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
Background: GC

Heap
Background: GC
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
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

Node Controller  Node Controller

Partitioner  Join  UDF  Aggregate  Join  UDF

Control path  Data path
Two Paths, Two Hypotheses
Two Paths, Two Hypotheses

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

Cloud

- Node Controller
  - Aggregate
  - Join
  - UDF
- Node Controller
  - Aggregate
  - Join
  - UDF

Partitioner

Cluster Controller

MASTER

Control path

Data path
Two Paths, Two Hypotheses

Data Loads and Feeds
Queries and Results
Data Publishing

Cloud

Partitioner
Cluster Controller

Node Controller

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

Generational Hypothesis
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

Control path

Data path

Generational Hypothesis
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

Generational Hypothesis

Epochal Hypothesis
WordCount
WordCount

Document

Mapper
WordCount

Document

Mapper

setup
WordCount

Diagram:
- Document
- Mapper
- setup
- words
WordCount

Document

setup

words

map

Mapper
WordCount

Document

Mapper

Heap

setup

words

map
WordCount

Document

Mapper

Heap

setup

words

map
WordCount

Document

Mapper

Heap

setup

words

map
WordCount

Document

Mapper

Heap

setup

words

map
WordCount

Document

setup
goes to words
map

Mapper

Heap
WordCount

Document

Mapper

Heap

setup

map

cleanup

words
WordCount

Document

setup

words

map

cleanup

Mapper

Heap
WordCount

Document

setup

words

map

cleanup

Mapper

Heap
WordCount

Many data objects have the same life span:

- Document
- Mapper
- Heap

setup

words

map

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

Document ➔ setup ➔ words ➔ map ➔ cleanup ➔ GC

Mapper ➔ Heap
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?

**generational 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
Reduced memory management cost

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]

annotation & refactoring
Correctness

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

```c
void main() {

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

void main() {

} //end of main
How to Create Regions?

```c
void main() {
  epoch_start(); // epoch #1
  for( ) {
    ...
  }
  epoch_end();
}
```

1, t₁
1, t₂

CS,*
How to Create Regions?

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

```
epoch_start();
for( ) {
epoch_start();
for( ) {
}
epoch_end();
}
epoch_end();
```
How to Create Regions?

```c
void main() {
    epoch_start();
    for(  ) {
        epoch_start();
        for(  ) {
            }  // end of main
        epoch_end();
    }  // end of main
    epoch_end();
    }  // end of main
```
void main() {

epoch_start(); \[\text{epoch #1}\]
for(   ) {

epoch_start(); \[\text{epoch #2}\]
for(   ) {

epoch_start(); \[\text{epoch #3}\]
for(   ) {

} \[\text{epoch_end()};\]

} \[\text{epoch_end()};\]

} //end of main

epoch_start();
for(   ) {

epoch_start();
for(   ) {

} \[\text{epoch_end()};\]

region

epoch_start();
for(   ) {

} \[\text{epoch_end()};\]
void main() {
    epoch_start();
    for( ) {
        epoch_start();
        for( ) {
            epoch_end();
        }
        epoch_end();
    }
    epoch_end();
} // end of main

JOIN(3,t_1, 2,t_1) = 2,t_1
Region Semilattice

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

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();
}
```

```
JOIN(2, t1, 2, t2) = CS,*
```

```
epoch #1
```

```
epoch #2
```

```
epoch #3
```

```
region
```

```
1, t1
```

```
1, t2
```

```
2, t1
```

```
2, t2
```

```
3, t1
```

```
3, t2
```

```
CS,*
```
Region Semilattice

```c
void main() {
    epoch_start();
    for( ) {
        epoch_start();
        for( ) {
            epoch_start();
            for( ) {
            }
            epoch_end();
        }
        epoch_end();
    }
    epoch_end();
    // end of main
}
```

\[
\text{JOIN}(1,t_1,1,t_2) = \text{CS,*}
\]

\[
\begin{align*}
\text{region} & : \{1,t_1,1,t_2\} \\
& = \text{JOIN}(1,t_1,1,t_2)
\end{align*}
\]
How to Reclaim Regions Correctly?

Object Promotion Algorithm
How to Reclaim Regions **Correctly**?

