
  - Design Strategies
  - Pre-Paging Algorithm
  - Self-Ballooning Algorithm
  - Page-fault Detection Algorithms
- **Performance Results:**
  - Amortized Analysis (4 graphs)
  - Real Applications (4 graphs)

Xen / Migration Primer

- **Types of Migration:**
  - Process Migration:
    - File handles, socket addressing IPC, memory mappings
  - System Migration:
    - primarily the virtual network
    - much more transparent
  - We use the Xen Hypervisor
- **Uses:**
  - Consolidation, Maintenance
  - Power Savings, Load Distribution
- **Migration Considerations:**
  - CPU state + Memory state
  - Live or Non-Live
  - Residual Dependencies
  - Primary Difficulty: **Memory State**

Live, Residual-Free Migration

- **Sequence of Events:**
  - Preparation Time
  - Down Time: CPU transfer, No execution progress made
  - Resume Time

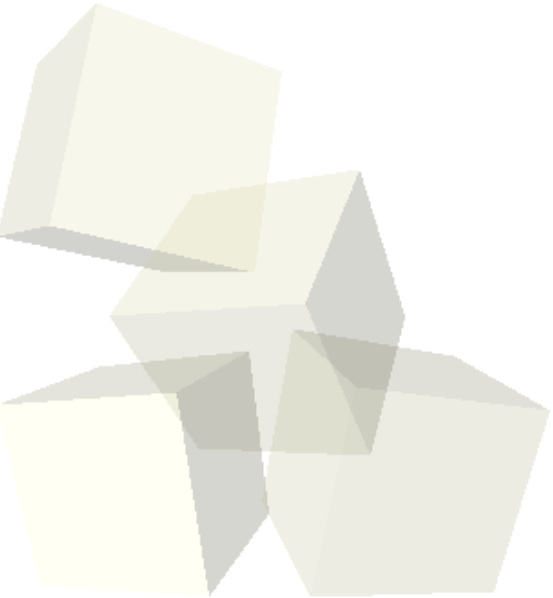


  - You can transfer memory before, or after....

Design Goals

- **Pre-Copy: (preparation time, non-deterministic):**
  - Memory First (dirty pages), CPU Last
  - Problems: tracking memory dirtying
- **Post-Copy: (resume time, deterministic):**
  - CPU First, Memory Last (page-faults)
- **Problems during Resume-Time:**
  - System-level Page-Fault Detection
  - System-level Residual Dependencies (pre-paging)
  - Eliminate Free Pages (ballooning)
- **Common Performance Metrics:**
  - Pages Transferred, Page-Faults
  - Downtime, Total Time
  - CPU and Network Overheads

Solutions



Copy Strategies

- **Demand Paging Only:**
  - Page-faults, residual pages
- **Basic Post-Copy:**
  - Called “Flushing”
  - Perform address-space flush
- **Pre-Paging:**
  - Pre-paging is a general caching strategy
  - Flush intelligently
  - ``Bubbling” Algorithm
- **Hybrid Scheme:**
  - Perform one pre-copy pass
  - Then, execute Post-Copy
- **This presentation:**
  - Pre-paging only

The Bubbling Algorithm

```
1. let N        <- total # of guest VM pages
2. let page[N]   <- set of all guest VM pages
3. let bitmap[N] <- all zeroes
4. let fault     <- 0

5. PrePage (Guest VM)
6.   let count <- 0
7.   while (count < N)
8.     if fault is non-zero
9.       set count <- fault
10.      continue
11.    if bitmap[count] is clear
12.      set bitmap[count] <- 1
14.      bubble <- max(0, fault - (count - fault))
15.      queue page[count]  for transmission
16.      queue page[bubble] for transmission
17.    count++
```

- **Total time is linear**
- **Each page sent once**
- **Kernel concurrency**

```
18. PageFault (Guest-page X)
19.   if bitmap[X] is zero
20.     set bitmap[X] <- 1
21.     set fault <- X   // shift pre-paging window
22.     transmit page[X] immediately

23. PreCopy (Guest VM)
24.   while bitmap contains zeroes
25.     PrePage (Guest VM)
```

Basic idea:

Two threads of control:

