Cloud Systems

- Microsoft Azure
- Amazon Web Services
- Dyn
- Hive
- Redis
- Apache HBase
- CouchDB
- Cassandra
- Amazon DynamoDB
- MongoDB
- Apache Spark
- Hadoop
Fault

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

Machine failures/updates in 29-day Google trace log\textsuperscript{[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
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
Cloud systems are fault tolerant?

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

Diagram: Merkle tree repair in Cassandra
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

![Diagram of Task attempt in MapReduce](image)

- Task 1 (TA1) and Task 2 (TA2) are running.
- AM (Application Master) monitors the tasks.
- CanCommit (ta1) message is sent from TA1 to AM.
- DoneCommit (ta2) message is sent from TA2 to AM.
- A fault occurs at node A, causing communication to hang.
- TA2's task attempt hangs due to the fault.
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] Leesatapornwonga. 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
State of the art – Model checking

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?
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;
}

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

```
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;
}
```

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

- 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?

- 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 model of TOF bugs
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?

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

A new fault-aware logical time model

New data flow at fault timing (a).

New data flow n at fault timing (n).
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
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
- Consider both synchronization and fault-tolerance operations

A correct run

YES!
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

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
**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

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

- 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
- 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

![Diagram of Traditional logical time model]

- $W_0$: No data flow
- $W$: HB
- $R$: Arrow pointing to $R$
Fault-aware logical time model

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

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

--- Crash VS. regular

New data flow between \textit{Ncrash} and \textit{Nregular} introduced by fault.
Fault-aware logical time model

-- Crash VS. recovery

Ncrash

Nrecovery

W

R
Fault-aware logical time model

-- Crash VS. recovery

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

-- Crash VS. recovery

\[ W_0 \rightarrow W \rightarrow R \rightarrow N_{\text{recovery}} \]
Fault-aware logical time model

-- Crash VS. recovery

![Diagram showing Ncrash and Nrecovery with data flow and R]
Fault-aware logical time model

-- Crash VS. recovery

Data flow between N\textit{crash} and N\textit{recovery} is totally determined by TOF.
Fault-aware logical time model

-- Crash VS. recovery

Fault-tolerance: sanity check

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

![Diagram showing the model with nodes W₀, W, and R, and data flows labeled as HB and New data flow, and Ncrash and Nregular labels.](image-url)
What are TOF bugs?
What are TOF bugs?

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

- Fault timing: before W

Crash-regular

Faulting:

- before W

Diagram:

- W
- HB
- R

N\text{crash} \quad N\text{regular}
What are TOF bugs?

- **Fault timing**: before W

![Diagram](image_url)

1. **Crash-regular**
2. **Merkle tree repair in Cassandra**

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

- Fault timing: before W
What are TOF bugs?

- **Fault timing:** before W

Crash-regular

Merkle tree repair in Cassandra

Fault timing: before W
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
What are TOF bugs?

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

- **Crash-recovery**
  - Fault timing: after W

Fault timing:
- **before W**: When the fault occurs before the write operation finishes, the system may not execute the recovery protocols in a timely manner, leading to potential data loss.
- **after W**: When the fault occurs after the write operation finishes, the system may still execute the recovery protocols, but the data that was written before the fault might be corrupted.
What are TOF bugs?

- Fault timing: after W
What are TOF bugs?

- 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

1. Produc correct runs
2. Identify conflicting ops
3. 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
Step 1: 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
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

Fault-free run

Faulty run
Step 2: identify conflicting operations

- Crash-regular TOF bug

- 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

- 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 showing the relationship between Replica, Primary, SnapshotRequest, and SnapshotReply.
Step 2: identify conflicting operations

- Crash-regular TOF bug
- Crash-recovery TOF bug
Step3: 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

```java
// 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

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

Fault-tolerance mechanism: timeout

Sol: statically check R’s bytecode.
Step 3: 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>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#. Bench</td>
<td>#. Unknown</td>
<td>#. FP</td>
<td>#. Bench</td>
<td>#. Unknown</td>
<td>#. FP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(harmful)</td>
<td>(harmful)</td>
<td></td>
<td>(harmful)</td>
<td>(harmful)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CA1&amp;2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HB-1</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td></td>
<td></td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HB-2</td>
<td></td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MR-1</td>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MR-2</td>
<td></td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ZK</td>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>10</td>
<td></td>
<td></td>
<td></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>
<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>
<td>#. FP</td>
</tr>
<tr>
<td>CA1&amp;2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>-</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>HB-1</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>-</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>HB-2</td>
<td>-</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>MR-1</td>
<td>-</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>MR-2</td>
<td>-</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>ZK</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Total</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>10</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

  - Create a HDFS flag-file before committing
    - Done

Readme, VM and scripts to reproduce each harmful report:

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
Take away

- 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/