Yak: A High-Performance
Big-Data-Friendly
Garbage Collector

Khanh Nguyen, Lu Fang, Guoqing Xu,
Brian Demsky, Sanazsadat Alamian

University of California, Irvine

Shan Lu
University of Chicago

Onur Mutlu
ETH Zürich
BIG DATA
BIG DATA
Background: GC

Heap
Background: GC
Background: GC

Heap
Background: GC

Heap
Background: GC

Heap
Scalability

JVM crashes due to OutOfMemory error at early stage

[Fang et al., SOSP’15]
Scalability

JVM crashes due to OutOfMemory error at early stage

[Fang et al., SOSP’15]

Management cost

GC time accounts for up to 50% of the execution time

[Bu et al., ISMM’13]
Scalability

JVM crashes due to OutOfMemory error at early stage

[Fang et al., SOSP’15]

Management cost

GC time accounts for up to 50% of the execution time

[Bu et al., ISMM’13]

High cost of the managed runtime is a fundamental problem!
Two Paths, Two Hypotheses
Two Paths, Two Hypotheses

- Data Loads and Feeds
- Queries and Results
- Data Publishing

Cloud

Cluster Controller

- Node Controller
- Node Controller

Partitioner

Aggregate
Join
UDF
Aggregate
Join
UDF
Two Paths, Two Hypotheses

Data Loads and Feeds → Queries and Results → Data Publishing

Cloud

Cluster Controller

Node Controller

Partitioner

Aggregate

Join

UDF

Control path

Data path
Two Paths, Two Hypotheses

Data Loads and Feeds → Queries and Results → Data Publishing

Cloud

Cluster Controller

Partitioner

Node Controller

…

Node Controller

Aggregate
Join
UDF
Aggregate
Join
UDF

Control path

Data path
Two Paths, Two Hypotheses

Data Loads and Feeds → Cloud

Queries and Results

Data Publishing

Cloud

Partitioner

Cluster Controller

Node Controller

... Node Controller

Aggregate

Join

UDF

Aggregate

Join

UDF

Control path

Data path
Two Paths, Two Hypotheses

Data Loads and Feeds
Queries and Results
Data Publishing

Cloud

Partitioner
Cluster Controller
Node Controller
...
Node Controller

Aggregate
Join
UDF
Aggregate
Join
UDF

Control path

Data path
Two Paths, Two Hypotheses

Data Loads and Feeds
Queries and Results
Data Publishing

Cloud

Cluster Controller
Partitioner
Node Controller
Node Controller
Node Controller
Node Controller

Aggregate
Join
UDF
Aggregate
Join
UDF

Control path
Data path
Two Paths, Two Hypotheses

Data Loads and Feeds → Queries and Results → Data Publishing

Cloud

Control path

Data path

Generational Hypothesis
Two Paths, Two Hypotheses

Data Loads and Feeds | Queries and Results | Data Publishing

Cloud

Cluster Controller

Node Controller

Partitioner

Aggregate | Join | UDF | Aggregate | Join | UDF

Control path

Data path

Generational Hypothesis
Two Paths, Two Hypotheses

Data Loads and Feeds → Queries and Results → Data Publishing

Cloud

Cluster Controller

Node Controller

Partitioner

Aggregate

Join

UDF

Control path

Data path

Generational Hypothesis

Epochal Hypothesis
WordCount
WordCount

Document

Mapper
WordCount

Document

setup

Mapper
WordCount

Document

Mapper

setup

words
WordCount

Document

Mapper

setup

words

map
WordCount

Document

Mapper

Heap

setup

words

map
WordCount

Document

setup

words

map

Mapper

Heap
WordCount

Document

Mapper

Heap

setup

words

map
WordCount

Document

Mapper

setup

Heap

words

map
WordCount

Document

Mapper

setup

words

map

Heap
WordCount

Document

Mapper

setup

words

map

cleanup

Heap
WordCount

Document

Mapper

setup

Heap

words

map

cleanup
WordCount

Document

Mapper

Heap

setup

words

map

cleanup
WordCount • Many data objects have the same life span:

