

# Satori: Enlightened Page Sharing

Grzegorz Milos, Derek G. Murray,  
Steven Hand, Michael Fetterman



UNIVERSITY OF  
CAMBRIDGE

# Outline

- Why share pages?
- Why use enlightenments?
- Unique selling propositions
  - Duplication detection
  - Savings distribution
  - Fault handling
- Implementation
- Performance results

# Page sharing

- Identical O Ss share lots of identical data
  - Program binaries
  - Shared libraries
  - Configuration files
- More memory is always useful
  - Reduce paging
  - Increase page cache size
  - Less I/O , better performance !

# Enlightenment

... or “paravirtualisation”

... or “changing the OS”

- Use information from the OS to guide sharing
  - Reduces the overhead
- Avoid pathological behaviour
  - Double paging
  - Host demand paging
- Proof-of-concept on Linux

# USP 1 : duplicate detection

- Satori monitors block reads
  - Hash data when read, share pages if identical
  - Block reads from common substrate disks
- Other systems scan memory periodically
  - Rate limited to prevent overhead
  - Can miss short-lived sharing opportunities

# USP 2: savings distribution

- Want to encourage sharing
  - The more you share the more memory you get
  - Each VM gets a *sharing entitlement* based on how many pages it shares
- Deflate the balloon to increase allowance

# U SP 3 : fault handling

- What happens when sharing is broken?
  - Need some more memory
  - ... but it's in use
  - Typically use host paging
- Modify the OS to nominate “volatile” pages
  - Clean buffer cache pages
  - Hypervisor can reassign them without warning
  - “Repayment FIFO” based on IBM's CMM

# Implementation

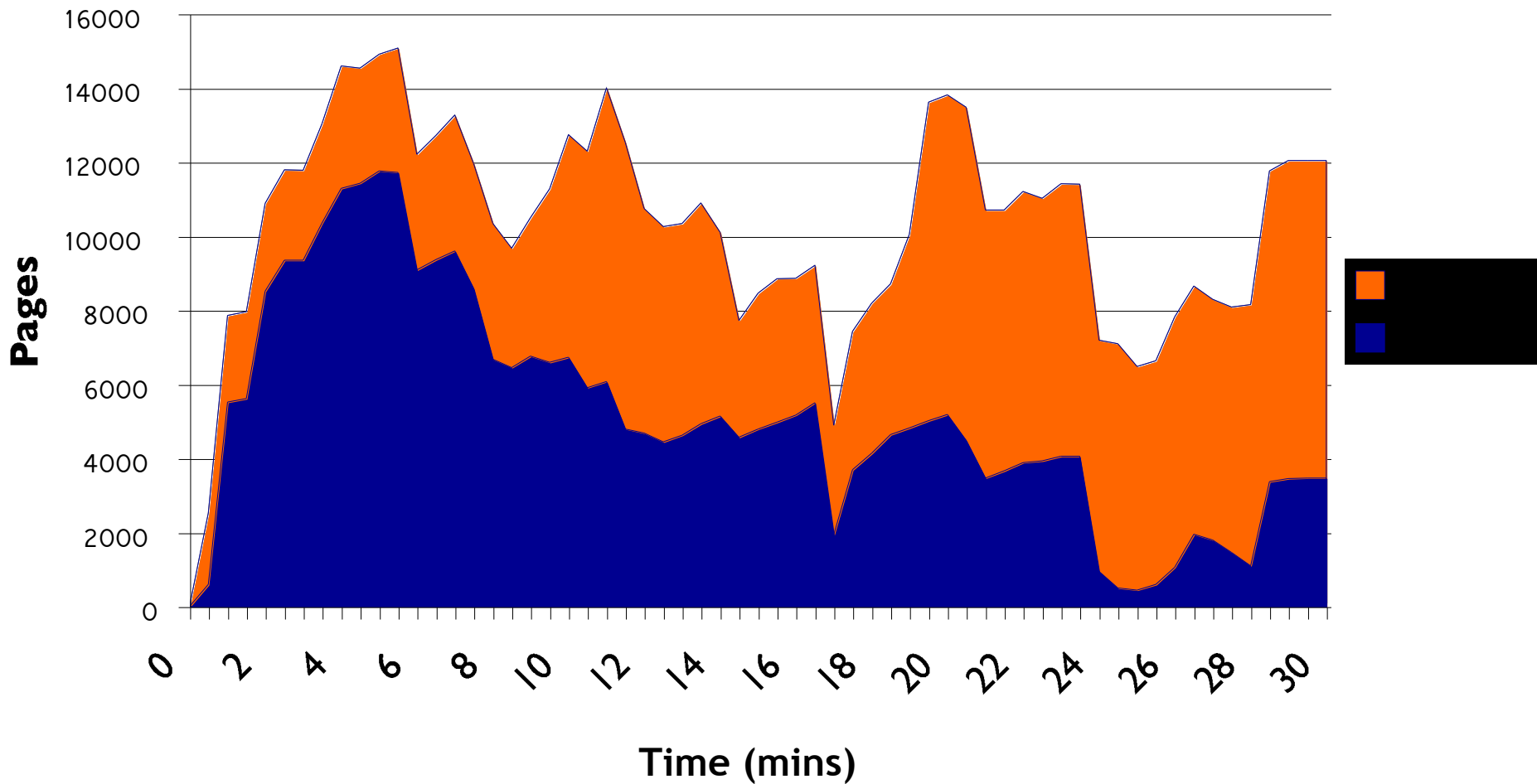
- Hypervisor (5351 LOC)
  - Based on Xen 3.1 with PV shadow PTs
  - Added sharing-related hypercalls
- Dom0 (3894 LOC)
  - Modified blk\_tap to perform hash lookup
- DomU (2306 LOC)
  - Ported IBM's CMM to Xen/x86