Object Promotion Algorithm

Two key tasks:

• **What**: Identify escaping objects:
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
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

<CS,*>  <r₁, t₁>  <r₂, t₁>
epoch_end()
Region Deallocation

epoch_end()

<CS,*>

<r₁, t₁>

<CS,*>

<r₂, t₁>

Remember
Set

epoch_end()
Region Deallocation

epoch_end()

<CS,*>

<r₁, t₁>

<CS,*>

<r₂, t₁>

记住

Set

t₁’s stack

红橙绿

蓝黄
Region Deallocation

```
epoch_end()
```

Remember:
Set

\(<\text{CS,*}>\>

\(<r_1, t_1>\>

\(t_1\)'s stack

\(<r_2, t_1>\)
Region Deallocation

epoch_end()

<CS,*> <r₁, t₁> t₁’s stack

Remember Set

<r₂, t₁>

Barrier
Region Deallocation

epoch_end()

<CS,*>

<r₁, t₁>

t₁’s stack

t₂’s stack

t₃’s stack

Remember

Set

Barrier
Region Deallocation

epoch_end()

<CS,*>

<r₁, t₁>

t₁’s stack

t₂’s stack

t₃’s stack

Barrier

Remember Set

<r₂, t₁>
Region Deallocation

\[<\text{CS,*}>\]  \[<r_1, t_1>\]

\[t_1\text{'s stack}\]

\[t_2\text{'s stack}\]

\[t_3\text{'s stack}\]

escaping roots

<\[r_2, t_1]\>

Remember Set

epoch\_end()
Region Deallocation

epoch_end()

Remember Set

escaping roots

<r₂, t₁>

<r₁, t₁>

<CS, *>

t₁’s stack

t₂’s stack

t₃’s stack

Barrier
Region Deallocation

epoch_end()

<CS,*>

<r₁, t₁>

t₁’s stack

escaping roots

<r₂, t₁>

Barrier

t₂’s stack

t₃’s stack

Remember Set

...
Region Deallocation

epoch_end()

<CS,*>  <r₁, t₁>  t₁’s stack  t₂’s stack  t₃’s stack

escaping roots

Remember Set

<r₂, t₁>

…
Region Deallocation

<CS,*>  <r₁, t₁>

t₁’s stack

<CS,*>  <r₁, t₁>

t₁’s stack

Remember Set

escaping roots

<r₂, t₁>

t₂’s stack

Barrier

t₃’s stack

epoch_end()
Region Deallocation

epoch_end()

<CS,*>  <r₁, t₁>

t₁’s stack  t₂’s stack  t₃’s stack

escaping roots

Remember Set

Barrier

<r₂, t₁>

Remember Set
Region Deallocation

epoch_end()
Region Deallocation

epoch_end()

Set

Remember

escaping roots

...
Region Deallocation

epoch_end()

<CS,*>  <r₁, t₁>

<CS,*>  <r₁, t₁>

t₁’s stack

<CS,*>

Remember Set

escaping roots

...  ...  ...  ...  ...  ...

<r₂, t₁>

<CS,*>

t₂’s stack

t₃’s stack

Barrier
Region Deallocating

- `<CS,*>` to `<r1, t1>`
- `t1`'s stack
- `t2`'s stack
- `t3`'s stack

Barrier

`epoch_end()`
Region Deallocation

epoch_end()

<CS,*>

<r₁, t₁>

t₁’s stack

t₂’s stack

t₃’s stack
RegionDeallocation

<CS,*>

<r₁, t₁>

t₁’s stack

epoch_end()
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>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GC Time</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>App. Time</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Total Time</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Normalized Performance (to PS)

- **Hyracks**: 0.2
- **Hadoop**: 0.4
- **GraphChi**: 0.6

---

**Summary**

- **Hyracks** shows significant improvements in GC Time, App. Time, and Total Time compared to Hadoop and GraphChi.
- **Hadoop** and **GraphChi** have similar baseline performances and improvements.

---

**Note**: These improvements are normalized to the performance of the baseline system (PS).
Improvement Summary

-78%

Normalized Performance (to PS)

GC Time

Hyracks
Hadoop
GraphChi

App. Time

Hyracks
Hadoop
GraphChi

Total Time

Hyracks
Hadoop
GraphChi
Improvement Summary

Normalized Performance (to PS)

-78%

-2%

GC Time

App. Time

Total Time

Hyracks

Hadoop

GraphChi
<table>
<thead>
<tr>
<th></th>
<th>Hyracks</th>
<th>Hadoop</th>
<th>GraphChi</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GC Time</strong></td>
<td><img src="image" alt="" /></td>
<td><img src="image" alt="" /></td>
<td><img src="image" alt="" /></td>
</tr>
<tr>
<td><strong>App. Time</strong></td>
<td><img src="image" alt="" /></td>
<td><img src="image" alt="" /></td>
<td><img src="image" alt="" /></td>
</tr>
<tr>
<td><strong>Total Time</strong></td>
<td><img src="image" alt="" /></td>
<td><img src="image" alt="" /></td>
<td><img src="image" alt="" /></td>
</tr>
</tbody>
</table>

**Normalized Performance (to PS)**

- **-78%**
- **-2%**
- **-33%**

*Improvement Summary*
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!