Document

setup

map

words

cleanup

Mapper

Heap
WordCount • Many data objects have the same life span:

- Document
  - setup
  - map
  - cleanup

Mapper

Heap

words

GC
WordCount • Many data objects have the same life span:

– GC run in the middle is wasted
WordCount • Many data objects have the same life span:

- GC run in the middle is wasted
- Can be efficiently reclaimed together
WordCount • Many data objects have the same life span:

- GC run in the middle is wasted
- Can be efficiently reclaimed together
Many data objects have the same life span:

- GC run in the middle is wasted
- Can be efficiently reclaimed together
Region-based Memory Management

• Sophisticated static analysis won’t work for data-intensive systems
Region-based Memory Management

• Sophisticated static analysis won’t work for data-intensive systems

• What about control path?
Region-based Memory Management

• Sophisticated static analysis won’t work for data-intensive systems

• What about control path?

genерational GC + region-based memory management
Yak Approach

Heap
Yak Approach

Control Space  Data Space
Yak Approach

Control Space

Generational GC

Data Space
Yak Approach

Control Space

Generational GC

Data Space

Region-based Memory Management
Yak Approach

Control Space
- Generational GC

Data Space
- Region-based Memory Management

Reduced memory management cost
Yak Approach

Control Space
- Generational GC

Data Space
- Region-based Memory Management

Reduced memory management cost

annotate epoch boundary:
- epoch_start()
- epoch_end()
Correctness

Region
Correctness

Region
Correctness

• User-based approach solution:
  – Facade [Nguyen et al., ASPLOS’15]
Correctness

• User-based approach solution:
  – Facade [Nguyen et al., ASPLOS’15]
Correctness

- User-based approach solution:
  - Facade [Nguyen et al., ASPLOS’15]

annotation & refactoring

Developers
Correctness

- User-based approach solution:
  - Facade [Nguyen et al., ASPLOS’15]
Yak: An Automatic Solution

• Yak: the first hybrid GC
**Yak**: An Automatic Solution

• **Yak**: the first hybrid GC
  – Implemented in OpenJDK 8
    • Modified the interpreter, two JIT compilers, the heap layout, the Parallel Scavenge GC
  – NO code refactoring needed;
  correctness guaranteed by the system
Yak: An Automatic Solution

- **Yak**: the first hybrid GC
  - Implemented in OpenJDK 8
    - Modified the interpreter, two JIT compilers, the heap layout, the Parallel Scavenge GC
  - NO code refactoring needed;
  - correctness guaranteed by the system
- On average, vs. default production GC (PS):
  - Reduce **33%** execution time
  - Reduce **78%** GC time
Challenges

• How to create regions?
• How to reclaim regions correctly?
How to Create Regions?

• A region starts at a call to `epoch-start` and ends at a call to `epoch-end`
  • The location of epochs affects performance but not correctness
• Regions are thread-local
• Regions can be nested
How to Create Regions?

void main() {

} //end of main
How to Create Regions?

```c
void main() {

} //end of main
```
How to Create Regions?

```c
void main() {
    epoch_start(); /* epoch #1 */
    for(   ) {
    }
    epoch_end();
} //end of main
```
How to Create Regions?

```c
void main() {

    epoch_start();  // epoch #1
    for(   ) {
        epoch_start();  // epoch #2
        for(   ) {
            // Code...
        }
    }
    epoch_end();

} //end of main
```
How to Create Regions?

void main() {
    epoch_start(); epoch #1
    for(    ) {
        epoch_start(); epoch #2
        for(    ) {
        }
        epoch_end();
    }
    epoch_end();
}

//end of main
void main() {

epoch_start();  // epoch #1
for(   ) {

epoch_start();  // epoch #2
for(   ) {

epoch_start();  // epoch #3
for(   ) {

}  // end of main

epoch_end();