# O verheads

- W orst-case: large-chunk sequential reads
  - O verhead of hashing: 0.2%
  - O verhead of hashing + IPC : 34.8%
- “Realistic” benchmark (kernel compile )
  - W ithout Satori: 780 seconds
  - Satori enabled: 779 seconds

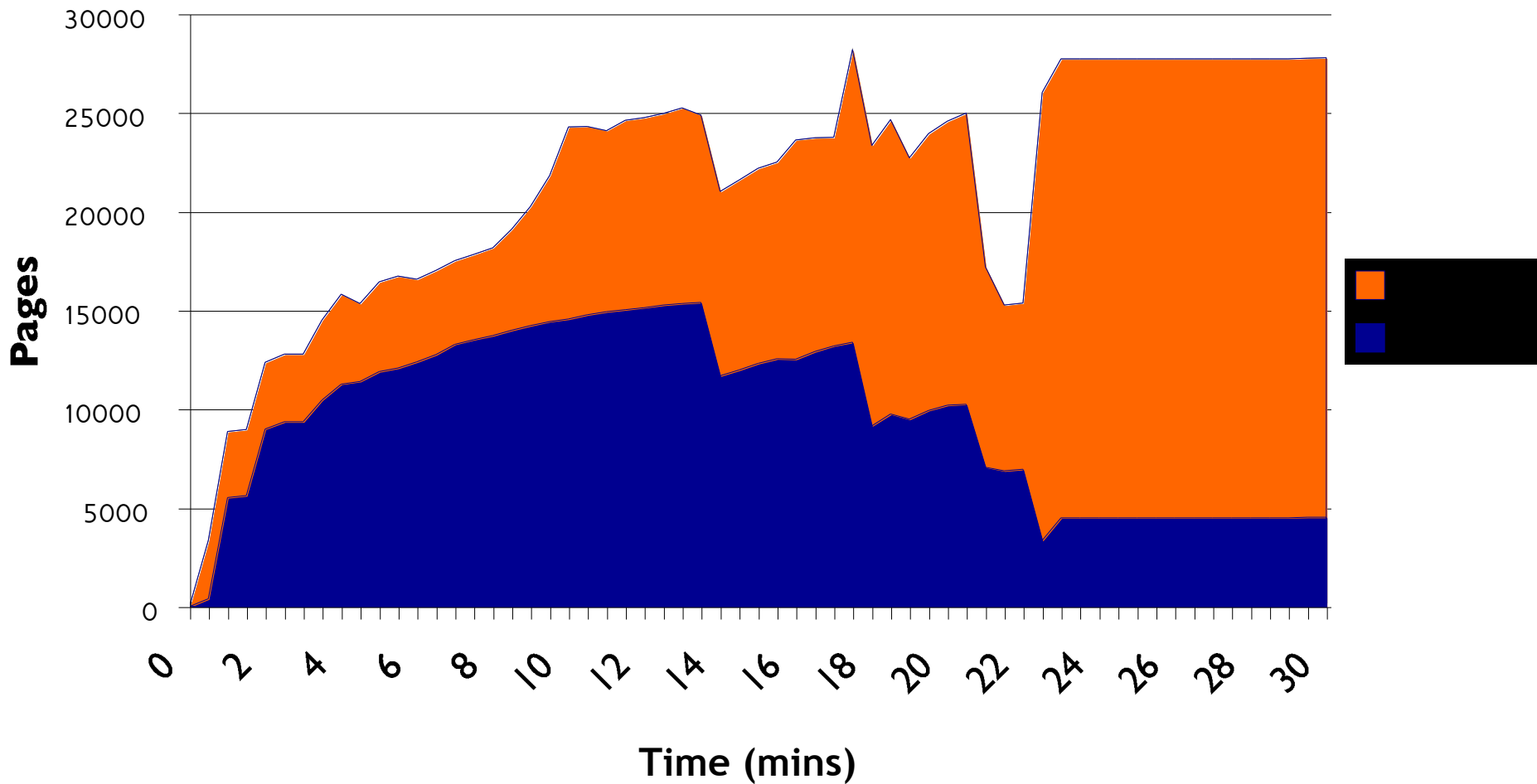
# Effectiveness

## Kernel Compilation (256MB)



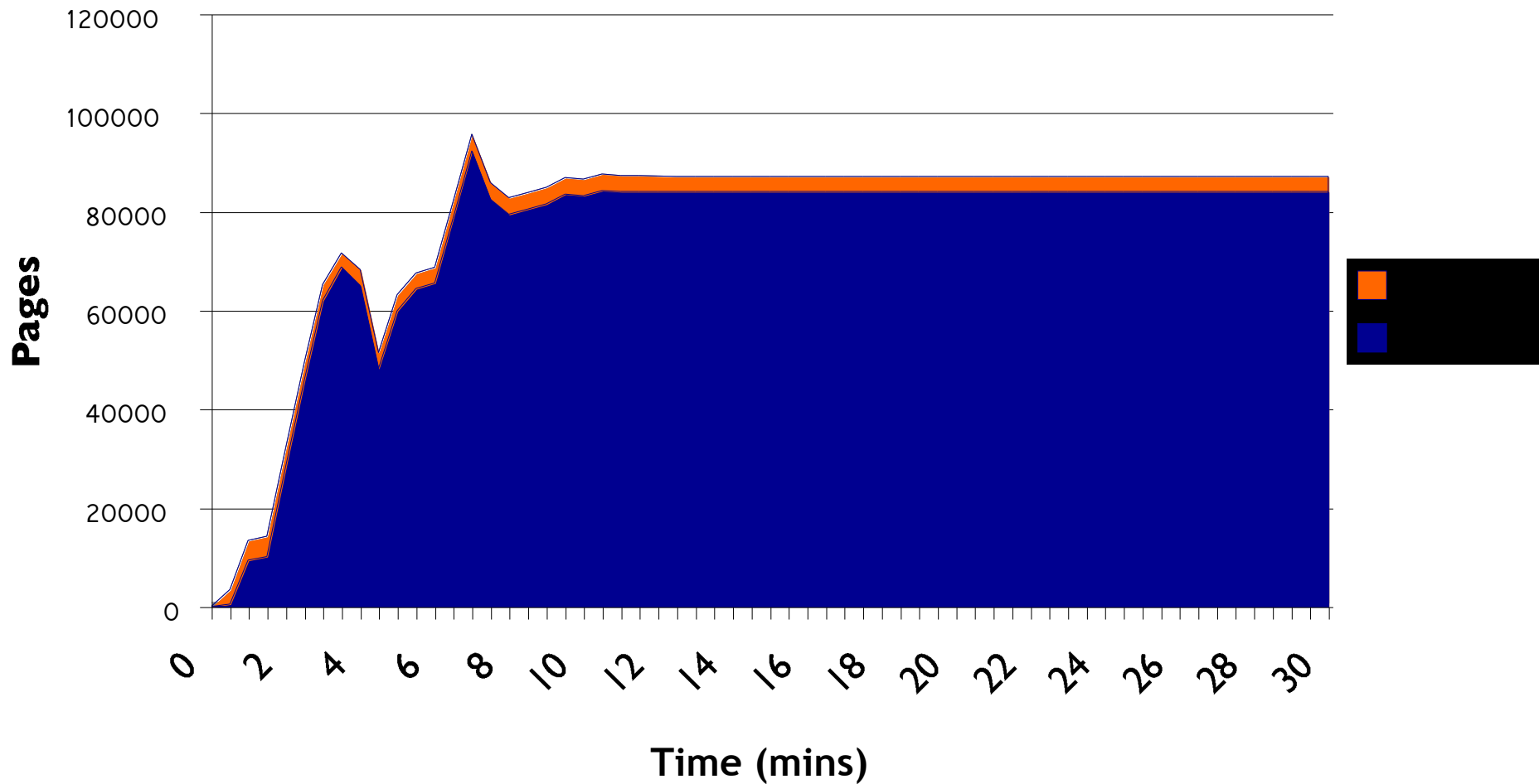
# Effectiveness

## Kernel Compilation (512MB)



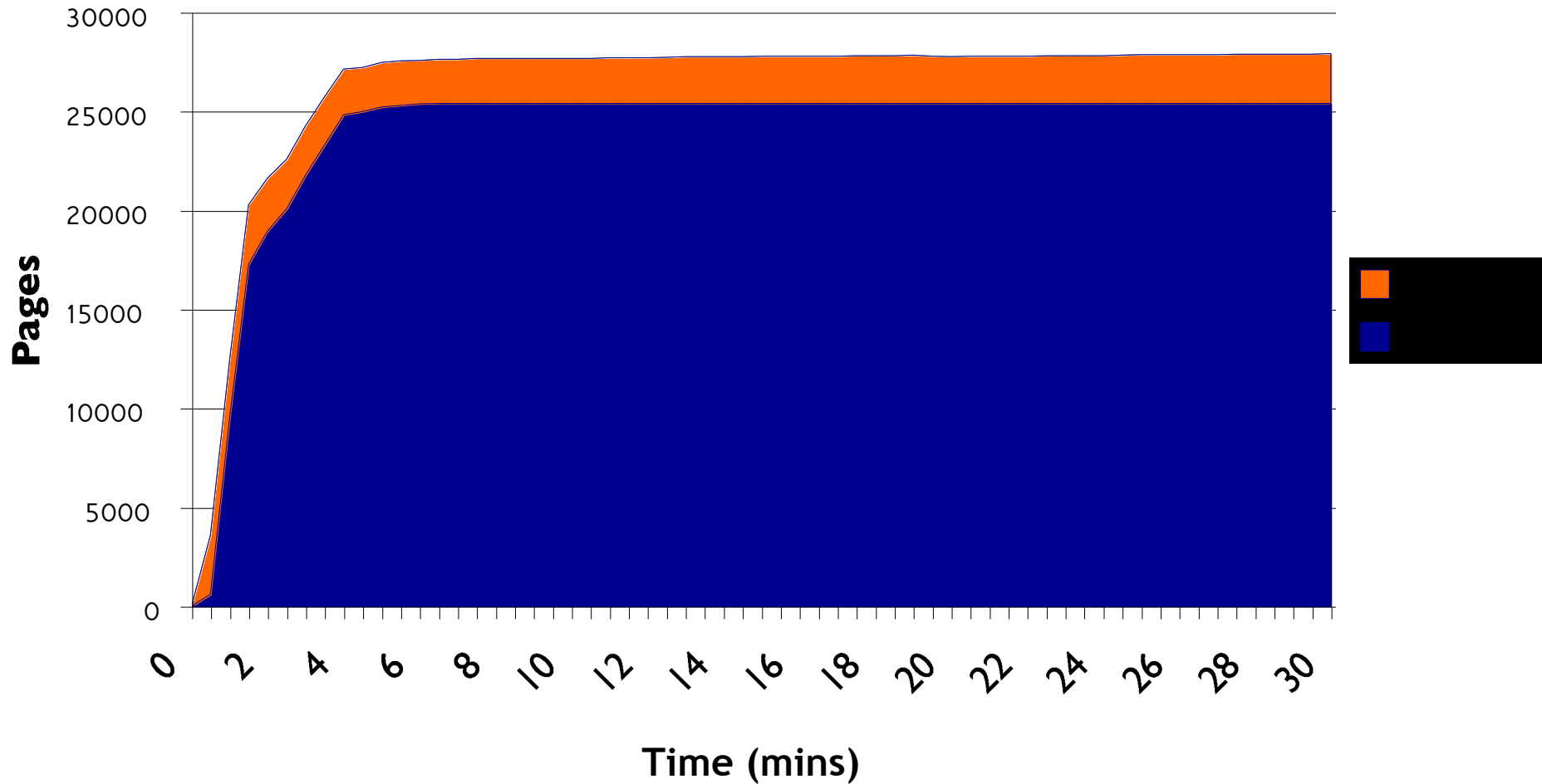
# Effectiveness

httperf



# Effectiveness

## RUBiS



# Conclusions

- More memory means more performance
- Page caches are useful sources of sharing
- Enlightening the OS avoids host paging
- Satori is efficient and effective!

