DCatch: Automatically Detecting Distributed Concurrency Bugs in Cloud Systems

Haopeng Liu, Guangpu Li, Jeffrey Lukman, Jiaxin Li, Shan Lu, Haryadi Gunawi, and Chen Tian*
Cloud systems

The 10 Biggest Cloud Outages of 2016
www.crn.com/slide-shows/cloud/.../the-10-bigges
Dec 29, 2016 - The biggest cloud outages of 2016 incl
large, are increasingly vulnerable from downtime.
Distributed concurrency bugs (DCbugs)
Distributed concurrency bugs (DCbugs)

- Unexpected timing among distributed operations
Distributed concurrency bugs (DCbugs)

• Unexpected timing among distributed operations
• Example
Distributed concurrency bugs (DCbugs)

- Unexpected timing among distributed operations
- Example

MapReduce-3274
DCbugs need to be tackled

• Common in distributed systems [1, 2, 3]
  – 26% failures caused by non-deterministic [1]
  – 6% software bugs in clouds system [2]

[3] Leesatapornwongsa. TaxDC. In ASPLOS’16
DCbugs need to be tackled

- Common in distributed systems [1, 2, 3]
- Difficult to avoid, expose and diagnose

[3] Leesatapornwongsa. TaxDC. In ASPLOS’16
DCbugs need to be tackled

• Common in distributed systems [1, 2, 3]
• Difficult to avoid, expose and diagnose

[3] Leesatapornwongsa. TaxDC. In ASPLOS’16
DCbugs need to be tackled

• Common in distributed systems [1, 2, 3]
• Difficult to avoid, expose and diagnose

“There isn’t a week going by without new bugs about races.”

[3] Leesatapornwongsa. TaxDC. In ASPLOS’16
DCbugs need to be tackled

- Common in distributed systems [1, 2, 3]
- Difficult to avoid, expose and diagnose

```
Hadoop Map/Reduce / MAPREDUCE-3274
```

```
HBase / HBASE-4397
```

```
HBase / HBASE-6147
```

“We have already fix many cases, however it seems exist many other [racing] cases.”

[3] Leesatapornwongsa. TaxDC. In ASPLOS’16
DCbugs need to be tackled

• Common in distributed systems [1, 2, 3]
• Difficult to avoid, expose and diagnose

[3] Leesatapornwongsa. TaxDC. In ASPLOS’16
DCbugs need to be tackled

- Common in distributed systems [1, 2, 3]
- Difficult to avoid, expose and diagnose

[3] Leesatapornwongsa. TaxDC. In ASPLOS’16
DCbugs need to be tackled

• Common in distributed systems [1, 2, 3]
• Difficult to avoid, expose and diagnose

[3] Leesatapornwongsa. TaxDC. In ASPLOS’16

“Great catch, Sid! Apologies for missing the race condition.”
DCbugs need to be tackled

- Common in distributed systems [1, 2, 3]
- Difficult to avoid, expose and diagnose

Can we detect DCbugs before they manifest?

[3] Leesatapornwongsa. TaxDC. In ASPLOS’16
Previous work

• Model checking
  – Work on abstracted models
  – Face state-space explosion issue
Our idea

• Follow the philosophy of traditional concurrency bug detection
Our idea

• Follow the philosophy of traditional concurrency bug detection
Our idea

• Follow the philosophy of traditional concurrency bug detection
Our idea

• Follow the philosophy of traditional concurrency bug detection
Example

Get task (Repeat) → Cancel task
//RPC thread
Task getTask(jID){
    ...
    return jMap.get(jID);
}

//UnReg thread
void unReg(jID){
    jMap.remove(jID);
    ....
}
Local concurrency bug detection

ASPLOS Architectural Support for Programming

ASPLOS is a multi-disciplinary conference for researchers, compilers, languages, operating systems, networks, and engineers to present their latest research findings and computer systems innovations of the past two decades. Multiprocessors, clusters and networks-of-workstations

This conference occurs at a time when computer processor performance scaling and to new demands increasingly important as boundaries between hardware and capabilities of computing devices become.

Search within ASPLOS: "concurrency bug" "race"

Published Since 2006

Refine by Publication Year
Local concurrency bug detection
Local concurrency bug detection

**ASPLOS**

ASPLOS is a multi-disciplinary conference for researchers in compilers, languages, operating systems, networks, and engineers to present their latest research findings. It focuses on the innovations of the past two decades, such as multiprocessors, clusters, and networks-of-workstations.

This conference occurs at a time when computer processor performance scaling and to new demands increasingly important as boundaries between hardware and software capabilities of computing devices become.

Search within ASPLOS: "concurrency bug" "race"

**Is the problem solved?**
Local concurrency bug detection
Local concurrency bug detection

C1: How to handle the huge amount of mem accesses?
Local concurrency bug detection

C1: How to handle the huge amount of mem accesses?

Challenges
Local concurrency bug detection

C1: How to handle the huge amount of mem accesses?

Challenges
Local concurrency bug detection

C1: How to handle the huge amount of mem accesses?

C2: What’s the happens-before model?

Challenges
Local concurrency bug detection

C1: How to handle the huge amount of mem accesses?

C2: What’s the happens-before model?

Challenges
Local concurrency bug detection

C1: How to handle the huge amount of mem accesses?

