FCatch: Automatically Detecting Time-of-fault Bugs in Cloud Systems

Haopeng Liu, Xu Wang*, Guangpu Li, Shan Lu, Feng Ye†, and Chen Tian†

http://fcatch.cs.uchicago.edu/
Cloud Systems
Fault

- Component failure: node crashes and message drops
Faults are common

Machine failures/updates in 29-day Google trace log\(^1\)
(12,583 distinct machines)

5% machines per day!

Cloud systems are fault tolerant?

- When a fault happens at node A
  - Communication with A will timeout
Cloud systems are fault tolerant?

- When a fault happens at node A
  - Communication with A will timeout

Merkle tree repair in Cassandra
Cloud systems are fault tolerant?

- When a fault happens at node A
  - Communication with A will timeout

Merkle tree repair in Cassandra

```
Replica   Primary
```

```
SnapshotRequest
```
Cloud systems are fault tolerant?

- When a fault happens at node A
  - Communication with A will timeout
Cloud systems are fault tolerant?

- When a fault happens at node A
  - Communication with A will timeout
Cloud systems are fault tolerant?

- When a fault happens at node A
  - Communication with A will timeout
Cloud systems are fault tolerant?

- When a fault happens at node A at a special moment
  - Communication with A will hang without timeout
Cloud systems are fault tolerant?

- When a fault happens at node A at a special moment
  - Communication with A will hang without timeout
Cloud systems are fault tolerant?

- When a fault happens at node A at a special moment
  - Communication with A will hang without timeout
Cloud systems are fault tolerant?

- When a fault happens at node A
  - Communication with A will hang without timeout
  - A restarted node will take over A’s task

Task attempt in MapReduce
Cloud systems are fault tolerant?

- When a fault happens at node A
  - Communication with A will hang without timeout
  - A restarted node will take over A’s task
Cloud systems are fault tolerant?

- When a fault happens at node A
  - Communication with A will hang without timeout
  - A restarted node will take over A’s task

Task attempt in MapReduce
Cloud systems are fault tolerant?

- When a fault happens at node $A$
  - Communication with $A$ will hang without timeout
  - A restarted node will take over $A$’s task
Cloud systems are fault tolerant?

- When a fault happens at node A at a special moment
  - Communication with A will hang without timeout
  - A restarted node will fail to take over A’s task
Cloud systems are fault tolerant?

- When a fault happens at node A at a special moment
  - Communication with A will hang without timeout
  - A restarted node will fail to take over A’s task
Cloud systems are fault tolerant?

- When a fault happens at node A at a special moment
  - Communication with A will hang without timeout
  - A restarted node will fail to take over A’s task
Cloud systems are fault tolerant?

- When a fault happens at node A at a special moment
  - Communication with A will hang without timeout
  - A restarted node will fail to take over A’s task
Cloud systems are fault tolerant?

- When a fault happens at node A at a special moment
  - Communication with A will hang without timeout
  - A restarted node will fail to take over A’s task
Why fight TOF bug?

- Common in distributed system [1,2,3]
  - 32% distributed concurrency bugs [1]

[1] Leesatapornwongsa. TaxDC. In ASPLOS’16
[3] Zhenyu. Failure recovery: When the cure is worse than the disease. In HotOS’13
Why fight TOF bug?

- Common in distributed system [1,2,3]
- Difficult to avoid, expose and diagnose
  - Fault rarely occurs during in-house testing
  - Only trigger under special timing

[1] Leesatapornwongsa. TaxDC. In ASPLOS’16
[3] Zhenyu. Failure recovery: When the cure is worse than the disease. In HotOS’13
State of the art – Model checking

Complex manual specifications
Q1: Can we judge what are TOF bugs without manual specifications?
State of the art – Random fault injection

Q1: Can we judge what are TOF bugs without manual specifications?
State of the art – Random fault injection

Q1: Can we judge what are TOF bugs without manual specifications?

Q2: Can we predict TOF bugs based on just one fault injection, instead of many?
Key insights

Q1: Can we judge what are TOF bugs without manual specifications?
Key insights

Q1: Can we judge what are TOF bugs without manual specifications?

