DScope: Detecting Real-World Data Corruption Hang Bugs in Cloud Server Systems

Ting Dai¹, Jingzhu He¹, Xiaohui (Helen) Gu¹, Shan Lu², Peipei Wang¹

¹NC State University  ²University of Chicago
Real-World Data Corruption Problem

British Airway service was down for **hours** with financial penalty of **£ 100 million**.
A Data Corruption Hang Bug Example

Hadoop-8614

183  public static void skipFully(
184      InputStream in, long len) ... {
185      while (len > 0) {
186          long ret = in.skip(len);
187          ...  
188          ...  
189          len -= ret;
190      }
191  }

The loop stride (ret) is always 0 when in is corrupted.

Overview of DScope

Application bytecode
Loop path & exit condition extraction
I/O dependent infinite loop identification
False positive hang bug pruning
Data corruption hang bugs
Loop Path & Exit Condition Extraction

- Simple Loops

```java
for ( int j = 0; j < length; j++) {
    String rack = racks[j] ;
    ...
}
```

Loop path: \( 549 \rightarrow 550 \rightarrow \ldots \rightarrow 559 \rightarrow 560 \rightarrow 549 \)

Exit condition: \( j \geq length \)
Loop Path & Exit Condition Extraction

- Nested Loops

Loop paths:

Outer: $544 \rightarrow \ldots \rightarrow 549 \rightarrow 560 \rightarrow \ldots \rightarrow 571 \rightarrow 544$

Inner: $549 \rightarrow 550 \rightarrow \ldots \rightarrow 559 \rightarrow 549$

Outer: $544 \rightarrow \ldots \quad 560 \rightarrow \ldots \rightarrow 571 \rightarrow 544$

DScope then extracts the exit conditions for each loop path.
Loop Path & Exit Condition Extraction

- Loops with exception handling

```java
while (!dataFile.isEOF()) {
    ...
    try {
        key = decorateKey(...dataFile);
        ...
    } catch (Throwable th) {
        //ignore exception
    }
    ...
    try {
        if (key == null)
            throw new IOException(...);
        ...
    } catch (Throwable th) {
        //ignore exception
    }
}
```

- Group invocation statements based on arguments.
- All the statements in the same group throw exceptions when their arguments get corrupted.
- Remove infeasible loop paths.
- Extract exit conditions of the feasible loop paths.
I/O Dependent Infinite Loop Identification

• Exit conditions directly depend on I/O operations

//Soot IR
198  $i1 = r0.<InputStream: read()>(r2)  // $i1 is an I/O related variable
199  if $i1 == -1 goto line #203    //``$i1 == -1'' is the exit condition
...
202  goto line #198
I/O Dependent Infinite Loop Identification

- Exit conditions *indirectly* depend on I/O operations

```
// Soot IR
3  if $l8 >= $l0 goto line #12 //``l8 >= l0'' is the exit condition
...
5  $l2 = $l0 - $l8
6  $l4 = $r2.<InputStream: skip>(&$l2) //$l4 is an I/O related variable
7  $b5 = $l4 cmp 0L
8  if $b5 == 0 goto line #12 //``$b5 == 0'' is the exit condition
9  $l7 = $l8 + $l4
10  $i8 = $l7
11  goto line #3
```

Dependency:
- I/O operation
- $l4
- $l8
- $b5
- $l7
I/O Dependent Infinite Loop Identification

- Exit conditions depend on complex I/O related variables

  - DScope performs an integrated analysis by linking variable information from IR code, Java source code, and Java bytecode.

  - User annotated I/O variables.
False Positive Filtering

Hadoop v2.5.0 WritableUtils.java

```java
307  public static long readVLong(DataInput stream)...
308   byte firstByte = stream.readByte();
309   int len = decodeVIntSize(firstByte);
...
314  for (int idx = 0; idx < len-1; idx++) {
      ...
   }
```

- **False positive condition:**
  - The loop stride is always **positive** when the loop index has a fixed upper bound;
  - The loop stride is always **negative** when the loop index has a fixed lower bound.

