

# A Hybrid Approach to Cloud System Performance Bug Detection, Diagnosis and Fix

Ting Dai

Advisor: Xiaohui (Helen) Gu

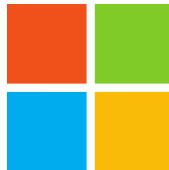
*NC State University*

*July 23<sup>rd</sup>, 2019*

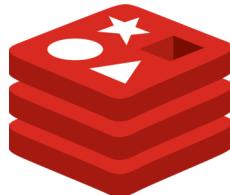
# About Me

- **Background:**
  - Master and Bachelor degree from Nanjing University of Posts and Telecommunications, China
  - Summer intern in InsightFinder and IBM T.J. Watson Research
- **Select papers:**
  - FabZK: Supporting Privacy-Preserving, Auditable Smart Contracts in Hyperledger Fabric [[DSN'19](#)]
  - DScope: Detecting Real-World Data Corruption Hang Bugs in Cloud Server Systems [[SoCC'18](#)]
  - Hytrace: A Hybrid Approach to Performance Bug Diagnosis in Production Cloud Infrastructures [[TPDS'18, SoCC'17](#)]
  - Understanding Real-World Timeout Problems in Cloud Server Systems [[IC2E'18](#)]

# Real-World Performance Problems



- Nov. 2014, Microsoft Azure storage service outage caused by an infinite loop [[link](#)]



- Feb. 2016, Redis cloud outage caused by a resource-intensive infinite loop [[link](#)]



- Jan. 2018, Intel Meltdown patch will slow down your AWS EC2 server [[link](#)]

# State-of-the-Art

## Static analysis

Facebook Infer, Findbugs, PMD,  
Camarel[ICSE'15], Jin et al.[PLDI'12], etc

- **Advantage:**
  - No runtime overhead
  - Program semantics
- **Disadvantage:**
  - High FP by generic approaches
  - High FN by specific approaches

## Dynamic analysis

PerfScope[SoCC'14], PerfCompass[TPDS'15],  
X-ray[OSDI'12], PBI[ASPLOS'13], etc

- **Advantage:**
  - More accurate due to runtime behavior
- **Disadvantage:**
  - Limited coverage (FN)

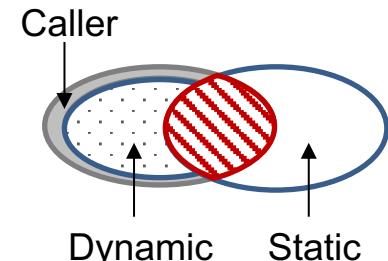
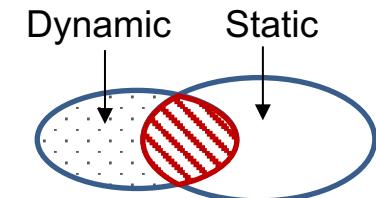
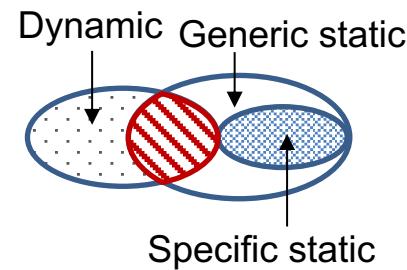
# Hybrid Analysis

- **Rationale:**

- Functions with buggy characteristics which are also experiencing a runtime problem are **most** likely related to a bug.

- **Design choice:**

- A suspicious caller function can cause its callee functions behave abnormally runtime.
- Generic static approach.



# My Contributions

## Oral preliminary

Hytrace

A hybrid approach for diagnosing generic performance bugs.

DScope

Advanced detection for specific performance bugs  
--- data corruption hang bugs.

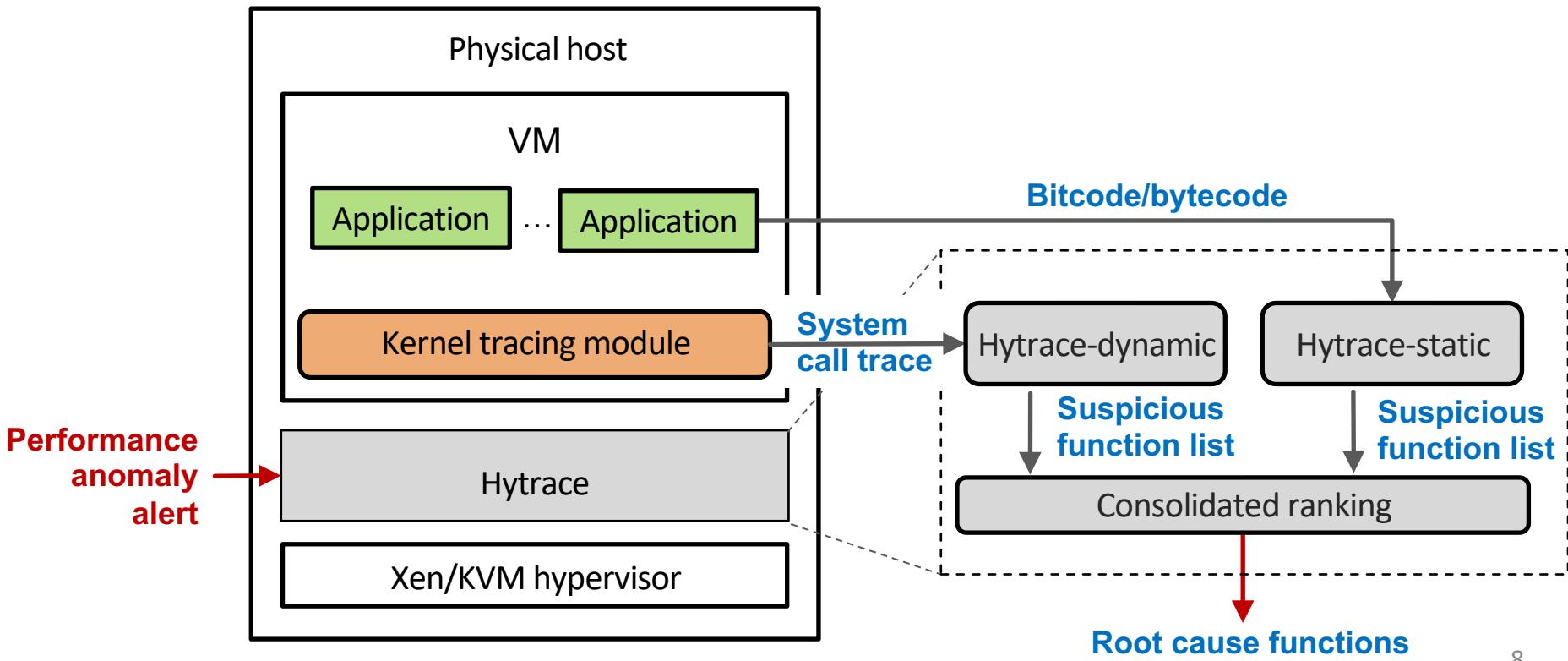
HangFix

Auto-fix for software hang bugs caused by data processing and inter-process communication failures.

