Dynamic Space Limits for Haskell

Edward Z. Yang <ezyang@cs.stanford.edu>

David Mazieres <dm>
The connection has timed out

The server at 10.10.0.1 is taking too long to respond.

- The site could be temporarily unavailable or too busy. Try again in a few moments.
- If you are unable to load any pages, check your computer's network connection.
- If your computer or network is protected by a firewall or proxy, make sure that Firefox is permitted to access the Web.

Try Again
```
Top - 08:48:40 up 1497 days, 16:14, 3 users, load average: 0.57, 0.46, 0.58
Tasks: 83 total, 1 running, 82 sleeping, 0 stopped, 0 zombie
Cpu(s): 0.2% us, 0.0% sy, 0.0% ni, 99.3% id, 0.5% wa, 0.0% hi, 0.0% si
Mem: 3995404k total, 3837964k used, 157440k free, 102904k buffers
Swap: 6144852k total, 4980988k used, 1163864k free, 748620k cached

<table>
<thead>
<tr>
<th>PID</th>
<th>USER</th>
<th>PR</th>
<th>NI</th>
<th>VIRT</th>
<th>RES</th>
<th>SHR</th>
<th>S</th>
<th>%CPU</th>
<th>%MEM</th>
<th>TIME+</th>
<th>SWAP</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>31294</td>
<td>root</td>
<td>15</td>
<td>0</td>
<td>6835m</td>
<td>1.9g</td>
<td>200</td>
<td>S</td>
<td>0.0</td>
<td>50.7</td>
<td>0:38.65</td>
<td>4.7g</td>
<td>minilogd</td>
</tr>
<tr>
<td>479</td>
<td>root</td>
<td>16</td>
<td>0</td>
<td>127m</td>
<td>3728</td>
<td>2420</td>
<td>S</td>
<td>0.0</td>
<td>0.1</td>
<td>0:45.88</td>
<td>123m</td>
<td>avagent.bin</td>
</tr>
<tr>
<td>3726</td>
<td>mysql</td>
<td>16</td>
<td>0</td>
<td>150m</td>
<td>32m</td>
<td>4576</td>
<td>S</td>
<td>0.0</td>
<td>0.8</td>
<td>1:44.12</td>
<td>118m</td>
<td>mysqld</td>
</tr>
<tr>
<td>30789</td>
<td>nobody</td>
<td>17</td>
<td>0</td>
<td>195m</td>
<td>98m</td>
<td>2952</td>
<td>S</td>
<td>0.0</td>
<td>2.5</td>
<td>0:13.46</td>
<td>96m</td>
<td>spamd</td>
</tr>
</tbody>
</table>
```
rlimits?

in the operating system?
Connections

Worker Process

Worker Process

Worker Process

Web Server

[every server ever]
Connections

- Worker Process
- Worker Process
- Worker Process

Web Server

[every server ever]
Connections

Worker Process

Worker Process

Web Server

[every server ever]
He's Dead, Jim!

Something caused this webpage to be killed, either because the operating system ran out of memory, or for some other reason. To continue, press Reload or go to another page.

Learn more
Resource Limits in the programming language

[JRes, Luna, KaffeOS, WF'04, PRW'03]
Dynamic Space Limits: Challenges
Dynamic Space Limits: Challenges

What does “memory usage” mean?

live words on the heap

true OS usage (e.g. heap fragmentation)

retainer pays

allocator pays
Dynamic Space Limits: Challenges

What does “memory usage” mean?

- Wick-Flatt’04
- Price-Rudy-Wallach’03

live words on the heap

retainer pays

- JRes
- Luna
- KaffeOS

allocator pays

true OS usage

This Work
Dynamic Space Limits: Challenges

What does “memory usage” mean?

How can I structure the heap so that measuring usage is easy?

- KaffeOS
- Luna
- This Work
- JRes
- Price-Rudy-Wallach ’03
- Wick-Flatt ’04

more isolation

less isolation
Dynamic Space Limits: Challenges

What does “memory usage” mean?

How can I structure the heap so that measuring usage is easy?

What happens when a thread runs out of memory? Kill the thread?
Dynamic Space Limits: Challenges

What does “memory usage” mean?

How can I structure the heap so that measuring usage is easy?

What happens when a thread runs out of memory? Kill the thread?

How do I evict users from the system?
Executive Summary

create and use resource containers

newRC limit

withRC rc expr

killRC rc

...without compromising memory safety

forkRC rcset expr

copyRCResult cp result

RCIORRef / RCMVar

enforcing limits within x2 of truth, efficiently!