It’s a FP because the loop stride is always 1 and the upper bound (len-1) is fixed.

len is I/O dependent
Loop Stride and Bound Inference

- Stride and bounds are denoted by
  - *Numeric primitives*

```java
for (int idx = 0; idx < len-1; idx++) {
    ...
}
```

- **Bound** (len-1)
- **Stride** (1)
Loop Stride and Bound Inference

- Stride and bounds are denoted by
  - APIs in 60 commonly used Java classes
    - Forward index  
    - Reverse index  
    - Check bounds  
    - Reset index  
    - Update bounds

```java
RandomAccessReader dataFile;

while (!dataFile.isEOF()) {
  ...
  dataSize = dataFile.readLong();
}
```

- Bound checking
- Stride forwarding
Evaluation

- Implemented a prototype of DScope using Soot;
- State-of-the-art static bug detectors:
  - Findbugs
  - Infer
## Bug Detection Results

<table>
<thead>
<tr>
<th>System</th>
<th>DScope</th>
<th>Findbugs</th>
<th>Infer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TP</td>
<td>FP</td>
<td>TP</td>
</tr>
<tr>
<td>Cassandra</td>
<td>v2.0.8</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Compress</td>
<td>v1.0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Hadoop</td>
<td>v0.23.0</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>Common</td>
<td>v2.5.0</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Mapreduce</td>
<td>v0.23.0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>v2.5.0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>HDFS</td>
<td>v0.23.0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>v2.5.0</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>Yarn</td>
<td>v0.23.0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>v2.5.0</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>Hive</td>
<td>v1.0.0</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>v2.3.2</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>Kafka</td>
<td>v0.10.0.0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Lucene</td>
<td>V2.1.0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>42</strong></td>
<td><strong>37</strong></td>
<td><strong>2</strong></td>
</tr>
</tbody>
</table>
Data Corruption Hang Bug Types

- **Type 1**: Error codes returned by I/O operations directly affect loop strides.
- **Type 2**: Corrupted data content indirectly affects loop strides.
- **Type 3**: Improper exception handling directly affects loop strides.
- **Type 4**: Improper exception handling indirectly affects loop strides.
Data Corruption Hang Bug Types

• Type 1: Error codes returned by I/O operations directly affect loop strides.

Hadoop-8614

183  public static void skipFully(InputStream in, long len) … {
184      while (len > 0) {
185          long ret = in.skip(len);  // Corrupted InputStream
186          …
187          …
189          len -= ret;  // The loop stride (ret) is always 0 when in is corrupted.
Data Corruption Hang Bug Types

• Type 2: Corrupted data content indirectly affects loop strides.

```
private void readLocalFile(Path path, ...) {... {
  ... 
  byte[] data = new byte[BUFFER_SIZE];
  long size = 0;
  while (size >= 0) {
    size = in.read(data);
  }
}
```

HDFS-13514

194 BUFFER_SIZE = conf.getInt();  // Corrupted configuration file

The loop stride (size) is always 0 when conducting read op on an empty array.
False Negative Example

The loop index, stride or bounds are only related to specific application I/O functions.

HDFS-5438

1668 while (!fileComplete) {
1669 fileComplete = dfsClient.namenode.complete(src,
            dfsClient.clientName, last);

...
False Positive Example

Hadoop v2.5 BlockReaderLocal.java

472 private int **readWithBounceBuffer**(ByteBuffer buf…) {
481 do {
492 …
502 bb = **drainDataBuf**(buf);
512 } while (buf.remaining() > 0);
514 }

277 private int **drainDataBuf**(ByteBuffer buf) {
286 buf.put(dataBuf);
291 }

- **The forwarding-index** Java APIs and the **checking-bounds** Java APIs are located in **different** application function.
Conclusion

- DScope is a new data corruption hang bug detection tool for cloud server systems.
  - Combines candidate bug discovery and false positive filtering.
  - Evaluated over 9 cloud server systems and detects 42 true data corruption hang bugs including 29 new bugs.
Acknowledgements

• DScope is supported in part by NSF CNS1513942 grant and NSF CNS1149445 grant.

Thank you