- One to service page-faults

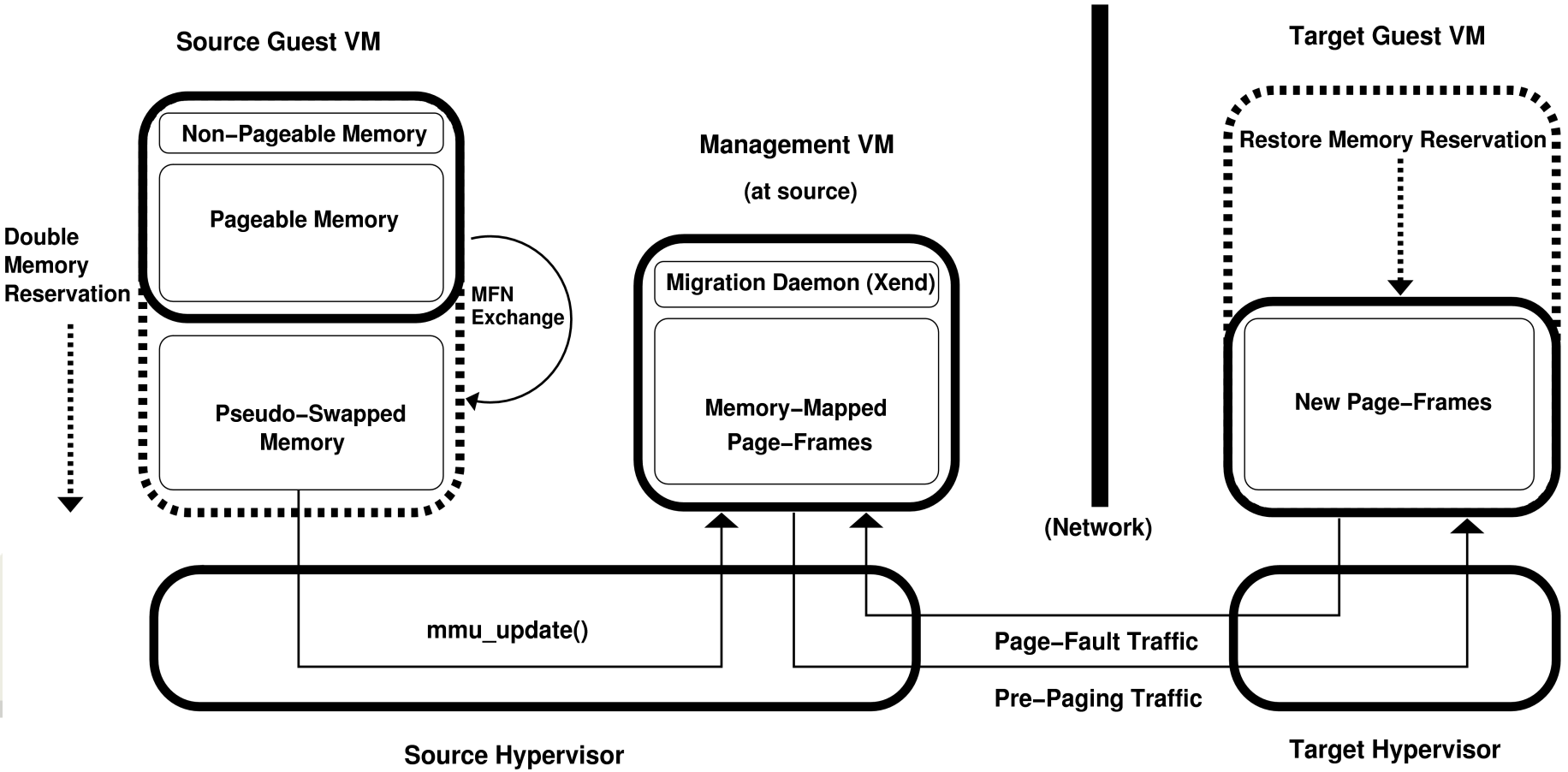
- One to perform pre-paging

When they overlap, make adjustments

Page-Fault Detection (1)

- **Page Tracking:**
  - Fastest Method
  - Mark resident PTEs **not present**
    - Just Before downtime
  - Real CPU exceptions
- **Pseudo Paging:**
  - Easy Implementation, Leverages prior work.
  - Treat source node as “swap device”
  - Non-pageable memory overhead
- **Shadow-Paging:**
  - Compromised between previous methods
  - Extra level of indirection:
    - copy of read-only page tables
  - Already used heavily in modern hypervisors
  - Trap into post-copy system @ target

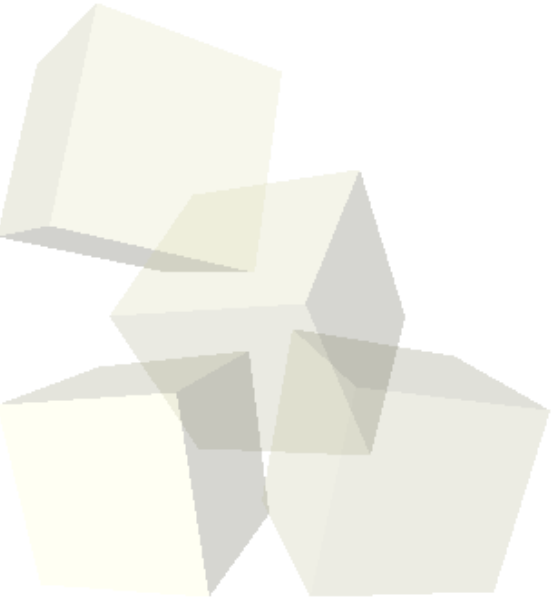
Page-Fault Detection (2)