Focus

# **Hytrace: A Hybrid Approach to Performance Bug Diagnosis in Production Cloud Infrastructures**

# Overview of Hytrace



# Result Summary

- Hytrace is a hybrid performance diagnosis approach.
  - Combines offline static analysis and online dynamic bug inference.
  - Evaluated over **133** performance bugs on 9 cloud server systems.
  - Reduces FP functions & improve the rank of bug related functions.
  - Imposes less than **3%** CPU overhead to production cloud environments.

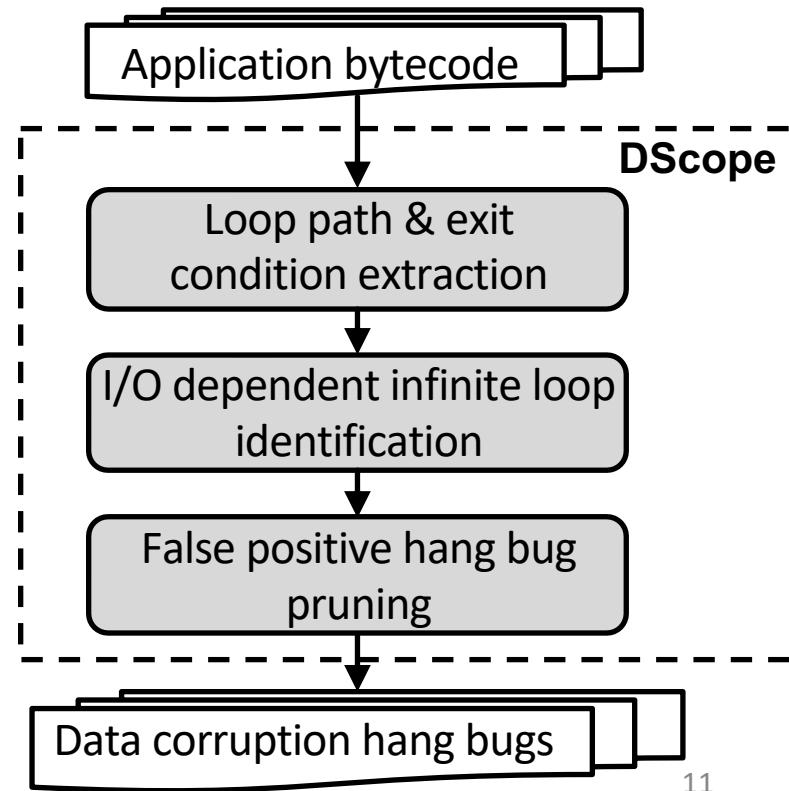
# DScope: Detecting Real-World Data Corruption Hang Bugs in Cloud Server Systems

# A Data Corruption Hang Bug Example

Hadoop-8614

```
183 public static void skipFully(  
    InputStream in, long len) ... {  
184     while (len > 0) {  
185         long ret = in.skip(len); Corrupted  
InputStream  
...  
...  
189         len -= ret; The loop stride (ret) is  
always 0 when in is  
corrupted.  
190     }  
191 }
```

## Overview of DScope



# Bug Detection Results

| System        |           | DScope |    | Findbugs | Infer |
|---------------|-----------|--------|----|----------|-------|
|               |           | TP     | FP | TP       | TP    |
| Cassandra     | v2.0.8    | 2      | 1  | 0        | 1     |
| Compress      | v1.0      | 2      | 2  | 0        | -     |
| Hadoop Common | v0.23.0   | 4      | 6  | 0        | 0     |
|               | v2.5.0    | 6      | 6  | 0        | 0     |
| Mapreduce     | v0.23.0   | 3      | 0  | 0        | 0     |
|               | v2.5.0    | 2      | 0  | 0        | 0     |
| HDFS          | v0.23.0   | 1      | 1  | 0        | 0     |
|               | v2.5.0    | 3      | 5  | 1        | -     |
| Yarn          | v0.23.0   | 2      | 2  | 1        | 0     |
|               | v2.5.0    | 2      | 5  | 0        | 0     |
| Hive          | v1.0.0    | 7      | 6  | 0        | -     |
|               | v2.3.2    | 5      | 1  | 0        | 0     |
| Kafka         | v0.10.0.0 | 1      | 1  | 0        | 0     |
| Lucene        | V2.1.0    | 2      | 1  | 0        | 0     |
| Total         |           | 42     | 37 | 2        | 1     |