A new model of TOF bugs
Key insights

Q1: Can we judge what are TOF bugs without manual specifications?

A new model of TOF bugs
Key insights

Q1: Can we judge what are TOF bugs without manual specifications?

A new model of TOF bugs
Q1: Can we judge what are TOF bugs without manual specifications?

A new model of TOF bugs

```java
Boolean CanCommit(taID) {
    ...
    T.commitID = taID;
}
```

```java
Boolean CanCommit(taID) {
    if (T.commitID) {
        return T.commitID == taID;
    }
}
```
Q1: Can we judge what are TOF bugs without manual specifications?

A new model of TOF bugs

```java
Boolean CanCommit(taID) {
    ...  
    T.commitID = taID;
}

Boolean CanCommit(taID) {
    if (T.commitID) {
        return T.commitID == taID;
    }
}
```
Q1: Can we judge what are TOF bugs without manual specifications?

A new model of TOF bugs

```java
Boolean CanCommit(taID) {
    T.commitID = taID;
}

Boolean CanCommit(taID) {
    if (T.commitID) {
        return T.commitID == taID;
    }
}
```
Q1: Can we judge what are TOF bugs without manual specifications?

A new model of TOF bugs
Key insights

Q1: Can we judge what are TOF bugs without manual specifications?

A new model of TOF bugs
Key insights

Q1: Can we judge what are TOF bugs without manual specifications?

A new model of TOF bugs

- Write and read to a shared state (heap, file, ...)


Key insights

Q1: Can we judge what are TOF bugs without manual specifications?

A new model of TOF bugs

- Write and read to a shared state (heap, file, ...)
- From a crash node and a non-crash node
Key insights

Q1: Can we judge what are TOF bugs without manual specifications?

A new model of TOF bugs

- Write and read to a shared state (heap, file, ...)
- From a crash node and a non-crash node
- Data flow changes with TOF change
Key insights

Q1: Can we judge what are TOF bugs without manual specifications?

A new model of TOF bugs

- Write and read to a shared state (heap, file, ...)
- From a crash node and a non-crash node
- Data flow changes with TOF change

YES!
Key insights

Q1: Can we judge what are TOF bugs without manual specifications?

Q2: Can we predict TOF bugs based on just one fault injection, instead of many?
Key insights

Q1: Can we judge what are TOF bugs without manual specifications?

Q2: Can we predict TOF bugs based on just one fault injection, instead of many?
Key insights

Q1: Can we judge what are TOF bugs without manual specifications?

Q2: Can we predict TOF bugs based on just one fault injection, instead of many?

A new fault-aware logical time model
Key insights

Q1: Can we judge what are TOF bugs without manual specifications?

Q2: Can we predict TOF bugs based on just one fault injection, instead of many?

A new fault-aware logical time model

- Predict data flow changes caused by time-of-fault changes

A correct run

New data flow 1 at fault timing (a)

New data flow n at fault timing (n)
Q1: Can we judge what are TOF bugs without manual specifications?

Q2: Can we predict TOF bugs based on just one fault injection, instead of many?

A new fault-aware logical time model

- Predict data flow changes caused by time-of-fault changes
- Consider both synchronization and fault-tolerance operations
Contribution

A new model of TOF bugs

- Write and read to a shared state (heap, file, ...)
- From a crash node and a non-crash node
- Data flow changes with TOF change
Contribution

A new model of TOF bugs
- Write and read to a shared state (heap, file, ...)
- From a crash node and a non-crash node
- Data flow changes with TOF change

A new fault-aware logical time model
- Predict data flow changes caused by TOF changes
- Consider both synch. and fault-tolerance ops
Contribution

A new model of TOF bugs
- Write and read to a shared state (heap, file, ...)
- From a crash node and a non-crash node
- Data flow changes with TOF change

A new fault-aware logical time model
- Predict data flow changes caused by TOF changes
- Consider both synch. and fault-tolerance ops

FCatch tool
Contribution

- A new model of TOF bugs
  - Write and read to a shared state (heap, file, ...)
  - From a crash node and a non-crash node
  - Data flow changes with TOF change