Self-Ballooning

- **What is Ballooning?**
  - Change OS physical memory @ runtime
  - What is it used for?
- **The Free Memory Problem:**
  - Free pages: will get RE-allocated soon (COW)
  - Clean Pages: will get RE-dirtied soon
  - How do we eliminating them?
    - → Free Pages: Balloon, Clean Pages: Hybrid
  - Free page elimination mandatory for **Pre-Copy**
- **Our Ballooning Algorithm:**
  - Inflate every 5 seconds
  - Deflate during memory pressure
    - → How do you do that transparently?
  - Stop inflation during memory pressure

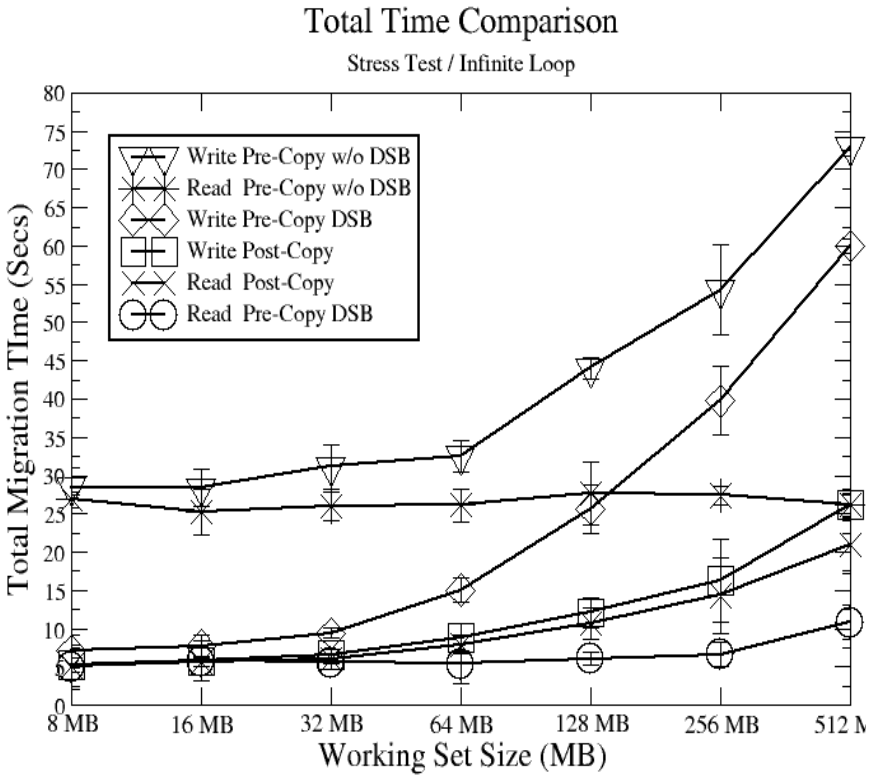
Evaluation

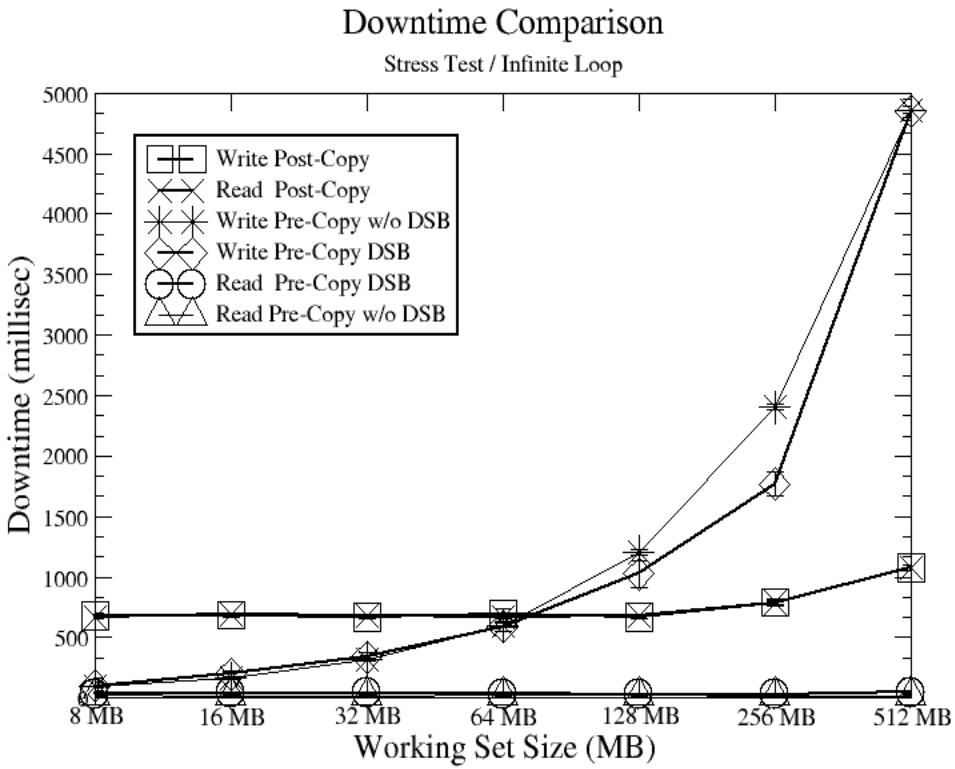


Lab Environment

- **2-node Xen testbed**
