Harvesting Idle Memory for Application-Managed Soft State with Midas

Yifan Qiao, Zhenyuan Ruan, Haoran Ma
Adam Belay, Miryung Kim, Harry Xu
Soft State Is Everywhere

Increases performance but safe to discard

Examples:

```python
# Return results directly if fib(n) has been calculated.
@memoize
def fib(n):
    if n < 2:
        return n
    else:
        return fib(n-1) + fib(n-2)
```
Soft State Is Everywhere

Application: need

Got

Miss

Put

Construct

Soft State Store
Soft State Is Everywhere

- Trade memory for performance
- Perfect use case for idle memory
Managing Soft State Is Hard

How to improve performance of UICollectionView containing lots of small images?

In my iOS app I have UICollectionView that displays around 1200 small (35x35 points) images. The images are stored in application bundle.

I am correctly reusing UICollectionViewCells but still have performance problems that vary depending on how I address image loading:

- My app is application extension and those have limited memory (40 MB in this case). Putting all 1200 images to Assets catalog and loading them using UIImage(named: "imageName") resulted in memory crashes - system cached images which filled up the memory. At some point the app needs to allocate bigger portions of memory but these were not available because of cached images. Instead of triggering memory warning and cleaning the cache, operating system just killed the app.

- I changed the approach to avoid images caching. I put images to my project (not to assets catalog) as png files and I am loading them using

  NSBundle.mainBundle().pathForResource("imageName", ofType: "png")

  now. The app no longer crashes due to memory error but loading of single image takes much longer and fast scrolling is lagging even on the newest iPhones.
Managing Soft State Is Hard

How to improve performance of UICollectionView containing lots of small images?

In my iOS app I have UICollectionView that displays around 1200 small (35x35 points) images. The images are stored in application bundle.

I am correctly reusing UICollectionViewCell's but still have performance problems that vary depending on how I address image loading:

- My app is application extension and those have limited memory (40 MB in this case). Putting all 1200 images to Assets catalog and loading them using UIImage(named: imageUrl) results in high memory usage. At some point the app needs to allocate bigger portions of memory but these were not available because of cached images. Instead of triggering memory warning and cleaning the cache, operating system just killed the app.

- I changed the approach to avoid images caching. I put images to my project (not to assets catalog) as png files and I am loading them using
  NSBundle.mainBundle().pathForResource("imageName", ofType: "png") now. The app no longer crashes due to memory error but loading of single image takes much longer and fast scrolling is lagging even on the newest iPhones.
Option 1: Storing All Soft State

Available Memory

UI Application

Cache

Available Memory
Option 1: Storing All Soft State

UI Application

Image Cache
Option 1: Storing All Soft State

UI Application

Image Cache

Out-of-memory error!
Option 2: Statically Limiting Cache Size

UI Application

Reserve

Image Cache

Available Memory

Small static size

Left on the table
Option 3: Leveraging OS Page Cache

UI Application

Page Cache

Available Memory

Limited to cache disk blocks

OS Kernel
Design Goals

Option 1: storing all soft state

Option 2: static limit on cache size

Option 3: OS kernel page cache
Design Goals

Option 1: storing all soft state
Option 2: static limit on cache size
Option 3: OS kernel page cache

Responding to memory pressure
Taking full advantage of available memory
Democratizing what can be stored

Can we have a new virtual memory abstraction for soft state?
Midas: A Soft Memory Abstraction

1. Offer the illusion of an *unlimited* cache space

User → UI Application → Kernel

Unlimited space for soft state
Midas: A Soft Memory Abstraction

1. Offer the illusion of an **unlimited** cache space

   ![Diagram of UI Application]

2. Rapidly unmap memory pages to avoid running out of memory

   ![Diagram of Available Memory (pages) and OS Kernel]
Midas: A Soft Memory Abstraction

1. Offer the illusion of an **unlimited** cache space

   - UI Application

2. Rapidly unmap memory pages to avoid running out of memory

3. Transparently access lost soft state by reconstruction

   - Available Memory (pages)

   - OS Kernel
Midas: A Soft Memory Abstraction

① Offer the illusion of an unlimited cache space

② Rapidly unmap memory pages to avoid running out of memory

③ Transparently access lost soft state by reconstruction

How to access soft memory?

How to reclaim soft memory?
How to Access Soft Memory?

Soft memory pointers

- Similar to smart pointers

Soft memory pool

- Allocator
How to Access Soft Memory?

Soft memory pointers
• Similar to smart pointers

Soft memory pool
• Allocator
How to Access Soft Memory?

Soft memory pointers
• Similar to smart pointers

Soft memory pool
• Allocator
How to Access Soft Memory?

Soft memory pointers

• Similar to smart pointers
  ➢ Transparent reconstruction

Soft memory pool

• Allocator

Access by reconstruction
How to Transparently Reconstruct Soft State?

Soft memory pointers
- Similar to smart pointers
- Transparent reconstruction

Soft memory pool
- Allocator
- Initialized with reconstruction logic
How to Reclaim Soft Memory?

Which soft state to evict?

Available Memory (limited)

Cooperative reclamation

Application
Soft Pointers
Runtime

User
Kernel

OS Kernel
Runtime Memory Management

Log-structured allocator

- Organize soft memory as segments

Diagram:
- Application
- Soft Pointers
- Runtime
- Segment
- Pages
- OS Kernel
Log-structured allocator

- Organize soft memory as segments
- Track access frequency (hotness)
Runtime Memory Management

Log-structured allocator
• Organize soft memory as segments
• Track access frequency (hotness)

Concurrent evacuator
◆ Continuously compact objects

Compact and segregate hot/cold/dead objects
Runtime Memory Management

Log-structured allocator

- Organize soft memory as segments
- Track access frequency (hotness)

Concurrent evacuator

- Continuously compact objects

Application

Soft Pointers

Runtime

Free segment  Cold segment  Hot segment

Pages  Pages  Pages

OS Kernel
The OS kernel coordinates with the runtime to reclaim memory.
The OS kernel coordinates with the runtime to reclaim memory.
Runtime Cooperative Reclamation

The OS kernel coordinates with the runtime to reclaim memory

Application
Soft Pointers
Runtime

Release segment

Revoke memory

Memory pressure

OS Kernel
The OS kernel coordinates with the runtime to reclaim memory.