}  // end of main

epoch_end();

epoch_end();

} // end of main

region

CS,*

1,t1

1,t2

2,t1

2,t2

3,t1

3,t2
void main() {
  epoch_start(); //epoch #1
  for(   ) {
    epoch_start(); //epoch #2
    for(   ) {
      epoch_start(); //epoch #3
      for(   ) {
      }      //epoch_end();
    }      //epoch_end();
  }      //epoch_end();
}      //end of main

JOIN(3,t_1, 2,t_1) = 2,t_1
void main() {

epoch_start(); epoch #1
for(   ) {
   epoch_start(); epoch #2
   for(  ) {
      epoch_start(); epoch #3
      for(   ) {
      }
   } epoch_end();
   epoch_end();
}

epoch_end();
} //end of main

Region Semilattice

JOIN(3,t1, 2,t1) = 2,t1
Region Semilattice

```c
void main() {
    epoch_start();  // epoch #1
    for(   ) {
        epoch_start();  // epoch #2
        for(   ) {
            epoch_start();  // epoch #3
            for(   ) {
                epoch_end();
            }
            epoch_end();
        }
        epoch_end();
    }
    epoch_end();
} // end of main
```

\[
\text{JOIN}(1,t_1, 2,t_1, 2,t_2) = \text{CS,*}
\]
void main() {
epoch_start();  
for(   ) {
  epoch_start();
  for(   ) {
  }  
  epoch_end();
}
epoch_end();
}

region

JOIN(2,t1, 2,t2) = CS,*
How to Reclaim Regions Correctly?

Object Promotion Algorithm
How to Reclaim Regions Correctly?

Object Promotion Algorithm
Two key tasks:
• **What:** Identify escaping objects:
Object Promotion Algorithm

Two key tasks:

• **What:** Identify escaping objects:
  - Tracking of cross-region/space references in write barrier
    - A **fast path** for intra-region references
    - Inter-region references are recorded in the **remember sets** of their destination regions
How to Reclaim Regions Correctly?

Object Promotion Algorithm

Two key tasks:

• **What**: Identify escaping objects:
  - Tracking of cross-region/space references in write barrier
    - A *fast path* for intra-region references
    - Inter-region references are recorded in the *remember sets* of their destination regions

• **Where**: Decide the relocation destination:
How to Reclaim Regions Correctly?

Object Promotion Algorithm

Two key tasks:

• **What**: Identify escaping objects:
  - Tracking of cross-region/space references in write barrier
    - A fast path for intra-region references
    - Inter-region references are recorded in the
      remember sets of their destination regions

• **Where**: Decide the relocation destination:
  - Query region semilattice
Region Deallocation

epoch_end()
Region Deallocation

epoch_end()

<CS,*>

<r_1, t_1>

<r_2, t_1>

Remember Set
Region Deallocation

epoch_end()

<CS,*>  <r₁, t₁>

Remember
Set

<r₂, t₁>

t₁’s stack
Region Deallocation

epoch_end()

<r_1, t_1>

<CS, *>

t_1’s stack

<r_2, t_1>

Remember Set
Region Deallocation

\( <\text{CS,*}> \quad <r_1, t_1> \quad t_1\text{'s stack} \quad <r_2, t_1> \)

\[ \text{epoch}_\text{end}() \]

Remember Set

Barrier
Region Deallocation

epoch_end()

<CS,*>  <r₁, t₁>  t₁’s stack

Remember Set

<r₂, t₁>

Barrier

t₂’s stack  t₃’s stack
Region Deallocation

epoch_end()

<CS,*>

<r₁, t₁>

t₁’s stack

t₂’s stack

Barrier

t₃’s stack

Remember
Set

<r₂, t₁>
Region Deallocation

\[ <\text{CS,}*>, <r_1, t_1>, \text{t}_1\text{'s stack}, \text{t}_2\text{'s stack}, \text{t}_3\text{'s stack} \]

\[ \text{escaping roots} \]

\[ <r_2, t_1> \]

\[ \text{epoch\_end()} \]