  - 4GB of DRAM per node
  - 2.8 Ghz, Dual-Core
  - Xen: 3.2.1
  - Linux: 2.6.18.8
  - Gigabit ethernet
- **Two types of experiments:**
  - Amortized Analysis: (6 plots, 4 graphs)
    - Diabolical Program
  - Real Applications: (4 plots, 4 graphs)
    - SpecWEB 2005 (huge, needed 6 nodes)
    - NetPerf, BitTorrent, and Kernel Compile
- **Metrics Considered:**
  - Total Time, Downtime,
    Pages Transferred, Page-faults

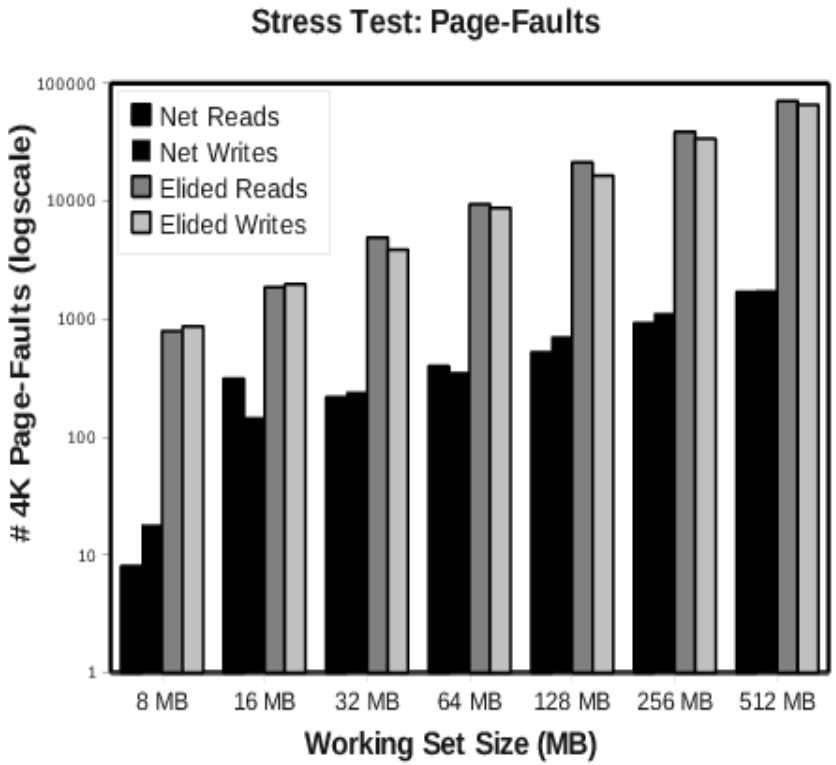
Amortized Analysis (1)

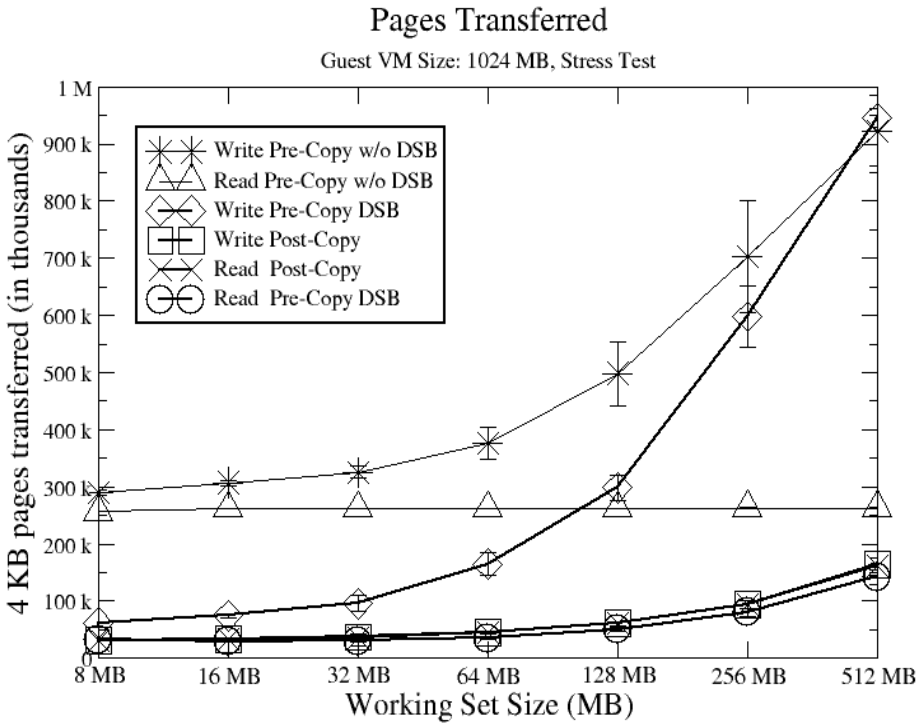




- **Total Time:**
  - Post-copy reductions are high. DSB high.
- **Downtime:**
  - Post-copy remains low.
    Pre-copy continues to increase.

