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

Christian Navasca*  Cheng Cai*  Khanh Nguyen*
Brian Demsky†  Shan Lu‡  Miryung Kim*  Harry Xu*

*UCLA  †UCI  ‡THE UNIVERSITY OF CHICAGO

1.0.0
October 30, 2019
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
Tungsten Processes Native Bytes, But is Limited

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

Input from Worker A
Deserialization
Data Processing
map, reduce, etc.
Serialization
Output to Worker X
Output to Worker Y
Input from Worker B
Goal: Remove Objects Through *byte inlining*

Process *inlined bytes* instead of *objects*.

```java
class foo {
    String s;
    int i;
}
```
The Gerenuk Compiler Replaces Objects With Addresses

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

String s = foo.s
foo.i = 10

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

Escaping references are not allowed:

\[
\begin{align*}
  v &= \text{foo.s} \\
  \text{bar} &= \text{new Baz()} \\
  \text{bar.g} &= v \quad /* \text{foo.s escapes through bar, violation */}
\end{align*}
\]

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 \rightarrow \text{Annotations} \rightarrow \text{Gerenuk Compiler} \rightarrow \text{Speculatively transform source} \rightarrow \text{Executable} \rightarrow \text{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

Data Processing

Deserialization

Input from Worker A

Input from Worker B

map, reduce, etc.

User must annotate

Serialization

Output to Worker X

Output to Worker Y

User must annotate
Traditional Static Analysis Must Consider All Paths

if (foo.i == 3)

F

foo.i = 5

T

foo.s = “LongerString”
aborts enable Speculative Transformation

if (foo.i == 3)

abort()

foo.s = “LongerString”

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
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

![Graph showing Speedup and Memory Usage for PageRank, KMeans, and Gradient Boosting]
Gerenuk Improves End-to-End Performance of Hadoop by 1.4x

- In-Mapper Combiner: 1.2x speedup, 0.84x memory usage
- Community Expert Detection: 1.4x speedup, 0.83x memory usage
- Inactive Users Filtering: 2.1x speedup, 0.79x memory usage
Violations are Costly but Infrequent

No experiments hit *abort* instructions

**Simulated PageRank Aborts**

- **Number of Aborts**: 1, 2, 4, 6, 8
- **Slowdown**: 1.29, 1.43, 1.51, 1.66, 1.8

**StackOverflow Analytics**

- **Number of Aborts**: 1
- **Slowdown**: 1.12

19
Summary

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%