# Result Summary

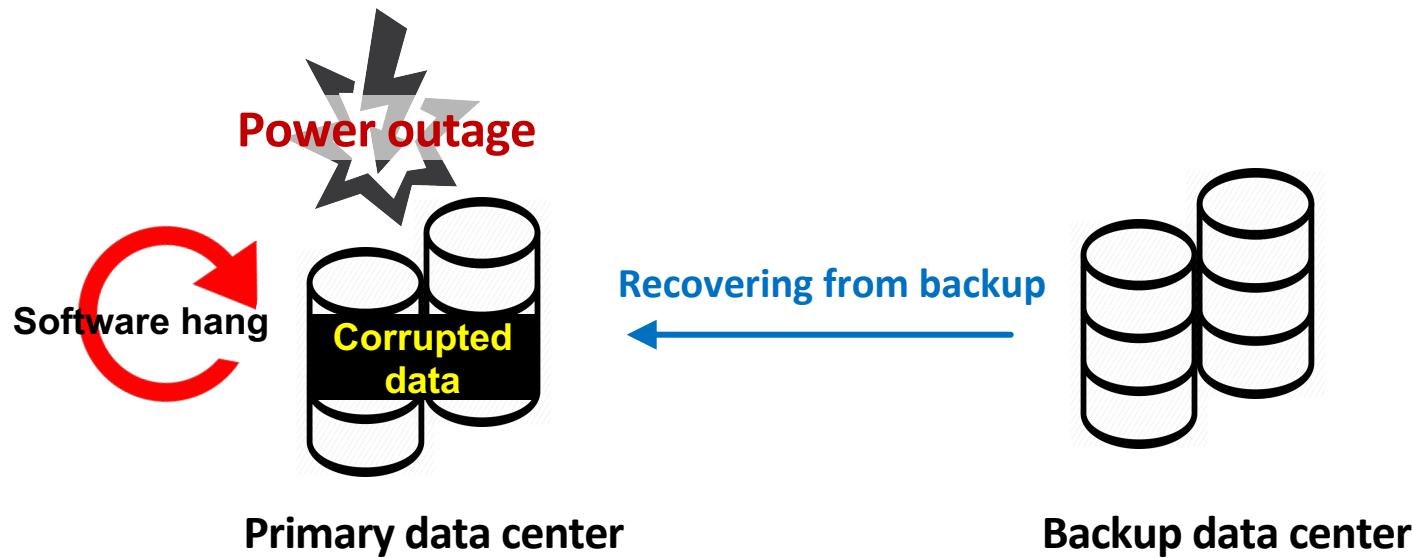
- DScope is a new data corruption hang bug detection tool for cloud server systems.
  - Combines candidate bug discovery and false positive filtering.
  - Evaluated over 9 cloud server systems and detects **42** true data corruption hang bugs including **29** new bugs.

# **HangFix: Automatically Fixing Software Hang Bugs in Cloud Systems**

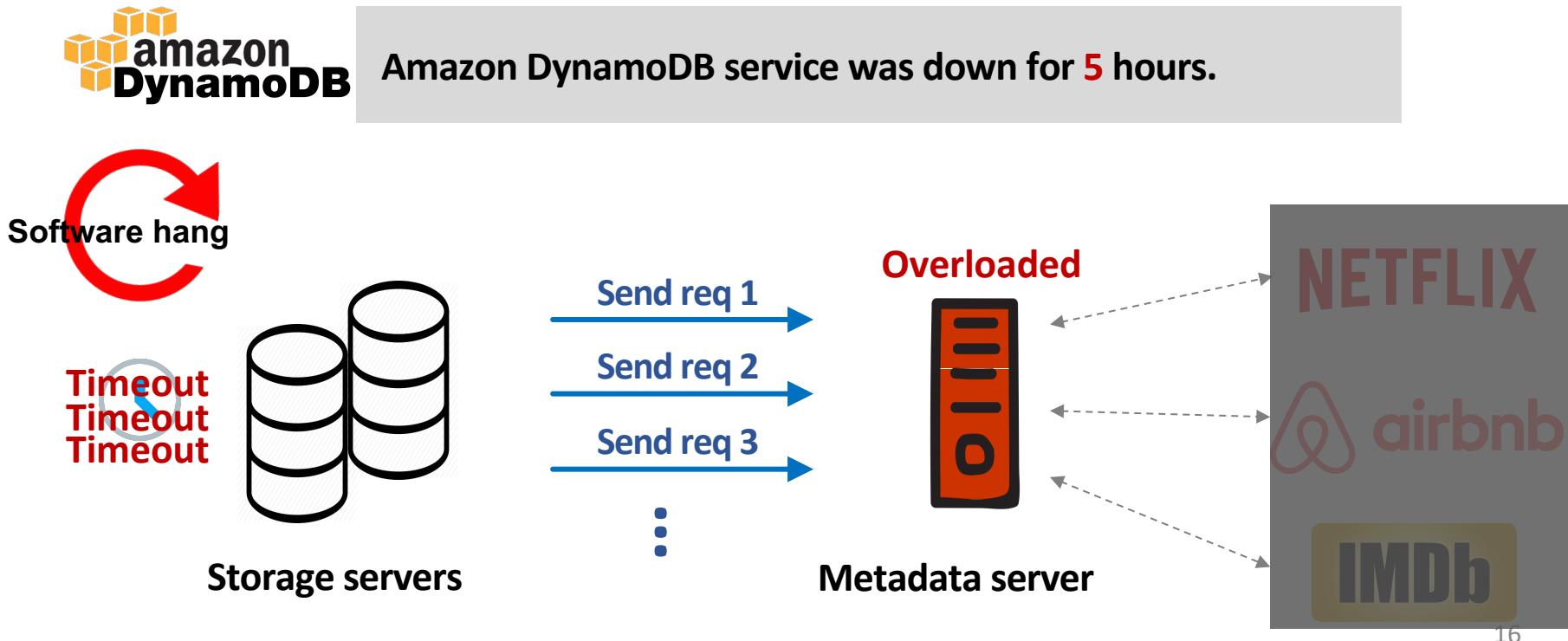
# Real-World Hang Problem Caused by Data Corruption



British Airway service was down for **hours** with financial penalty of **£ 100 million**.