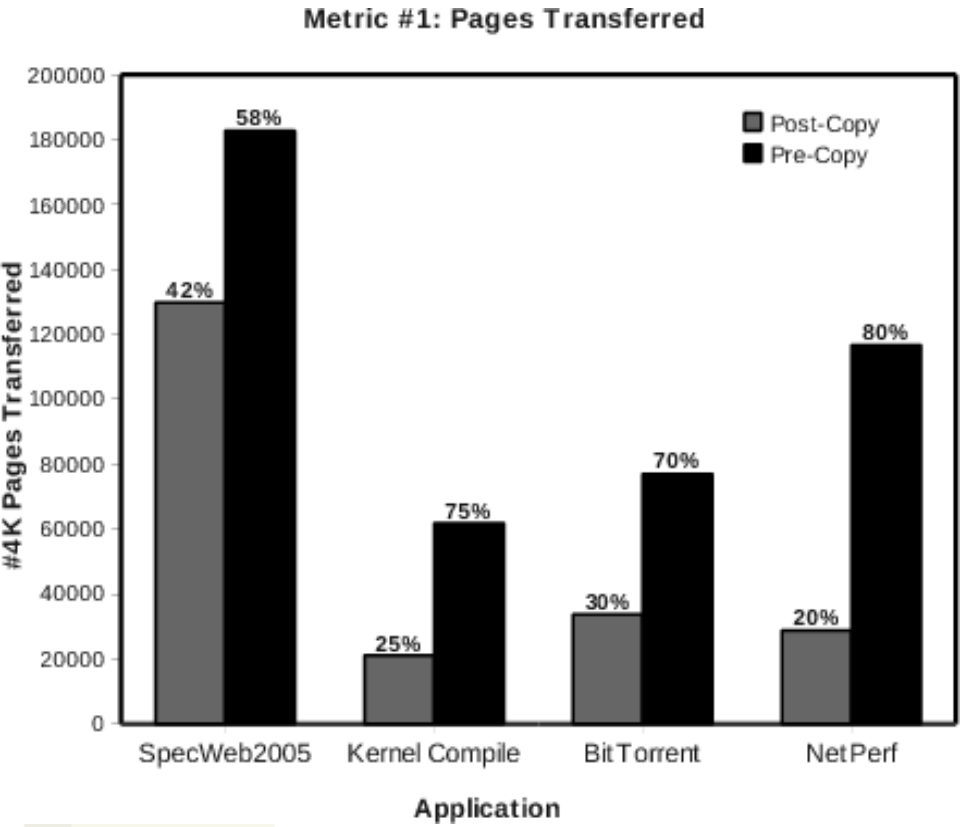
Amortized Analysis (2)

