Gerenuk: Thin Computation over Big Native Data Using Speculative Program Transformation

Christian Navasca*
Brian Demsky†

Cheng Cai*
Shan Lu‡

Khanh Nguyen*
Miryung Kim*

Harry Xu*

UCLA

UCI

†
‡
Previous Approaches Focus on One Aspect of Overhead

Skyway (Serialization and Deserialization)
- 1.4x speedup
- 77% more network traffic

Yak (GC)
- 1.7x speedup
- 12% increased memory usage
Instead of processing Objects, process bytes
  • Removes object overhead, greatly improves performance

However, Tungsten is not general
  • Only for simple data types
  • Adds overhead to certain applications

Can we find a scalable, general solution?
Our Solution: Gerenuk

We ran on 12 applications across two frameworks:
- Improved performance by 1.6x
- Reduced memory usage by 26%
Developers Write Data Processing Applications

Data Processing

Deserialization

Input from Worker A

Input from Worker B

map, reduce, etc.

Data Processing

Serialization

Output to Worker X

Output to Worker Y
Goal: Remove Objects Through *byte inlining*

Process *inlined bytes* instead of *objects*.

```java
foo { String s; int i; }  
```

![Diagram](image-url)

- Header
- Payload
- Header
- Payload
- Payload
The Gerenuk Compiler Replaces Objects With Addresses

```java
foo { String s; int i; }
```

- **Header**
  - 3

- **Buffer**
  - "Gerenuk" 3

- **String**
  - `s = foo.s`
  - `foo.i = 10`

- **Buffer**
  - `s = readNative(Buffer + 0, 7)`
  - `writeNative(Buffer + 7, 4, 10)`
Byte Inlining Relies on Confinement

Escaping references are not allowed:

```java
v = foo.s
bar = new Baz()
bar.g = v /* foo.s escapes through bar, violation */
```

In this work, an object we can't inline contains a violation
Byte Inlining Relies on Reference-Immutability

Only primitive-type assignments are allowed:

```c
foo.i = 5 /* ok */
foo.s = "LongerString" /* violation */
```
Our Runtime Allows Recovery Through aborts

- Source Code
- Annotations

Gerenuk Compiler

Speculatively transform source

Executable

Gerenuk Runtime
An abort Runs the Original Task

This is only applicable to dataflow systems (all tasks are independent)
Our Compiler Uses Static Analysis to Find Violations

The Gerenuk Compiler inserts abort instructions when we detect violations

Two main challenges:

1. Scalability
2. False positives
Insight: Most of the Objects are Data Objects

Reduce our scope to only Data Processing

95% of objects created

Input from Worker A

Deserialization

Data Processing

User must annotate

map, reduce, etc.

Input from Worker B

Serialization

Output to Worker X

Output to Worker Y

User must annotate
Traditional Static Analysis Must Consider All Paths

\[
\text{if}(\text{foo}.i == 3)
\]

- True (T)
- False (F)

- \text{foo}.s = "LongerString"
- \text{foo}.i = 5

- False (X)
aborts enable Speculative Transformation

\[
\text{if}(\text{foo.i} == \text{3})
\]

\[
\begin{align*}
\text{foo.s} &= \text{"LongerString"} \\
\text{foo.i} &= \text{5}
\end{align*}
\]

Gerenuk Runtime

aborts can be expensive, but should be rare
We Ran 12 Applications Across Two Frameworks

Spark
5 applications (LiveJournal, 37GB Synthetic)
Spark library applications

Hadoop
7 applications (StackOverflow, Wikipedia)
MapReduce jobs found on StackOverflow

11-node cluster, each node contains:
- 2 Xeon(R) CPU E5-2640 v3 processors
- 32GB memory
- 200GB SSD
- CentOS 6.9
- Connected via InfiniBand
Gerenuk Improves Runtime and Memory in Spark and Hadoop

We ran on 12 applications across two frameworks:

**Spark**
- Improved performance by 2x
- Reduced memory usage by 18%

**Hadoop**
- Improved performance by 1.4x
- Reduced memory usage by 31%
Gerenuk Improves End-to-End Performance of Spark by 2x

- PageRank: Speedup = 1.45, Memory Usage = 0.91
- KMeans: Speedup = 2.4, Memory Usage = 0.65
- Gradient Boosting: Speedup = 3.4, Memory Usage = 0.87
Gerenuk Improves End-to-End Performance of Hadoop by 1.4x

- In-Mapper Combiner: 1.2x
- Community Expert Detection: 1.4x
- Inactive Users Filtering: 2.1x

Speedup
Memory Usage
Violations are Costly but Infrequent

No experiments hit *abort* instructions

### Simulated PageRank Aborts

<table>
<thead>
<tr>
<th>Number of Aborts</th>
<th>Slowdown</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.29</td>
</tr>
<tr>
<td>2</td>
<td>1.43</td>
</tr>
<tr>
<td>4</td>
<td>1.51</td>
</tr>
<tr>
<td>6</td>
<td>1.66</td>
</tr>
<tr>
<td>8</td>
<td>1.8</td>
</tr>
</tbody>
</table>

### StackOverflow Analytics

<table>
<thead>
<tr>
<th>Number of Aborts</th>
<th>Slowdown</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.12</td>
</tr>
</tbody>
</table>
We present Gerenuk, which contains:

A compiler that speculatively transforms a program
A runtime that handles assumption violations

We ran on 12 applications across two frameworks:

**Spark**
- Improved performance by 2x
- Reduced memory usage by 18%

**Hadoop**
- Improved performance by 1.4x
- Reduced memory usage by 31%