3% baseline overhead

5%/20% @ 100/1000
The rest of the talk

- How the block-structured heap lets us efficiently enforce our chosen cost semantics.

- How to explicitly deallocate containers without violating memory safety.

- Evaluation & Beyond Haskell
Block-structured heap [DEB'94 MHJP'08]
Traditional View of the Heap

- `thread` 
- `stack`

- `allocates into`
- `heap`

- `root set`
Block-structured Heap [DEB'94, MHJP'08]

- thread
- stack

- allocates into

- nursery
- gen 0
- gen 1
Block-structured Heap

thread stack

allocates into

nursery

gen 0

gen 1

[OEB'94 MHJP'08]
Containers are chains of blocks

allocates into = charges to Alice

thread
stack

nursery  gen 0  gen 1

Alice  Bob  Carl  Dave
withRC Bob program

- thread
- stack

Allocates into:
- nursery
- gen 0
- gen 1

Names:
- Alice
- Bob
- Carl
- Dave
evaluate `alicesThunk`

<table>
<thead>
<tr>
<th>thread</th>
<th>stack</th>
</tr>
</thead>
</table>

```
<table>
<thead>
<tr>
<th>nursery</th>
<th>gen Ø</th>
<th>gen 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Bob</td>
<td>Carl</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Dave</td>
</tr>
</tbody>
</table>
```
bobsFunction arg1 arg2, (Function)

thread
stack

allocates into

nursery  gen Ø  gen 1

Alice  Bob  Carl  Dave
\text{thunk} = \text{fixed space usage cost order of evaluation independent}

\text{function} = \text{unbounded allocation}

\text{See paper for more details}
Resource Usage = N° Blocks in Container

Account for fragmentation

allocates into

thread
stack

1 2 3 4 5
nursery gen 0 gen 1

Alice Bob Carl Dave
asynchronous exception

thread  
stack

allocates into

nursery  gen Ø  gen 1

Alice  Bob  Carl  Dave
asynchronous exception

thread
stack

Haskell! [MPMR '06]

distributes into

nursery gen 0 gen 1

Alice Bob Carl Dave
Block-structured heap summary

- Coarse grained accounting (4KB)

- Reuse heap overflow checks \rightarrow \text{Low overhead}

- Cost semantics \leftrightarrow \text{Profiler}
Container eviction
killRC Bob

RECLAIMED

nursery  gen 0  gen 1

Alice  Bob  Carl  Dave
Is Bob's container garbage?
Is Bob's container garbage? No!
Revocable pointers?

Bob's container

Alice's container

revocable pointer
Revocable pointers?

Bob's container
EVICTED

Alice's container

considered unreachable
Revocable pointers?

[Luna]

Bob's container
EVICTED

Alice's container

Copy only!
Kill all retainers?

- thread
- thread
- thread

- Bob's container (EVICTED)
- Alice's container

Diagram showing relationships between threads and their associated containers.
Kill all retainers: Heap reachability

No global mutable references
Kill all retainers: Taint

- thread
  - {Bob}

- thread
  - {Alice, Bob}

- thread
  - {Alice}

Bob's container EVICTED

Alice's container
Kill all retainers: Taint

- Bob's container: EVICTED
- Alice's container

can be done w/ Restricted IO Monads
Kill all retainers

Bob's container (EVICTED)

Alice's container

can be done w/ Restricted IO Monads
Kill all retainers

Care for some data?

Bob's container
EVICTED

Alice's container
Kill all retainers

Care for some data?

Oh no! It was poisoned with Bob’s data!

Bob’s container

EVICTED

Alice’s container
Kill all retainers

thread
{Bob}

thread
{Alice, Bob}

thread
{Alice, Bob}

Bob's container
EVICTED

Alice's container

Oh no! It was poisoned with Bob's data!

Care for some data?
The retainer problem

- Require data to be copied across threads?

- Kill all threads that may have references to dead data?
The retainer problem

- Require data to be copied across threads?  
  **DO BOTH**

- Kill all threads that may have references to dead data?
Traditional Model

Interthread Communication

MVar

thread

\{Alice\}

Data
Traditional Model

readMVar m

thread

{Alice}

Interthread Communication

MVar

m

Data
Traditional Model

readMVar m

thread

{Alice, Bob}

MVar

m

Data

Bob's Container

Interthread Communication
New Model

Interthread Communication

Revocable Pointer

RCMVar

{Bob}

thread

{Alice}

Data

Bob's Container

NB: It doesn't matter what container RCMVar lives in.
New Model

```
readRCMVar m
```