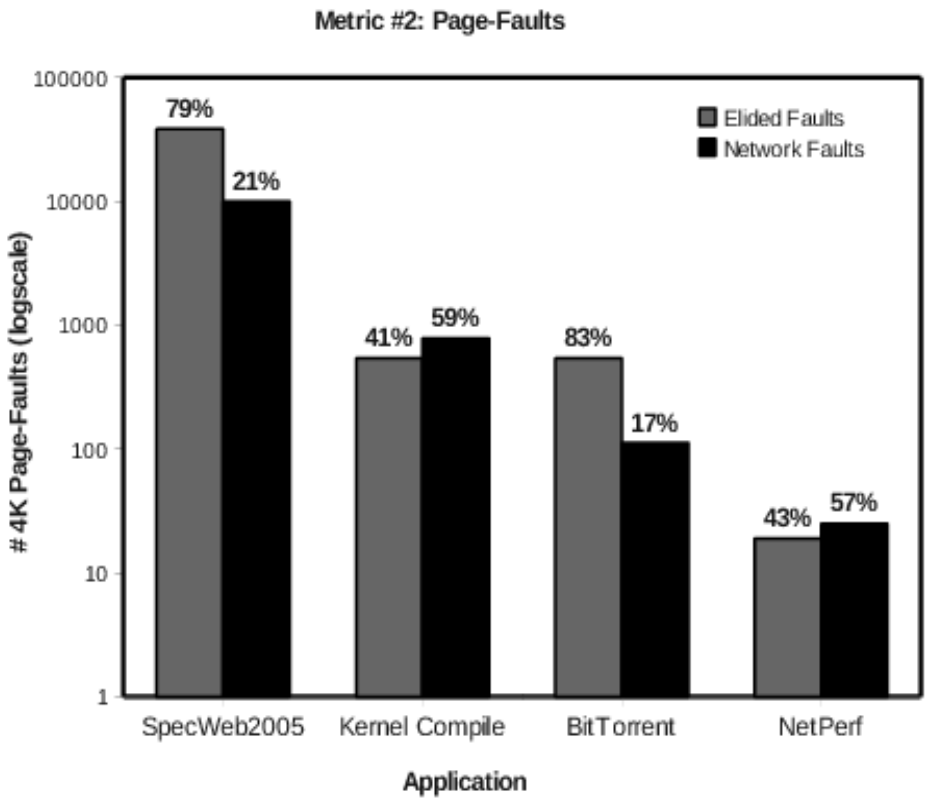




- **Page-Faults:**
  - Even in worst-case, pre-paging succeeds 50%
- **Pages Transferred:**
  - Similar to previous metrics. DSB high.
  - Post-copy improvements up to 75%.

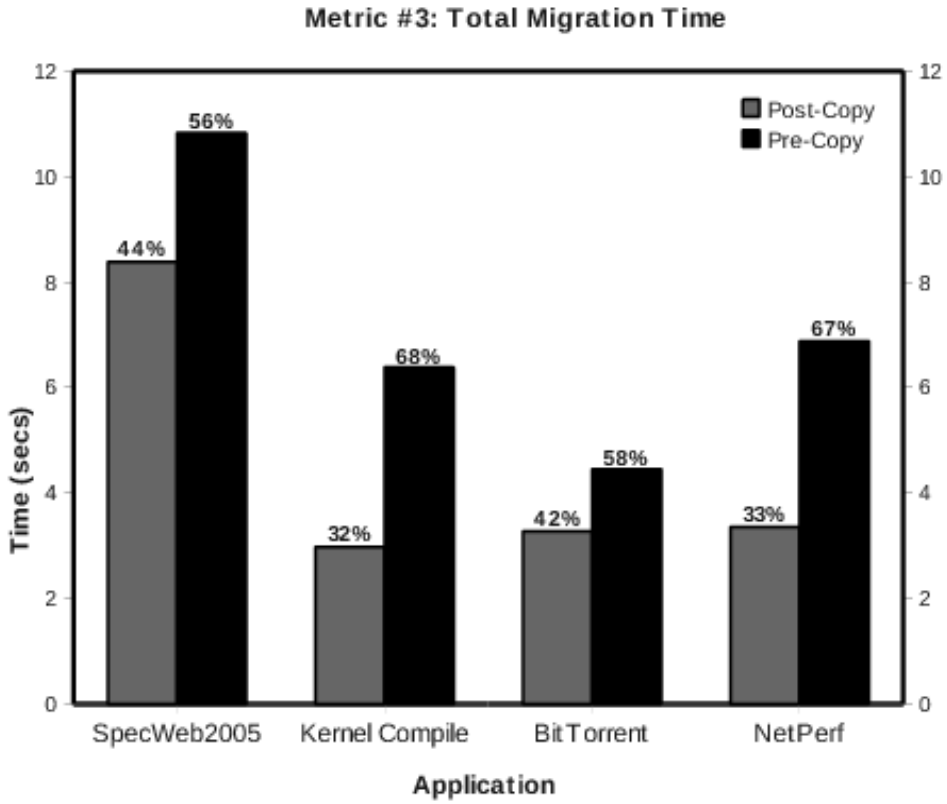
Applications (1)