- FCatch tool
  - Produce correct runs

- A new fault-aware logical time model
  - Predict data flow changes caused by TOF changes
  - Consider both synch. and fault-tolerance ops
Contribution

A new model of TOF bugs
- Write and read to a shared state (heap, file, ...)
- From a crash node and a non-crash node
- Data flow changes with TOF change

A new fault-aware logical time model
- Predict data flow changes caused by TOF changes
- Consider both synch. and fault-tolerance ops

FCatch tool
- Produce correct runs
- Identify conflicting ops
**Contribution**

**A new model of TOF bugs**
- Write and read to a shared state (heap, file, ...)
- From a crash node and a non-crash node
- Data flow changes with TOF change

**A new fault-aware logical time model**
- Predict data flow changes caused by TOF changes
- Consider both synch. and fault-tolerance ops

**FCatch tool**
- Produce correct runs
- Identify conflicting ops
- Identify fault-tolerant ops
Contribution

A new model of TOF bugs
- Write and read to a shared state (heap, file, ...)
- From a crash node and a non-crash node
- Data flow changes with TOF change

A new fault-aware logical time model
- Predict data flow changes caused by TOF changes
- Consider both synch. and fault-tolerance ops

FCatch tool
- Produce correct runs
- Identify conflicting ops
- Identify fault-tolerant ops
Contribution

A new model of TOF bugs
- Write and read to a shared state (heap, file, ...)
- From a crash node and a non-crash node
- Data flow changes with TOF change

A new fault-aware logical time model
- Predict data flow changes caused by TOF changes
- Consider both synch. and fault-tolerance ops

FCatch tool
- Produce correct runs
- Identify conflicting ops
- Identify fault-tolerant ops

Evaluation
- Report 31 TOF bugs
  - 16 of them truly harmful
Outline

- Motivation
- Fault-aware logical time model
- FCatch tool
- Evaluation
- Conclusion
Overview
Overview
Overview

Fault-aware logical time model

Where else could R read data from in future TOF?
Fault-aware logical time model

-- Crash VS. regular
Fault-aware logical time model

-- Crash VS. regular
Fault-aware logical time model

-- Crash VS. regular
Traditional logical time model

Ncrash  Nregular

W₀  W  R

No data flow  HB
Fault-aware logical time model

-- Crash VS. regular

\[ N_{\text{crash}} \quad N_{\text{regular}} \]

\[ W_0 \quad W \quad \text{HB} \quad R \]
Fault-aware logical time model

-- Crash VS. regular
Fault-aware logical time model

-- Crash VS. regular

New data flow between \( N_{\text{crash}} \) and \( N_{\text{regular}} \) introduced by fault.
Fault-aware logical time model

-- Crash VS. recovery
Fault-aware logical time model

-- Crash VS. recovery

Fault-aware logical time model
Fault-aware logical time model

-- Crash VS. recovery
Fault-aware logical time model

-- Crash VS. recovery
Data flow between \textit{Ncrash} and \textit{Nrecovery} is totally determined by TOF.
Fault-aware logical time model

-- Crash VS. recovery

Fault-tolerance: sanity check

//Recovery node
if (f.valid()) { //sanity check
dt = f.read(); //read
}
Fault-aware logical time model

$$N_{crash}$$  $$N_{regular}$$

$$W_0$$  New data flow  HB  $$R$$

$$N_{crash}$$  $$N_{recovery}$$

$$W_0$$  Data flow  $$R$$
What are TOF bugs?
What are TOF bugs?

Crash-regular
What are TOF bugs?
What are TOF bugs?

- *Fault timing:* before W
**What are TOF bugs?**

- **Fault timing:** before W

**Crash-regular**

**Merkle tree repair in Cassandra**

---

- **Replica**
- **Primary**

---

- **SnapshotRequest**
- **SnapshotReply**

---

- W
- HB
- R
What are TOF bugs?

- **Fault timing**: before W

Diagram:
- Crash-regular
- Merkle tree repair in Cassandra
- W
- HB
- R
- Replica
- Primary
- SnapshotRequest
- SnapshotReply
What are TOF bugs?