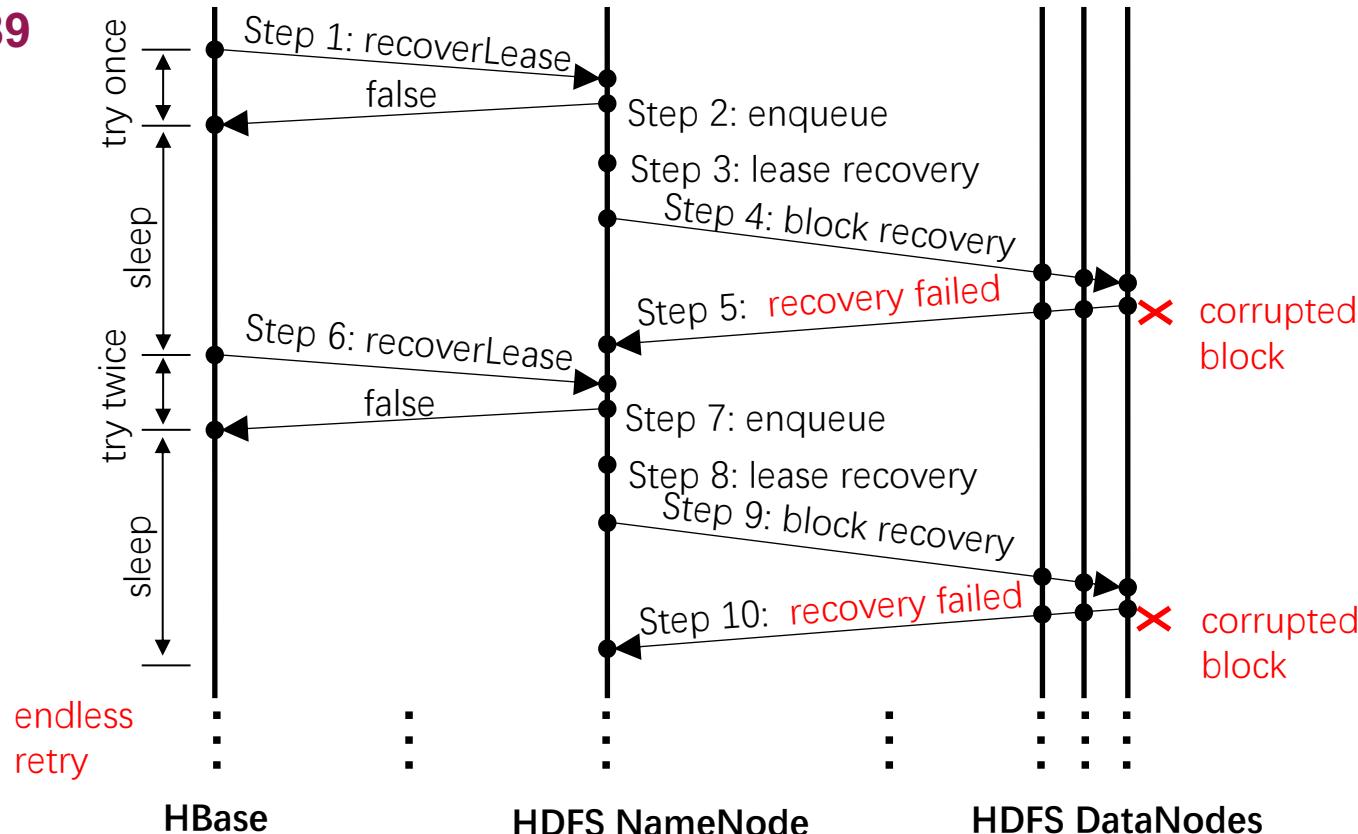


# Real-World Hang Problem Caused by Inter-process communication failure



# A Hang Bug Example

HBase-8389



# A Hang Bug Example

## HBase-8389

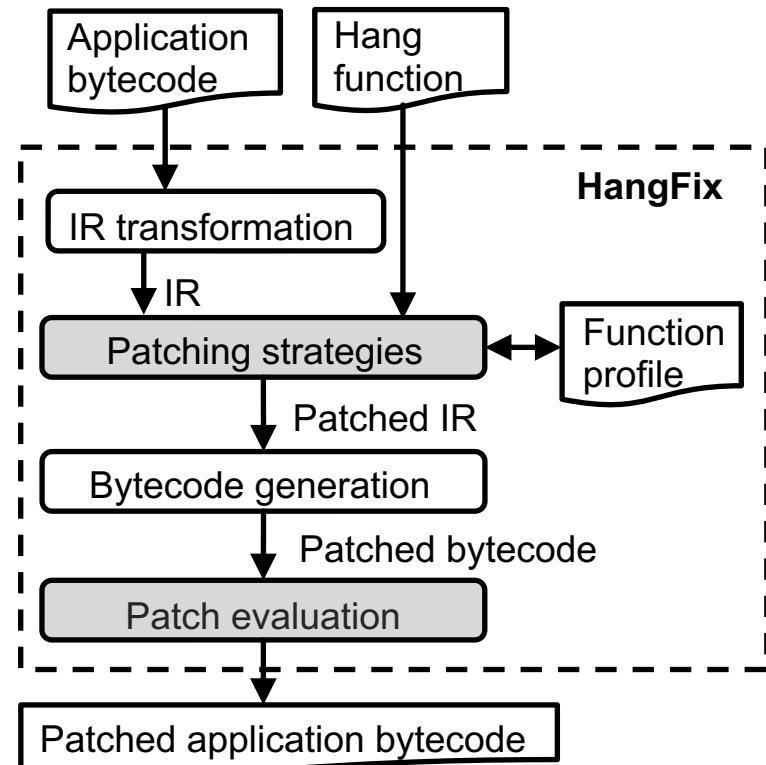
```

+ private String KEY = "recover.timeout";
+ private int DEFAULT = 900000;
+ private long timeout = conf.getInt(
+     KEY, DEFAULT);
+ long st = System.currentTimeMillis();

62 boolean recovered = false;
64 while (!recovered) {
...
71 recovered = dfs.recoverLease(p); //to HDFS
85 if (!recovered) {
96 Thread.sleep(...);
104 }
105 }

+ long e = System.currentTimeMillis() - st;
+ if (timeout > 0 && e >= timeout)
+ throw new TimeoutException("Timeout...");
105 }
```

# Overview of HangFix



# Methodology

- Search all the hang bugs in JIRA using **key words**: *hang, corruption, communicate, connect, read, write*;
- Popular **cloud systems**: Hadoop, Cassandra, etc;
- Extensively **studied** all the hang bugs to my best knowledge.

# Commonly Seen Hang Bug Patterns

- Pattern #1: Unexpected return value causes the loop stride to be incorrectly updated
- Pattern #2: Misconfigured variables cause the loop stride or index incorrectly updated
- Pattern #3: Improper exception or error handling skips loop index updating operations
- Pattern #4: Blocking-prone operations

# Patching Strategies

- Strategy #1: Checking error-prone return values to terminate an infinite loop
- Strategy #2: Checking misconfigured variable to restore default value or break the loop if the default value does not exist
- Strategy #3: Tracing the execution of index update operations and re-executing them or breaking the loop
- Strategy #4: Adding missing timeout over blocking-prone operations

## Pattern #1: Unexpected return value causes the loop stride to be incorrectly updated

### Cassandra-7330

```
114 protected void drain(InputStream dis,  
           long bytesRead) throws IOException {  
115     long toSkip = totalSize() - bytesRead;  
116     toSkip = toSkip - dis.skip(toSkip);  
117     while (toSkip > 0) {  
118         toSkip = toSkip - dis.skip(toSkip);  
119     } }
```

Inaccessible  
or corrupted

Loop stride

Be the same or  
get increased

0 or -1  
(unexpected return)

- The loop stride depends on an operation's returning value;
- The operation **returns** an **unexpected error code** due to some underlying faults such as **data corruption**.