Remember Set

Barrier
Region Deallocation

epoch_end()

<CS,*>  <r₁, t₁>

t₁’s stack

<CS,*>

t₂’s stack

 Barrier

t₃’s stack

Remember
Set

escaping roots

<r₂, t₁>
Region Deallocation

epoch_end()

<CS,*>  <r₁, t₁>

t₁’s stack  t₂’s stack  t₃’s stack

Remember Set

escaping roots

Barrier

<r₂, t₁>

Remember

Region Deallocation

epoch_end()

<CS,*>  <r₁, t₁>

t₁’s stack

t₂’s stack

t₃’s stack

Remember
Set

escaping roots

Barrier

<r₂, t₁>

...
Region Deallocation

epoch_end()

<CS, *>

<r₁, t₁>

t₁’s stack

t₂’s stack

t₃’s stack

escaping roots

<r₂, t₁>

Remember Set

Barrier
Region Deallocation

epoch_end()

<CS,*>

<r₁, t₁>

t₁’s stack

<CS,*>

<r₂, t₁>

t₂’s stack

t₃’s stack

Remember Set

escaping roots

Barrier

Remember Set
Region Deallocation

\[ <\text{CS,} \ast >, <r_1, t_1>, t_1's \text{ stack}, t_2's \text{ stack}, t_3's \text{ stack} \]

Remember Set

escaping roots

\[ <r_2, t_1> \]

epoch_end()
Region Deallocation

\[ <CS, *> \]
\[ <r_1, t_1> \]
\[ t_1's \ stack \]
\[ \text{t_2's stack} \]
\[ \text{t_3's stack} \]
\[ \text{escaping roots} \]
\[ \text{epoch_end()} \]

Remember
Set

...
Region Deallocation

epoch_end()

<CS,*> <r1, t1>

t1’s stack

Remember Set

escaping roots

... 

<r2, t1>

t2’s stack

t3’s stack

Barrier
Region Deallocation

epoch_end()

<CS, *>

<r₁, t₁>

<ref>

t₁’s stack

Barrier

t₂’s stack

t₃’s stack
Region Deallocation

epoch_end()

<CS,*> <r₁, t₁>

t₁’s stack
t₂’s stack
t₃’s stack
Region Deallocation

epoch_end()

<CS,*>  <r₁, t₁>

t₁’s stack
Region Deallocation

<CS, *>

<r₁, t₁>

t₁’s stack
Evaluations

3 real-world systems, 9 applications:

• **Hadoop**
  – Popular distributed MapReduce implementation

• **Hyracks** [Borkar et al., ICDE’11]
  – Data-parallel platform to run data-intensive jobs on a cluster of shared-nothing machines

• **GraphChi** [Kyrola et al., OSDI’12]
  – High-performance graph analytical framework for a single machine
Improvement Summary

<table>
<thead>
<tr>
<th></th>
<th>Hyracks</th>
<th>Hadoop</th>
<th>GraphChi</th>
<th>Normalized Performance (to PS)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GC Time</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>App. Time</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Total Time</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Improvement Summary

-78%
Improvement Summary

-78%  
-2%

Normalized Performance (to PS)

GC Time  
Hyracks  Hadoop  GraphChi

App. Time  
Hyracks  Hadoop  GraphChi

Total Time  
Hyracks  Hadoop  GraphChi
More Data in the Paper

• Statistics on Yak’s heap
• Overhead breakdown
  – Write barrier cost
  – Extra space cost
Conclusion

• **Goal:** Reduce memory management cost in Big Data systems

• **Solution:** *Yak*, the first hybrid GC
  – 33% execution time savings
  – 78% GC time reduction
  – Requires almost **zero** user effort
Thank You!