- **Fault timing:** before W
- **Fault-tolerance:** timeout
What are TOF bugs?

- Fault timing: before W
- Fault-tolerance: timeout
What are TOF bugs?

- **Fault timing:** before W
- **Fault-tolerance:** timeout

**Crash-regular**

**Crash-recovery**

- **Ncrash**
- **Nregular**
- **W**
- **R**
- **HB**
What are TOF bugs?

- **Crash-regular**
  - Fault timing: before W
  - Fault-tolerance: timeout

- **Crash-recovery**
  - Fault timing: after W
What are TOF bugs?

Fault timing: after W
What are TOF bugs?

- **Fault timing:** after W

**Task attempt in MapReduce**

- **TA1**
- **AM**
- **CanCommit (ta1)**
- **W**
- **TA2**
- **CanCommit (ta2)**
- **R**
- **hang**

**Crash-recovery**

- **Ncrash**
- **R**
- **Nrecovery**

- Fault timing: after W
What are TOF bugs?

- **Crash-regular**
  - *Fault timing:* before W
  - *Fault-tolerance:* timeout

- **Crash-recovery**
  - *Fault timing:* after W
  - *Fault-tolerance:* Sanity check
Outline

- Motivation
- Our TOF bug model
- FCatch tool
- Evaluation
- Conclusion
FCatch overview

Produce correct runs → Identify conflicting ops → Identify fault-tolerant ops
Step 1: produce correct runs

- What are correct runs to observe?
Step 1: produce correct runs

- What are correct runs to observe?

  - Fault timing: before W
  - Fault-tolerance: timeout
Step1: produce correct runs

- What are correct runs to observe?

  - Fault timing: before W
  - Fault-tolerance: timeout
Step 1: produce correct runs

- What are correct runs to observe?

- Fault timing: before W
- Fault-tolerance: timeout

No faults need to be injected!
Step 1: produce correct runs

- What are correct runs to observe?

- **Crash-regular**
  - Fault timing: before W
  - Fault-tolerance: timeout

- **Crash-recovery**
  - Fault timing: after W
  - Fault-tolerance: sanity check
Step 1: produce correct runs

- What are correct runs to observe?

- **Fault timing:** before $W$
- **Fault-tolerance:** timeout
- **Fault timing:** after $W$
- **Fault-tolerance:** sanity check

Fault-free run

Crash-regular

Crash-recovery
Step 1: produce correct runs

- What are correct runs to observe?

- **Fault timing:** before $W$
- **Fault-tolerance:** timeout

- **Fault timing:** after $W$
- **Fault-tolerance:** sanity check

- Crash-regular
  - $N_{crash}$
  - $N_{regular}$

- Crash-recovery
  - $N_{crash}$
  - $N_{recovery}$
Step 2: identify conflicting operations

- Crash-regular TOF bug

- Fault timing: before W
- Fault-tolerance: timeout
Step2: identify conflicting operations

- Crash-regular TOF bug
  - W & R are from different nodes (fault-free traces)
  - W & R have blocking happens-before relations

- Fault timing: before W
- Fault-tolerance: timeout
Step 2: identify conflicting operations

- Crash-regular TOF bug
  - W & R are **from** different nodes (fault-free traces)
  - W & R have blocking happens-before relations

![Merkle tree repair in Cassandra diagram]
Step 2: identify conflicting operations

- Crash-regular TOF bug
- Crash-recovery TOF bug
Step 3: identify fault-tolerant ops

- Crash-regular TOF bug

- Fault timing: before W
- Fault-tolerance: timeout
Step 3: identify fault-tolerant ops

- Crash-regular TOF bug

- **Fault timing:** before W

- **Fault-tolerance:** timeout

---

//Regular node
obj.wait(long timeout); //R: obj

Fault-tolerance mechanism: timeout
Step3: identify fault-tolerant ops

- Crash-regular TOF bug

- Fault timing: before W

- Fault-tolerance: timeout

```
//Regular node
obj.wait(long timeout); //R: obj
```