# Fixing Pattern #1 Hang Bug

## Patching Strategy:

- Identify error-prone return values which are used to update the loop stride and terminate the loop by throwing an exception with a known type.

# Fixing Pattern #1 Hang Bug

## Cassandra-7330

```
114 protected void drain(InputStream dis,  
           long bytesRead) throws IOException {  
115     long toSkip = totalSize() - bytesRead;  
116     toSkip = toSkip - dis.skip(toSkip);  
117     while (toSkip > 0) {  
118         toSkip = toSkip - dis.skip(toSkip);  
119     } }
```

0 and -1 are  
not checked.

## Patching Steps:

- Traverses the CFG of the hang function  $f$ ;
- Extracts all the invocation statements from  $f$  along the loop path;
- Checks whether every error-prone return value of every invocation  $f_i$  is checked.

## Function profile



**Class:** DataInputStream

**Method:** long skip(long)

**Error-prone return:** 0, -1

# Fixing Pattern #1 Hang Bug

## Cassandra-7330

```
114 protected void drain(InputStream dis,  
           long bytesRead) throws IOException {  
115     long toSkip = totalSize() - bytesRead;  
116     toSkip = toSkip - dis.skip(toSkip);  
117     while (toSkip > 0) {  
118         toSkip = toSkip - dis.skip(toSkip);  
+         long skipped = dis.skip(toSkip);  
+         toSkip = toSkip - skipped;  
+         if (skipped <= 0 ) {  
+             throw new IOException("Unexpected  
+             return value causes the loop stride  
+             to be incorrectly updated."); }  
119     }
```

0 and -1 are  
not checked.

## Patching Steps:

- An if branch with the condition of  $r_i == r_{err}$ ;
- If more than one error-prone return values, a **combined** condition in the form of  $r_i \geq r_{err\_min}$  or  $r_i \leq r_{err\_max}$ ;
- A **known** exception with the same type of the exception declared by function **f**.

## Pattern #2: Misconfigured variables cause the loop stride or index incorrectly updated

### Hadoop-15415

```
96 public static void copyBytes(...) {  
97 int bufferSize = conf.getInt("buffer.size",4096);  
98 copyBytes(in, bufferSize, true);  
99 }
```

```
74 public static void copyBytes(...) {
```

```
...  
77 byte buf[] = new byte[bufferSize];  
78 int bytesRead = in.read(buf);  
79 while (bytesRead >= 0) {  
84 bytesRead = in.read(buf);  
85 }  
86 }
```

Misconfigured  
to be 0

zero-size array

loop index

always be 0

- An **operation's parameter** or a **class field variable** is misconfigured.
- They reset the **loop index** to **the same value** in each iteration;
- They set the **loop stride** to be **non-positive/non-negative** infinitely when the loop has a fixed **upper/lower** bound.

# Fixing Pattern #2 Hang Bug

## Patching Strategy:

- Identifies the misconfigured variables and 1) automatically restores the default configuration values or 2) breaks the loop if the variable's default value does not exist.

# Fixing Pattern #2 Hang Bug

## Hadoop-15415

```
96 public static void copyBytes(...) {  
97 int bufferSize = conf.getInt("buffer.size",4096);  
98 copyBytes(in, bufferSize, true);  
99 }
```

```
74 public static void copyBytes(...) {  
...  
77 byte[] buf = new byte[bufferSize];  
78 int bytesRead = in.read(buf);  
79 while (bytesRead >= 0) {  
84     bytesRead = in.read(buf);  
85 }}
```

buf.size is not checked.

## Patching Steps:

- Traverses the CFG of the hang function  $f$ ;
- Extracts all the invocation statements from  $f$  along the loop path;
- Checks whether every **error-prone parameter** of every invocation  $f_i$  is checked;

## Function profile

**Class:** InputStream  
**Method:** int skip(byte[])  
**Error-prone para:** byte[0]

# Fixing Pattern #2 Hang Bug

## Hadoop-15415

```
96 public static void copyBytes(...) {  
97     int bufferSize = conf.getInt("buffer.size",4096);  
98     copyBytes(in, bufferSize, true);  
99 }
```

```
74 public static void copyBytes(...) {  
    ...  
77     byte[] buf = new byte[bufferSize];  
78     int bytesRead = in.read(buf);  
79     while (bytesRead >= 0) {  
84         bytesRead = in.read(buf);  
85     }}
```

## Patching Steps:

- Conducts inter-procedural analysis on the whole program to retrieve the **call graph** to function f;
- Performs data-dependency analysis backwards along the call path to identify the **configuration statement**;

# Fixing Pattern #2 Hang Bug

## Hadoop-15415

```
96 public static void copyBytes(...) {  
97     int bufferSize = conf.getInt("buffer.size",4096);  
+    if(bufferSize == 0) bufferSize = 4096; //default  
98     copyBytes(in, bufferSize, true);  
99 }
```

```
74 public static void copyBytes(...) {  
...  
77     byte[] buf = new byte[bufferSize];  
78     int bytesRead = in.read(buf);  
79     while (bytesRead >= 0) {  
84         bytesRead = in.read(buf);  
85     }}
```

## Patching Steps:

- Add a checker after the configuration statement;
- An if branch with the condition of  $r_i == r_{err}$  (Combined condition, optional);
- Restore default value;

# Fixing Pattern #2 Hang Bug

## Hadoop-15415

```
96 public static void copyBytes(...) {  
97     int bufferSize = conf.getInt("buffer.size",4096);  
    + if(buffSize == 0) bufferSize = 4096; //default  
98     copyBytes(in, bufferSize, true);  
99 }
```

```
74 public static void copyBytes(...) {  
    ...  
77     byte[] buf = new byte[bufferSize];  
78     int bytesRead = in.read(buf);  
79     while (bytesRead >= 0) {  
84         bytesRead = in.read(buf);  
    + if(buffSize == 0) break; //loop tail  
85 }
```