Q u e s t i o n s ?

# Satori: Enlightened Page Sharing

Grzegorz Milos, Derek G. Murray,  
Steven Hand, Michael Fetterman



## Outline

- Why share pages?
- Why use enlightenments?
- Unique selling propositions
  - Duplication detection
  - Savings distribution
  - Fault handling
- Implementation
- Performance results

## Page sharing

- Identical O Ss share lots of identical data
  - Program binaries
  - Shared libraries
  - Configuration files
- More memory is always useful
  - Reduce paging
  - Increase page cache size
  - Less I/O , better performance !

# Enlightenment

... or “paravirtualisation”

... or “changing the OS”

- Use information from the OS to guide sharing
  - Reduces the overhead
- Avoid pathological behaviour
  - Double paging
  - Host demand paging
- Proof-of-concept on Linux

## USP 1: duplicate detection

- Satori monitors block reads
  - Hash data when read, share pages if identical
  - Block reads from common substrate disks
- Other systems scan memory periodically
  - Rate limited to prevent overhead
  - Can miss short-lived sharing opportunities

## USP 2: savings distribution

- Want to encourage sharing
  - The more you share the more memory you get
  - Each VM gets a *sharing entitlement* based on how many pages it shares
- Deflate the balloon to increase allowance

## U SP 3: fault handling

- What happens when sharing is broken?
  - Need some more memory
  - ... but it's in use
  - Typically use host paging
- Modify the OS to nominate “volatile” pages
  - Clean buffer cache pages
  - Hypervisor can reassign them without warning
  - “Repayment FIFO” based on IBM's CMM

# Implementation

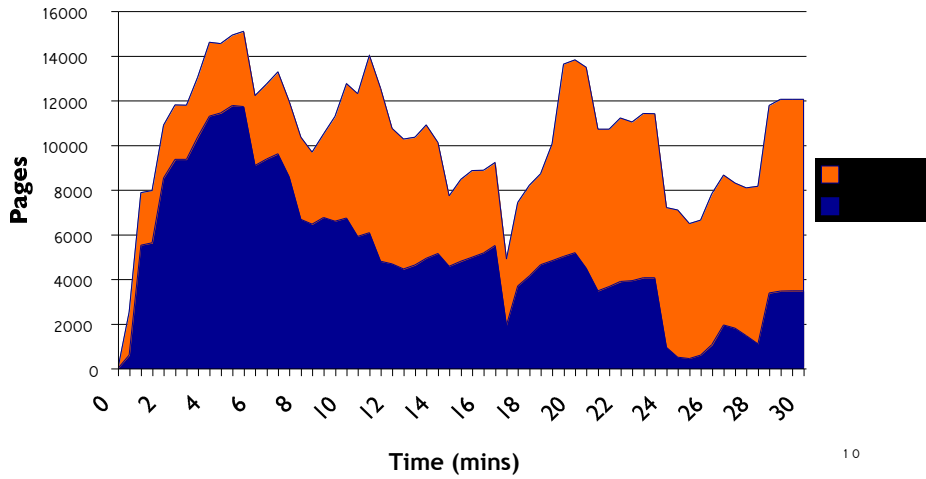
- Hypervisor (5351 LOC)
  - Based on Xen 3.1 with PV shadow PTs
  - Added sharing-related hypercalls
- Dom0 (3894 LOC)
  - Modified blktap to perform hash lookup
- DomU (2306 LOC)
  - Ported IBM's CMM to Xen/x86

## Overheads

- Worst-case: large-chunk sequential reads
  - Overhead of hashing: 0.2%
  - Overhead of hashing + IPC : 34.8%
- “Realistic” benchmark (kernel compile)
  - Without Satori: 780 seconds
  - Satori enabled: 779 seconds