Fault-tolerance mechanism: timeout

Sol: statically check R’s bytecode.
Step3: identify fault-tolerant ops

- Crash-regular TOF bug
- Crash-recovery TOF bug
Outline

- Motivation
- Our TOF bug model
- FCatch tool
- Evaluation
- Conclusion
Methodology

- **Benchmarks**
  - 7 real-world TOF bugs from TaxDC [1]
    - 3 crash-regular TOF bugs
    - 4 crash-recovery TOF bugs
  - 4 distributed systems

[1] Leesatapornwongsa. TaxDC. In ASPLOS’16
## Overall results

<table>
<thead>
<tr>
<th></th>
<th>Crash-regular</th>
<th></th>
<th>Crash-recovery</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#. Bench</td>
<td>#. Unknown</td>
<td>#. Bench</td>
<td>#. Unknown</td>
</tr>
<tr>
<td></td>
<td>(harmful)</td>
<td>(harmful)</td>
<td>(harmful)</td>
<td>(harmful)</td>
</tr>
<tr>
<td></td>
<td>#. FP</td>
<td>#. FP</td>
<td>#. FP</td>
<td>#. FP</td>
</tr>
<tr>
<td>CA1&amp;2</td>
<td>2</td>
<td>1</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>HB-1</td>
<td>1</td>
<td>0</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>6</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>HB-2</td>
<td>-</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MR-1</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MR-2</td>
<td>-</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>ZK</td>
<td>-</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Total</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>5</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>
## Overall results

<table>
<thead>
<tr>
<th></th>
<th>Crash-regular</th>
<th></th>
<th>Crash-recovery</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#. Bench (harmful)</td>
<td>#. Unknown (harmful)</td>
<td>#. FP</td>
<td>#. Bench (harmful)</td>
</tr>
<tr>
<td>CA1&amp;2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>HB-1</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>HB-2</td>
<td>-</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>MR-1</td>
<td>-</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>MR-2</td>
<td>-</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>ZK</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Total</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>
### Overall results

<table>
<thead>
<tr>
<th></th>
<th>Crash-regular</th>
<th></th>
<th>Crash-recovery</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#. Bench (harmful)</td>
<td>#. Unknown (harmful)</td>
<td>#. FP</td>
<td>#. Bench (harmful)</td>
<td>#. Unknown (harmful)</td>
</tr>
<tr>
<td>CA1&amp;2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>HB-1</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>HB-2</td>
<td>-</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>MR-1</td>
<td>-</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MR-2</td>
<td>-</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>ZK</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>
Case study: an unknown harmful report

Create a HDFS flag-file before committing

Done
Case study: an unknown harmful report

Create a HDFS flag-file before committing

Done

Create a HDFS flag-file before committing

Done
Case study: an unknown harmful report

- MR-5485

Create a HDFS flag-file before committing

Done

Create a HDFS flag-file before committing

Done

Create a HDFS flag-file before committing

Done
Case study: an unknown harmful report

- MR-5485

- Readme, VM and scripts to reproduce each harmful report:
  
  [Link](http://fcatch.cs.uchicago.edu/)
Other results in our paper

- Random fault injection
- Performance overhead
- Crash-point sensitivity
- ...

...
Outline

- Motivation
- Our TOF bug model
- FCatch tool
- Evaluation
- Conclusion
Take away

- TOF bugs are a timing problem.
Take away

- TOF bugs are a timing problem.
- Fault-aware logical time model.

- **Fault timing:** before W
- **Fault-tolerance:** timeout

- **Fault timing:** after W
- **Fault-tolerance:** Sanity check
TOF bugs are a timing problem.
Fault-aware logical time model.
FCatch detects TOF bugs from correct runs.

Fault timing: before W
Fault-tolerance: timeout

Fault timing: after W
Fault-tolerance: Sanity check
Crash-regular

- Fault timing: before W
- Fault-tolerance: timeout

Crash-recovery

- Fault timing: after W
- Fault-tolerance: Sanity check

Q&A

DCatch: Automatically Detecting Distributed Concurrency Bugs in Cloud Systems

http://fcatch.cs.uchicago.edu/
Thank you