## Patching Steps:

- Add a checker after the configuration statement;
- An if branch with the condition of  $r_i == r_{err}$  (Combined condition, optional);
- Restore default value;
- Add a checker if undefined configuration statement or public hang function;
- Break statement;

## Pattern #3: Improper exception or error handling skips loop index updating operations

### Cassandra-9881

```
103 public void scrub() {  
120     while (!dataFile.isEOF()){  
129         try{  
130             key = sstable.partitioner.decorateKey(  
131                 ByteBufferUtil.readWithShortLength(  
132                     dataFile));  
133             dataSize = dataFile.readLong();  
134         } catch (Throwable th){  
139             ...; //ignore Exception  
140         }  
141     }}}
```

The diagram shows the flow of the `scrub()` method. It starts with a `while` loop that continues as long as `dataFile.isEOF()` is false. Inside the loop, a `try` block attempts to read a key and its size from `dataFile`. If an exception occurs (caught in the `catch` block), it is ignored. A red callout labeled "always false" points to the condition in the `while` loop. A blue callout labeled "skipped" points to the `try` block. A blue callout labeled "Index update ops are skipped" points to the `dataFile.readWithShortLength` call. A red callout labeled "corrupted" points to the `dataFile` parameter. A red oval at the bottom right is labeled "Throw exception". Arrows indicate the flow from the `while` loop to the `try` block, from the `try` block to the `dataFile.readWithShortLength` call, and from the `dataFile` parameter to the "Throw exception" oval.

- **Exception happens or error code** is returned;
- An **improper exception/error handling** changes the control flow;
- Causes the **loop index updating operations** to be **skipped**.

# Fixing Pattern #3 Hang Bug

## Patching Strategy:

- Identifies the unexecuted loop index updating operations and 1) re-executes those operations or 2) terminates the loop by throwing a checkable and acceptable exception.

# Fixing Pattern #3 Hang Bug

## Cassandra-9881

```
103 public void scrub() {  
120     while (!dataFile.isEOF()) {  
+     int numIndexForward = 0;           Re-execution  
129     try {  
130         key = sstable.partitioner.decorateKey(  
131             ByteBufferUtil.readWithShortLength(  
132                 dataFile));  
+         numIndexForward++; //trace index forward  
134         dataSize = dataFile.readLong();  
+         numIndexForward++; //trace index forward  
139     } catch (Throwable th) {  
140         ...; //ignore Exception  
+         if(numIndexForward == 0) throw th;  
141     }}}
```

## Patching Steps:

- Introduces a **counter variable** to record the execution of index updating operations;
- **Increases** or **decreases** the counter value by one after each **index-forwarding** or **index-reversing** operation;
- **Checks** the counter value and **terminate** the loop or **re-execute** index updating operations if the loop index is not updated;

# Fixing Pattern #3 Hang Bug

## Cassandra-9881

```
436 private void scrubOne() throws IOException {  
444   scrubber.scrub();  
459 }
```

```
►103 public void scrub() throws IOException {  
    ...  
    + int numIndexForward = 0;  
    ...  
    + numIndexForward++; //trace index forward  
    ...  
    + numIndexForward++; //trace index forward  
    ...  
    + if(numIndexForward == 0) throw th;  
141 }
```

## Patching Steps:

- Checks the **call stack** of **f** backwards until it identifies the *n*-hop caller function of **f** who **declares a checkable exception** in its function signature;
- **Inserts** the same checkable exception in the signatures of function **f** and its *i*-hop callers,  $i = 1, 2, \dots, n - 1$ ;
- If there are more than one checkable exceptions, chooses the first and most specific one.

## Pattern #4: Blocking-prone operations

### Hive-5235

```
81 public void decompress(ByteBuffer in  
     ByteBuffer out) throws Blocking-prone  
83 try {           operation  
84     int cnt = inflater.inflate(  
85         out.array(), out.arrayOffset() +  
86         out.position(), out.remaining());  
87 } catch (DataFormatException e) {  
88     throw new IOException("Corrupted data");  
89 }
```

JNI

- **Component A blocks** itself waiting to be awakened or **keeps polling** from **Component B** until an operation finishes.
- **A** is where the hang function resides while **B** can be another thread, another node, in JNI code or even OS code.

# Fixing Pattern #4 Hang Bug

## Patching Strategy:

- Identifies the blocking-prone operations and inserts timeout settings on those blocking-prone operations.
  - Checks the elapsed time in each loop iteration and terminates the loop when it has used up the preset time — infinite polling;
  - Invokes timeout setting APIs;
  - Quarantines the blocking-prone operation in a thread with timeout.

# Fixing Pattern #4 Hang Bug

## Hive-5235

```
81 public void decompress(ByteBuffer in, ByteBuffer out) throws IOException {  
82     try (Inflater inflater = new Inflater(true)) {  
83         int cnt;  
84         while ((cnt = inflater.inflate(out, in)) > 0);  
85     } catch (DataFormatException e) {  
86         throw new IOException("Corrupted data");  
87     }  
88 }
```

Has timeout setting

## Patching Steps:

- Inquires the *function profile* and detects that the blocking-prone operation;
- Quarantines the blocking-prone operation in a callable or runnable thread with **timeout** settings.

# Fixing Pattern #4 Hang Bug

## Hive-5235

```
public int inflateWithTO(final Inflater inflater,
    final byte[] b, final int off, final int len)
    throws DataFormatException {
    ExecutorService executor =
        Executors.newSingleThreadExecutor();
    Callable<Integer> callable = new Callable<Integer>() {
        @Override
        public Integer call() throws DataFormatException {
            return inflater.inflate(b, off, len); }}
```

Future<Integer> future = executor.submit(callable);
int cnt = 0;
try { cnt = future.get(timeout, TimeUnit.MILLISECONDS);
} catch (Exception e) { future.cancel(true);
 throw new DataFormatException("Blocking");
} finally { executor.shutdown(); }
return cnt; }