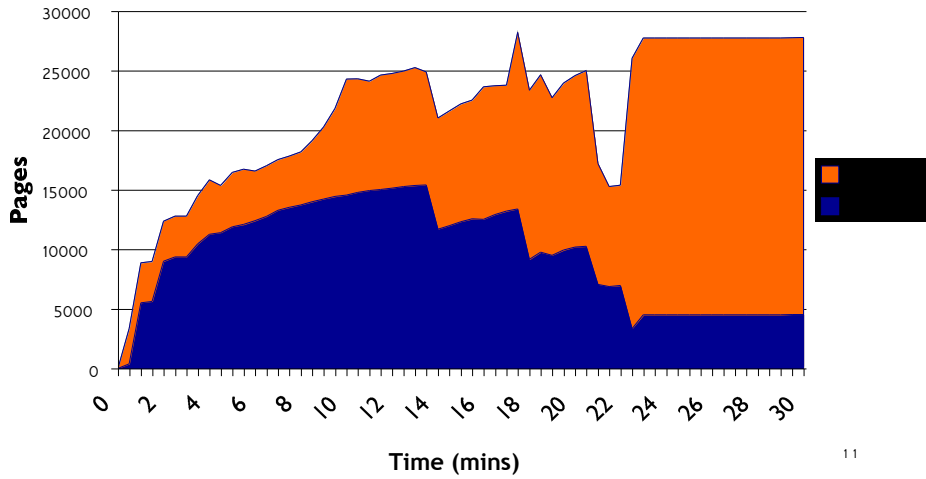
# Effectiveness

Kernel Compilation (256MB)



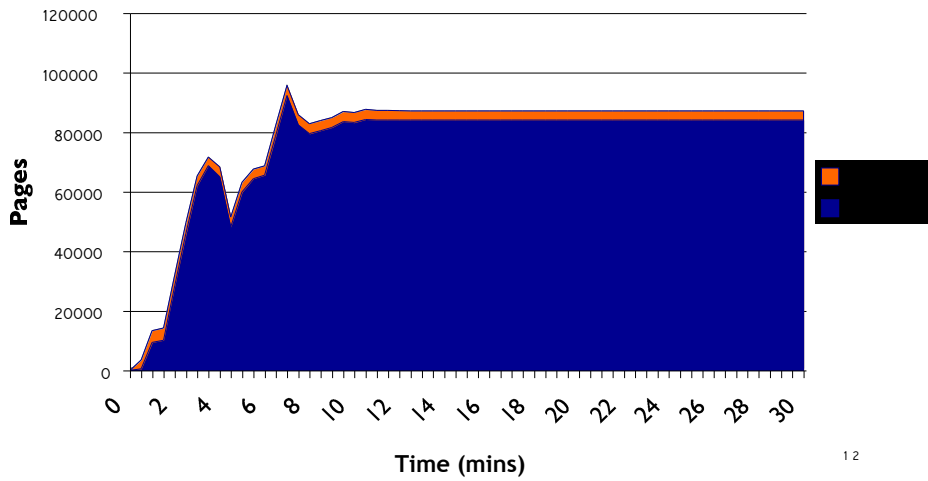
# Effectiveness

Kernel Compilation (512MB)



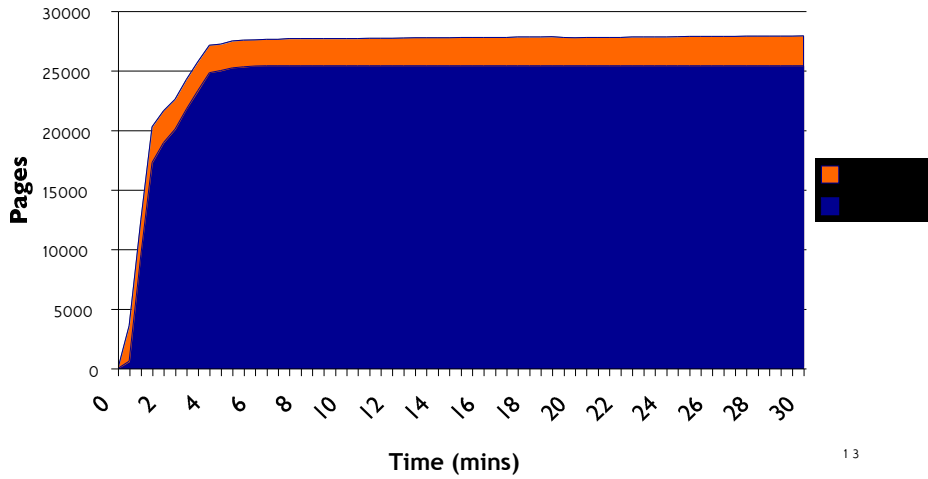
# Effectiveness

httperf



# Effectiveness

RUBiS



## Conclusions

- More memory means more performance
- Page caches are useful sources of sharing
- Enlightening the O S avoids host paging
- Satori is efficient and effective!

## Q uestions?

- Click to add an outline