Revocable Pointer

**RCMVar**

Interthread Communication

```
\{Bob\}
```

Data

Bob's Container

```
\{Alice\}
```

```
\{Bob\} \neq \{Alice\}
```

Thread
New Model

forkRC \{Alice, Bob\}

Revocable Pointer

thread \{Alice, Bob\}

RCMVar \{Bob\}

Data

Bob’s Container
New Model

`readRCMVar m`

Revocable Pointer

RCMVar

Interthread Communication

{Bob}

thread

{Alice, Bob}

{Bob} ⊆ {Alice, Bob}

Data

Bob's Container
New Model

```
copyRCMVar cp m
```

thread

Data

Alice's Container

Data

Bob's Container

Interthread Communication

Revocable Pointer

RCMVar

{Bob}
New programming model

Direct access is fast but dangerous
⇒ Must explicitly opt-in

Copying is slow but safe

Thread is a unit of isolation (forkRC, rcset, expr)
⇒ Do computation in disposable worker thread to reduce necessary copying
Evaluation
# Overhead (1 container)

<table>
<thead>
<tr>
<th>Program</th>
<th>Allocs</th>
<th>Time</th>
<th>Elapsed</th>
<th>TotalMem</th>
</tr>
</thead>
<tbody>
<tr>
<td>circsim</td>
<td>+0.0%</td>
<td>+3.2%</td>
<td>+3.1%</td>
<td>-5.0%</td>
</tr>
<tr>
<td>constraints</td>
<td>+0.0%</td>
<td>+2.8%</td>
<td>+2.9%</td>
<td>+0.0%</td>
</tr>
<tr>
<td>fibheaps</td>
<td>+0.2%</td>
<td>+2.9%</td>
<td>+2.9%</td>
<td>-0.6%</td>
</tr>
<tr>
<td>fulsom</td>
<td>+0.0%</td>
<td>+2.1%</td>
<td>+2.1%</td>
<td>-5.5%</td>
</tr>
<tr>
<td>gc_bench</td>
<td>+0.0%</td>
<td>+0.9%</td>
<td>+0.9%</td>
<td>+0.0%</td>
</tr>
<tr>
<td>happy</td>
<td>+0.9%</td>
<td>+5.4%</td>
<td>+5.5%</td>
<td>+0.5%</td>
</tr>
<tr>
<td>hash</td>
<td>+0.0%</td>
<td>+6.4%</td>
<td>+6.3%</td>
<td>+0.0%</td>
</tr>
<tr>
<td>lcss</td>
<td>+11.2%</td>
<td>+5.0%</td>
<td>+4.9%</td>
<td>+1.9%</td>
</tr>
<tr>
<td>mutstore1</td>
<td>+0.0%</td>
<td>+1.3%</td>
<td>+1.3%</td>
<td>+3.4%</td>
</tr>
<tr>
<td>mutstore2</td>
<td>+0.0%</td>
<td>-0.2%</td>
<td>-0.3%</td>
<td>-0.6%</td>
</tr>
<tr>
<td>power</td>
<td>+0.0%</td>
<td>+3.1%</td>
<td>+2.9%</td>
<td>+2.1%</td>
</tr>
<tr>
<td>spellcheck</td>
<td>+0.0%</td>
<td>+3.3%</td>
<td>+4.0%</td>
<td>+0.0%</td>
</tr>
</tbody>
</table>

Table 1. Garbage collector overhead by nofib
## Overhead (N containers)

<table>
<thead>
<tr>
<th>Conns</th>
<th>RC</th>
<th>RC disabled</th>
<th>Vanilla</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>2,511.7</td>
<td>2,515.2</td>
<td>2,514.5</td>
</tr>
<tr>
<td>50</td>
<td>12,271.3</td>
<td>12,311.2</td>
<td>12,351.3</td>
</tr>
<tr>
<td>100</td>
<td>19,891.2</td>
<td>20,756.2</td>
<td>20,885.6</td>
</tr>
<tr>
<td>1000</td>
<td>18,484.0</td>
<td>22,434.5</td>
<td>23,104.8</td>
</tr>
</tbody>
</table>

**Table 2.** Happstack measurements (requests per second)
Space Limits

Haskell
Track

Space Limits

Block-structured Heap
Enforce Space Limits

- Asynchronous Exceptions
- Block-structured Heap
Reclaim Space Limits

- Restricted IO Monads
- Asynchronous Exceptions
- Block-structured Heap
Space Limits

- Restricted IO Monads
- Asynchronous Exceptions
- Block-structured Heap

http://ezyang.com/rlimits.html