## Patching Steps:

- Uses the Executor-Callable-Future format to quarantine the target op inside a callable thread;

target  
operation

# Fixing Pattern #4 Hang Bug

## Hive-5235

```
public int inflateWithTO(final Inflater inflater,
    final byte[] b, final int off, final int len)
    throws DataFormatException {
    ExecutorService executor =
        Executors.newSingleThreadExecutor();
    Future<Integer> future = executor.submit(new Callable<Integer>() {
        + private Configuration conf = new Configuration();
        + private String KEY = "inflate.timeout";
        + private long DEFAULT= 5000;
        + private long timeout =conf.getLong(KEY, DEFAULT);
        ...
        return cnt;
    });
    try {
        int cnt = 0;
        try { cnt = future.get(timeout, TimeUnit.MILLISECONDS);
        } catch (Exception e) { future.cancel(true);
            throw new DataFormatException("Blocking");
        } finally { executor.shutdown(); }
        return cnt;
    }
}
```

## Patching Steps:

- The timeout setting is in the `future.get()` function with a configurable `timeout` variable.
- Introduces a configurable timeout variable `varnew` with the default value of `v`.
- Extracts the default value by searching the system's configuration files using keywords, such as “timeout”, “interval”, “block”, “poll”.

# Fixing Pattern #4 Hang Bug

## Hive-5235

```
81 public void decompress(...) throws IOException {  
93     try {  
94         + int cnt = inflateWithTO(inflater, ...);  
97     } catch (DataFormatException e) {  
98         throw new IOException("Corrupted data");  
99     }  
}
```

```
→ public int inflateWithTO(final Inflater inflater, ...)  
           throws DataFormatException {  
...  
    try { cnt = future.get(timeout, TimeUnit.MILLISECONDS);  
    } catch (Exception e) {  
        throw new DataFormatException("Blocking...");  
    }  
...}  
}
```

## Patching Steps:

- **Exception transformation** into existing exception handling mechanism.

# Evaluation Methodology

| #  | Bug name       | #  | Bug name       | #  | Bug name    |
|----|----------------|----|----------------|----|-------------|
| 1  | Cassandra-7330 | 15 | HDFS-13513     | 29 | HBase-8389  |
| 2  | Cassandra-9881 | 16 | HDFS-13514     | 30 | Hive-5235   |
| 3  | Compress-87    | 17 | HDFS-14481     | 31 | Hive-13397  |
| 4  | Compress-451   | 18 | HDFS-14501     | 32 | Hive-18142  |
| 5  | Hadoop-8614    | 19 | HDFS-14540     | 33 | Hive-18216  |
| 6  | Hadoop-15088   | 20 | Mapreduce-2185 | 34 | Hive-18217  |
| 7  | Hadoop-15415   | 21 | Mapreduce-5066 | 35 | Hive-18219  |
| 8  | Hadoop-15417   | 22 | Mapreduce-6990 | 36 | Hive-19391  |
| 9  | Hadoop-15424   | 23 | Mapreduce-6991 | 37 | Hive-19392  |
| 10 | Hadoop-15425   | 24 | Mapreduce-7088 | 38 | Hive-19395  |
| 11 | Hadoop-15429   | 25 | Mapreduce-7089 | 39 | Hive-19406  |
| 12 | HDFS-4882      | 26 | Yarn-163       | 40 | Kafka-6271  |
| 13 | HDFS-5438      | 27 | Yarn-1630      | 41 | Lucene-772  |
| 14 | HDFS-10223     | 28 | Yarn-2905      | 42 | Lucene-8294 |

- Implemented a prototype of HangFix using Soot;
- Benchmarks: 42 real-world hang bugs from 10 cloud server systems.

# Patch Evaluation

- We manually studied each bug and wrote testcases to reproduce all 42 hang bugs.
- We evaluate whether HangFix can fix a bug by checking whether the patched application bytecode can successfully pass the testcases.
- We compare HangFix with manual patches presented in bug reports in JIRA to further validate the effectiveness of HangFix.

# Patch Correctness

| Bug name       | Manual | HangFix |
|----------------|--------|---------|
| Cassandra-7330 | ✓      | ✓       |
| Compress-87    | ✓      | ✓       |
| Hadoop-8614    | ✓      | ✓       |
| Hadoop-15088   | ✗      | ✓       |
| Hadoop-15424   | ✗      | ✓       |
| Hadoop-15425   | ✗      | ✓       |
| Mapreduce-6990 | ✗      | ✓       |
| Yarn-163       | ✗      | ✓       |
| Yarn-2905      | ✓      | ✓       |
| Hive-13397     | ✓      | ✓       |
| Hive-18142     | ✗      | ✓       |
| Hive-18219     | ✗      | ✓       |
| Kafka-6271     | ✗      | ✓       |
| Pattern #1     | 5      | 13      |