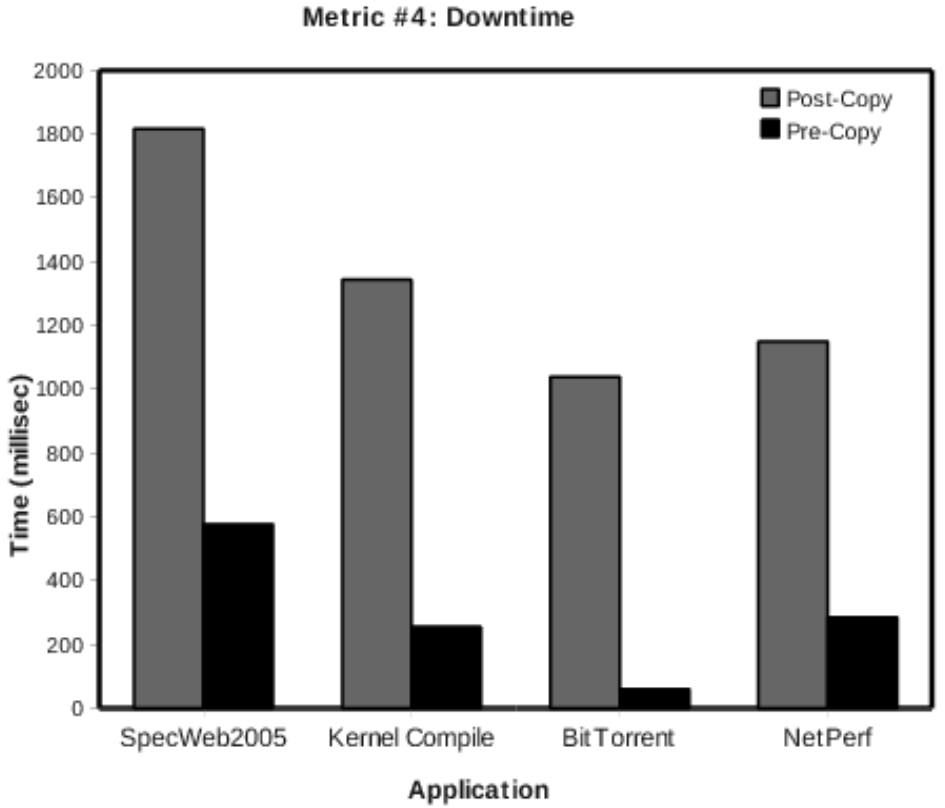




- **Pages Transferred:**
  - Minimizations up 75%.
- **Page-faults:**
  - Minimization up to 80%. Others also significant.

Applications (2)





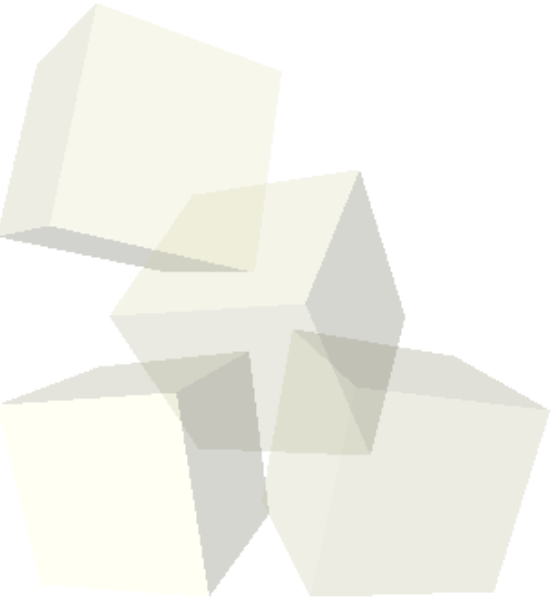
- **Total Time:**
  - More than 50% reduction
- **Downtime:**
  - Pre-copy wins because of page-fault detection

Conclusions

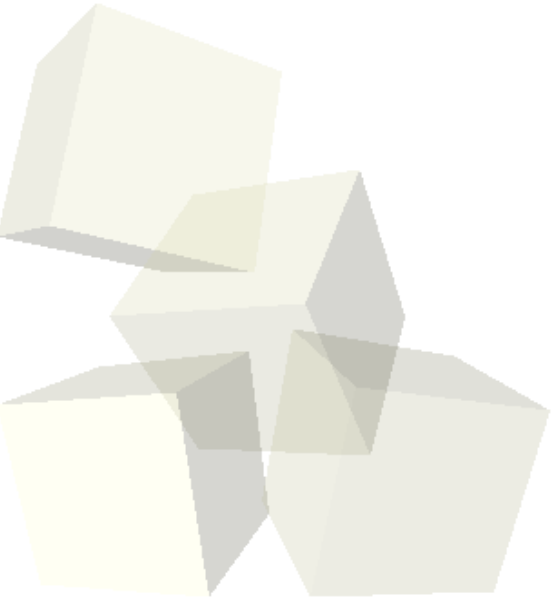
- **Pre-paging**
  - Effective at hiding page-fault latency
  - Can be improved with history / added knowledge
- **Ballooning**
  - Normalizes both approches
  - Also improves the pre-copy system
- **Post-Copy**
  - Very effective at minimizing pages transferred
  - Remains transparent
  - Preserves all of the existing performance metrics
    - ➔ Downtime affected by page-fault detection problems
- **Future Work:**
  - Better page-fault detection
  - More applications of ballooning
  - Pseudo-physical Address Space Expansion
  - Address Reliability concerns

Thank you!