Memory pressure
Runtime Cooperative Reclamation

The OS kernel coordinates with the runtime to reclaim memory.

What if the runtime fails to release memory timely?

Severe memory pressure
Kernel Enforced Reclamation

The OS kernel unmaps pages directly

Fail to release memory timely

Unmap directly

Severe memory pressure

OS Kernel
Kernel Enforced Reclamation

The OS kernel unmaps pages directly

Severe memory pressure

OS Kernel

Fail to release memory timely
Kernel Enforced Reclamation

The OS kernel unmaps pages directly

How to protect the application from segmentation faults?

Segmentation fault!

Application

Runtime

Soft Pointers

Pointer

Pages

Pages

OS Kernel
How to Protect the Application From Segfaults?

Fault free

1. Fault-guarded soft pointer interface
   - Hide raw references
   - Return values by copying

Application

Access by copying

Fault-guarded soft pointer interface

OS Kernel

Pages

Pages

Runtime

Soft Pointers

Pointer

Fault - guarded soft pointer interface

• Hide raw references
• Return values by copying
How to Protect the Application From Segfaults?

Fault free

1. Fault-guarded soft pointer interface
   - Hide raw references
   - Return values by copying

2. Fault-resilient runtime
   - Safely handle memory faults

Access by copying

Capture signal & recover

Application

OS Kernel

Pages

Faulted

Pages

Non-faulted

Faulted

Non-faulted

Soft Pointers

Runtime

Pointer
How much memory should we grant to each application?
How to Coordinate Soft Memory Between Apps?

- Application 1
  - Runtime
  - Soft Memory
  - Available Memory

- Application 2
- Application 3
- Application 4

Global Coordinator

Probe performance sensitivity

Least sensitive

Most sensitive
How to Coordinate Soft Memory Between Apps?

Application 1
- Runtime
- Soft Memory
- Available Memory

Application 2

Application 3

Application 4

Global Coordinator

Δ(memory)

Gradually regrant
Midas in Practice

SocialNet
(from DeathStarBench)

WiredTiger
(used by MongoDB)

HDSearch
(from µSuite)

Storage Server

Timeline Webpages

User Posts

B+ Tree Nodes

Feature Tensors

Page Cache
Midas in Practice

SocialNet (from DeathStarBench)
- RPCs
- Timeline Webpages
- User Posts

WiredTiger (used by MongoDB)
- DB
- Storage I/O
- B+ Tree Nodes

HDSearch (from µSuite)
- DNN
- Feature Tensors

Storage Server
- Disk I/O
- Disk Blocks

Global Coordinator
Evaluation

1. Can Midas harvest and coordinate soft memory among applications?

2. Can Midas quickly react to memory pressure?
Colocating Four Applications

- 20 GiB idle memory
Colocating Four Applications

- 20 GiB idle memory

Baselines:
1. **Overprovisioning**
   (67.5 GiB soft memory usage)
Colocating Four Applications

- 20 GiB idle memory

Baselines:
1. Overprovisioning
   (67.5 GiB soft memory usage)
2. Static Provisioning
   • 5GiB per app
Colocating Four Applications

• 20 GiB idle memory

Baselines:
1. Overprovisioning
   (67.5 GiB soft memory usage)
2. Static Provisioning
   • 5GiB per app
Colocating Four Applications

- 20 GiB idle memory

**Baselines:**

1. **Overprovisioning**
   (67.5 GiB soft memory usage)
2. **Static Provisioning**
   - 5GiB per app

**Midas**

- Initially 5GiB per app
- Dynamically coordinate

**Graph:**
- Overprovisioning
- Midas
- Static Provisioning
Better

0

5

10

15

20

25

30

35

Time (minutes)

0

25

50

75

100

Overall Throughput (%)

Overprovisioning

Midas

Static Provisioning

71% less memory

1.5x higher throughput

20 GiB idle memory

Baselines:

1. Overprovisioning
   (67.5 GiB soft memory usage)

2. Static Provisioning
   • 5GiB per app

Midas

• Initially 5GiB per app
• Dynamically coordinate
Reacting to Memory Pressure

- Run WiredTiger with 15 GiB soft memory initially
Reacting to Memory Pressure

- Run WiredTiger with 15 GiB soft memory initially
- Then launch the memory antagonist
  - **Fast** memory allocation (7 GiB/s) at t=5min and t=10min
Reacting to Memory Pressure

- Run WiredTiger with 15 GiB soft memory initially
- Then launch the memory antagonist
  - **Fast** memory allocation (7 GiB/s) at t=5min and t=10min
Reacting to Memory Pressure

- Run **WiredTiger** with 15 GiB soft memory initially
- Then launch the **memory antagonist**
  - **Fast** memory allocation (7 GiB/s) at t=5min and t=10min

Brief performance drop but quickly recover to the normal level

Midas rapidly scales down soft memory usage to avoid out-of-memory killing
Conclusion

Midas enables applications to harvest idle memory for application soft state

Key designs:

1. The soft memory abstraction offering seemingly unlimited cache space
2. A runtime that manages soft state in available idle memory
3. OS kernel support that quickly reclaims memory under pressure

https://github.com/uclasystem/midas
Thank You!