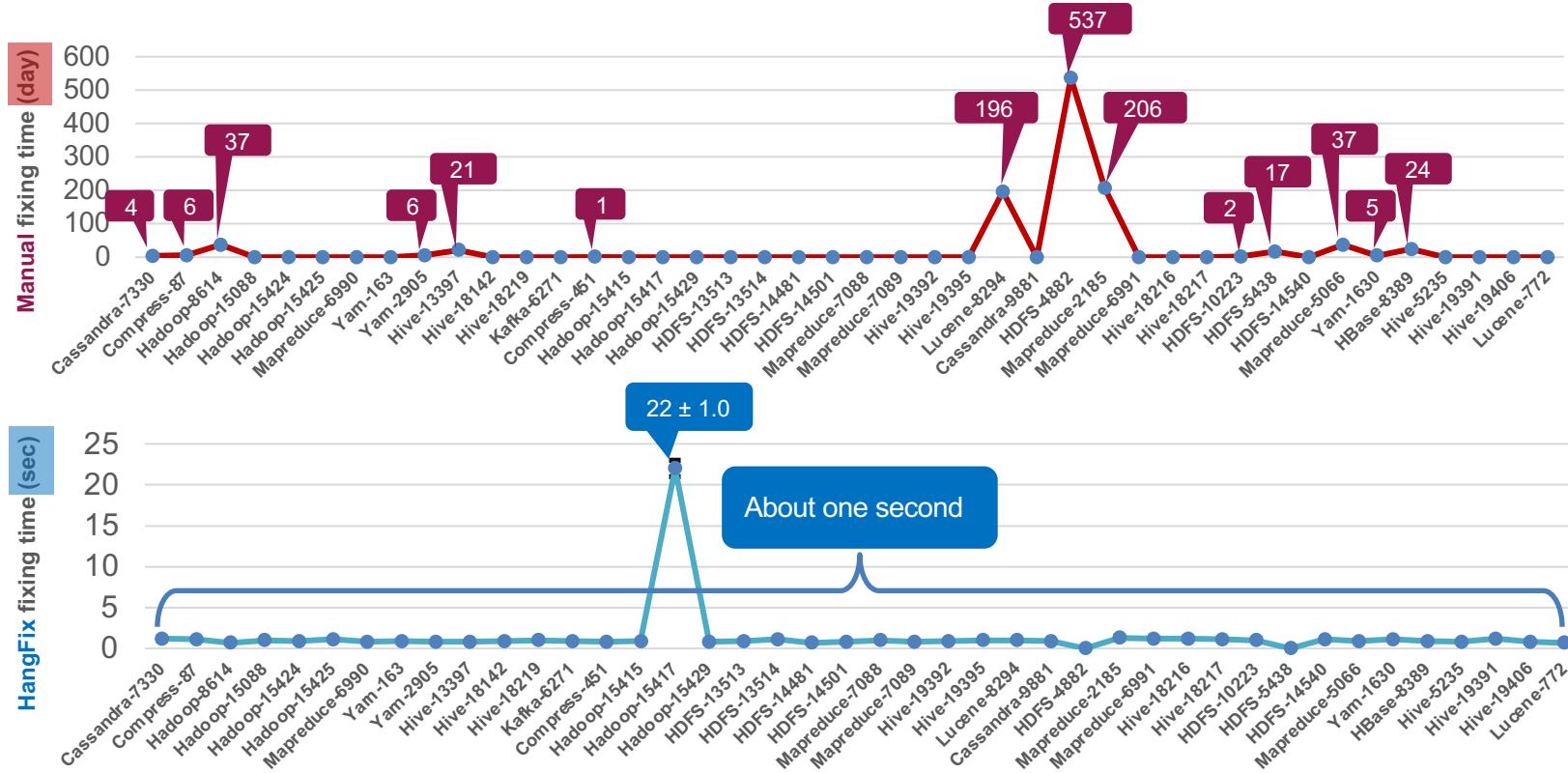
| Bug name       | Manual | HangFix |
|----------------|--------|---------|
| Compress-45    | ✓      | ✓       |
| Hadoop-15415   | ✗      | ✓       |
| Hadoop-15417   | ✗      | ✓       |
| Hadoop-15429   | ✗      | ✓       |
| HDFS-13513     | ✗      | ✓       |
| HDFS-13514     | ✗      | ✓       |
| HDFS-14481     | ✗      | ✓       |
| HDFS-14501     | ✗      | ✓       |
| Mapreduce-7088 | ✗      | ✓       |
| Mapreduce-7089 | ✗      | ✓       |
| Hive-19392     | ✗      | ✓       |
| Hive-19395     | ✗      | ✓       |
| Lucene-8294    | ✓      | ✓       |
| Pattern #2     | 2      | 13      |

| Bug name       | Manual | HangFix |
|----------------|--------|---------|
| Cassandra-9881 | ✗      | ✓       |
| HDFS-4882      | ✓      | ✗       |
| Mapreduce-2185 | ✓      | ✓       |
| Mapreduce-6991 | ✗      | ✓       |
| Hive-18216     | ✗      | ✓       |
| Hive-18217     | ✗      | ✓       |
| Pattern #3     | 2      | 5       |

| Bug name       | Manual | HangFix |
|----------------|--------|---------|
| HDFS-10223     | ✓      | ✓       |
| HDFS-5438      | ✓      | ✗       |
| HDFS-14540     | ✗      | ✓       |
| Mapreduce-5066 | ✓      | ✓       |
| Yarn-1630      | ✓      | ✓       |
| HBase-8389     | ✓      | ✓       |
| Hive-5235      | ✗      | ✓       |
| Hive-19391     | ✗      | ✓       |
| Hive-19406     | ✗      | ✓       |
| Lucene-772     | ✗      | ✓       |
| Pattern #4     | 5      | 9       |

- Manual patch can only fix **14** bugs.
- HangFix successfully fixes **40** out of **42** hang bugs;

# Fixing Time



# Negative Case Example

## HDFS-4882

```
369 public void run() {  
370   for(; fsnamesystem.isRunning(); ) {  
...  
374     checkLeases();-----  
+   fsnamesystem.getEditLog().logSync();  
...  
388 }
```

Restore the corrupted block

```
393 private synchronized void checkLeases() {  
...  
395   for(; sortedLeases.size() > 0; ) {  
396     final Lease oldest = sortedLeases.first();  
...  
+   int numIndexForward = 0;  
412   if(fsnamesystem.internalReleaseLease(p, ...)) {-----  
413     LOG.info("...");  
414     removing.add(p); //remove p from sortedLeases  
+   numIndexForward++;  
416 } else {  
417   LOG.info("...");-----  
418 }  
+   if(numIndexForward == 0) removing.add(p);  
...  
429 }}
```

Always true

Keep processing the same corrupted block

A file associated with the lease path p is corrupted

false

# Result Summary

- HangFix is an application-agnostic hang bug fixing tool.
  - Identifies hang bug patterns and produces software patches based on automatically generated patching strategies.
  - Evaluated on **42** real-world software hang bugs in 10 commonly used cloud server systems. HangFix successfully fixes **40** out of 42 hang bugs within **seconds**.

# Conclusion

- **Hytrace**: a hybrid approach to generic performance bug diagnosis.
  - Combines offline static analysis and online dynamic bug inference.
  - Evaluated over **133** performance bugs on 9 cloud server systems.
  - Reduces FP functions & improve the rank of root-cause functions.
- **DScope**: an advanced data corruption hang bug detection tool.
  - 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.
- **HangFix**: an application-agnostic hang bug fixing tool.
  - Identifies hang bug patterns and produces software patches based on automatically generated patching strategies.
  - Evaluated on **42** real-world software hang bugs in 10 commonly used cloud server systems. HangFix successfully fixes **40** out of 42 hang bugs within **seconds**.

## Limitation & Future Work

- Supporting hybrid diagnosis on distributed performance bugs.
- Integrating inter-procedural analysis on DScope.
- Strengthening the fixing strategies on HangFix.

**Thank you**