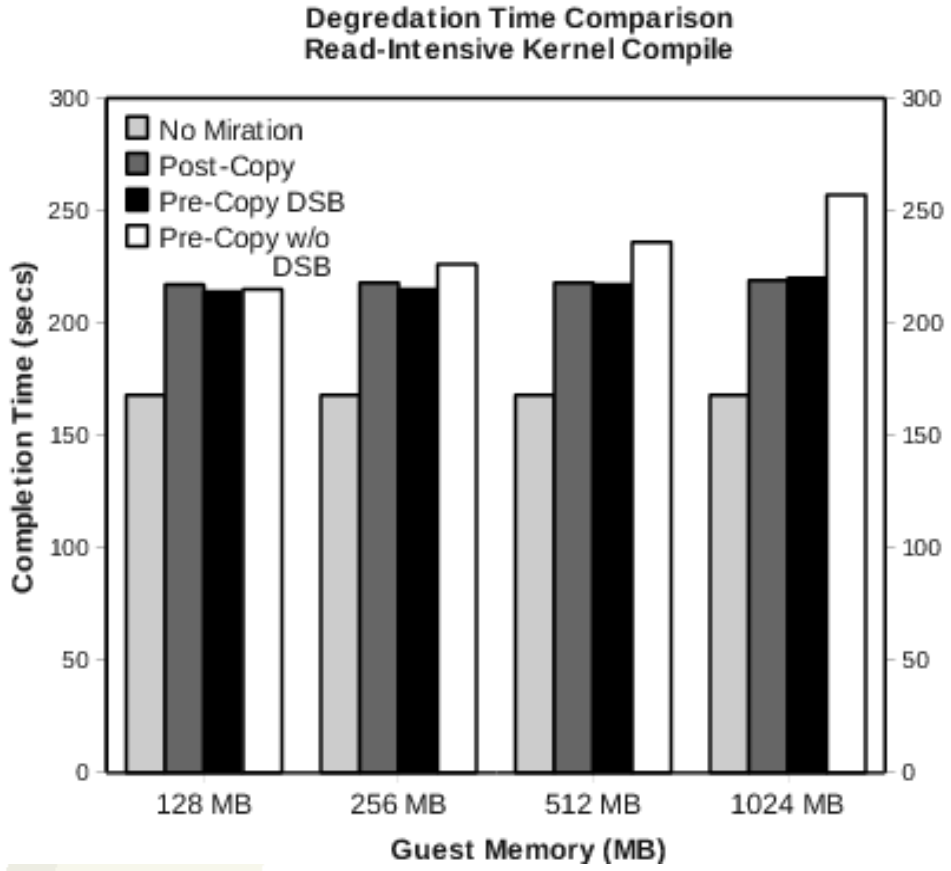
Questions?

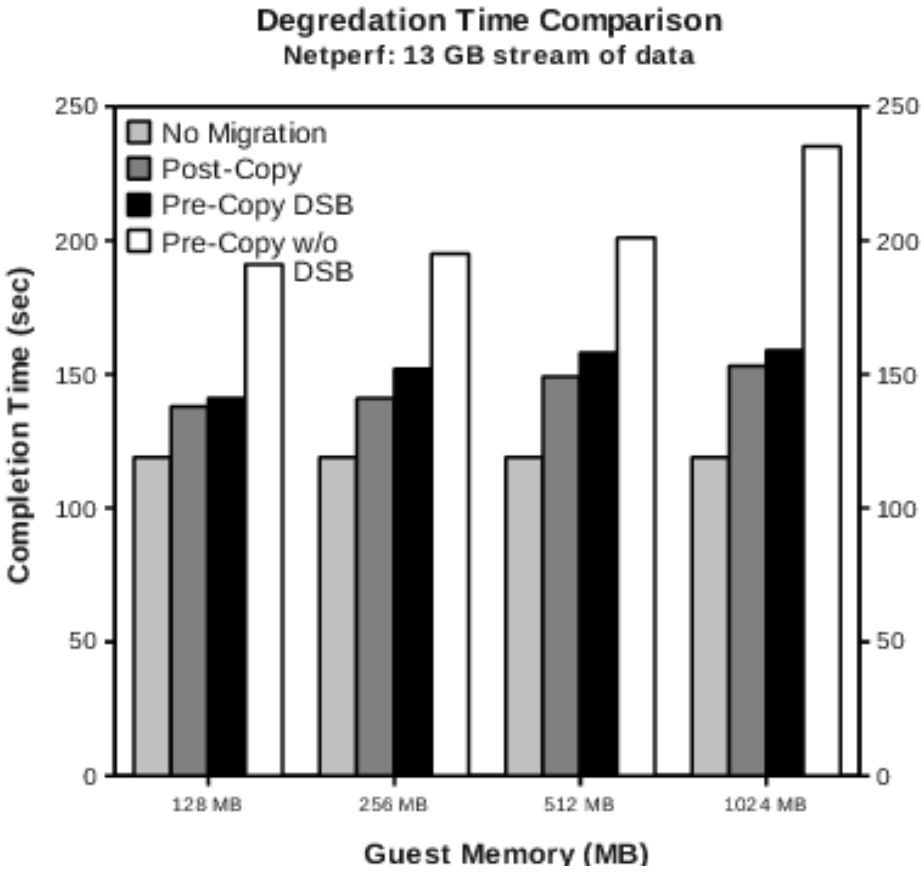


Backup Slides



Degradation Time





- **Kernel Compile:**
  - Read-bound. Degredation unchanged.
- **NetPerf:**
  - More write-bound. Degredation better for post-copy.
- **DSB:** decativation degrades significantly