C2: What’s the happens-before model?

C3: How to estimate the distributed impact of a race?
Local concurrency bug detection

C1: How to handle the huge amount of mem accesses?

C2: What’s the happens-before model?

C3: How to estimate the distributed impact of a race?
Local concurrency bug detection

C1: How to handle the huge amount of mem accesses?

C2: What’s the happens-before model?

C3: How to estimate the distributed impact of a race?

C4: How to trigger with distributed time manipulation?

Challenges
Contribution

- A comprehensive HB Model for distributed systems
Contribution

- A comprehensive HB Model for distributed systems
- DCatch tool detects DCbugs from correct runs

Challenges

C1: How to handle the huge amount of mem accesses?
C2: What’s the happens-before model?
C3: How to estimate the distributed impact of a race?
C4: How to trigger with distributed time manipulation?

Solved by DCatch
Contribution

• A comprehensive HB Model for distributed systems
• DCatch tool detects DCbugs from correct runs
• Evaluate on 4 systems
• Report 32 DCbugs, with 20 of them being truly harmful
Outline

• Motivation
• DCatch Happens-before Model
• DCatch tool
• Evaluation
• Conclusion
DCatch Happens-before Model

\[\text{HM} \text{Master} \]

\[\text{Th}d \]

\[\text{w} \]
DCatch Happens-before Model
DCatch Happens-before Model

HMaster

Thd

create

HRegionServer

Thd

RPC-call
DCatch Happens-before Model

**HMaster**

**HRegionServer**

Thd → create

RPC-call

Thd → EnQ

Event thread → e
DCatch Happens-before Model

- **HMaster**: create
- **HRegionServer**: Event thread
- **ZK Coordinator**: RPC-call, EnQ, Update
DCatch Happens-before Model

- **HM aster**
- **HRegionServer**

- **Thd**
- **Event thread**

- **Push**
- **RPC-call**
- **EnQ**
- **Update**

**ZK Coordinator**
DCatch Happens-before Model

- HMaster
  - Thd
  - Push
  - r

- Thd
  - create

- HRegionServer
  - Thd
  - Event thread
  - e
  - EnQ
  - Update

- RPC-call
- ZK Coordinator
DCatch Happens-before Model

Where is HB model for distributed systems?
DCatch Happens-before Model

- HMaster
  - Thd
  - Thd
  - Thd
  - Create
  - Push
  - ZK Coordinator

- HRegionServer
  - Thd
  - Eve thd
  - EnQ
  - Update

RPC-call
DCatch Happens-before Model

HMaster

Thd  Thd  Thd

HRegionServer

Thd  Eve thd

Dist.

r

create

Push

ZK Coordinator

w

EnQ

Update

Loc.  Dist.
DCatch Happens-before Model

HMaster

HRegionServer

ZK Coordinator

Sync.

Async.

Loc.

Dist.
DCatch Happens-before Model

- **HMaster**
  - Thd
  - W
  - Dist.
  - Cust.
  - ZK Coordinator

- **HRegionServer**
  - Thd
  - Eve thd
  - Async.
  - Update

**Graph:**
- Sync.
- Async.
- Cust.
- Stand.
- Loc.
- Dist.
Distributed rules

Local  Distributed

Sync.
Async.

Custom  Standard
Distributed rule #1

- Logical time clock (Leslie Lamport, 1978)
Distributed rule #1

- Logical time clock (Leslie Lamport, 1978)
Distributed rule #2

RPC

Asynch.    Synch.

Customize   Standard

RPC-call
RPC-rt

Machine 1

RPC-begin
RPC-end

Machine 2
Distributed rule #2

- **RPC**
- **Socket**
- **RPC-call** waiting
- **RPC-rt**
- **RPC-begin**
- **RPC-end**

- **Asynch.**
- **Synch.**

- Customize
- Standard
Distributed rule #2

RPC

Socket

Customize  Standard

RPC-call

waiting

RPC-rt

RPC-begin

RPC-end

Machine 1  Machine 2
Distributed rule #3

In multi-threaded systems:

```java
//Thread1
flag = True;
while(!flag){
}
...
```

```
//Thread
while(!flag){
}
```
Distributed rule #3

In multi-threaded systems:

```c
//Thread1
flag = True;
while(!flag){
}
...
```

```c
//Thread
while(!flag){
}
...
```
Distributed rule #3

In multi-threaded systems:

```c
//Thread1
flag = True;
/
/Thread
while(!flag){
}...
```

In distributed systems:

```c
//Thread
while(!flag){
}
```
Distributed rule #3

In distributed systems:

```cpp
//Thread1
flag = True;
```

```cpp
//Thread2
bool getFlag(){
    return flag;
}
```

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>RPC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Socket</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Customize** Standard

Dist. while-loop
Distributed rule #3

In distributed systems:

//Thread1
flag = True;

while(!getFlag()){
    ...
}

//Thread2
bool getFlag(){
    return flag;
}

Machine A

Machine B

Dist. while-loop

Synch.

Asynch.

RPC

Socket

Customize Standard
Distributed rule #4

ZooKeeper Service

Asynch.  Synch.

RPC
Socket

Customize  Standard

RPC
create
Push
RPC-call
EnQ
Update

ZK Coordinator
Distributed rule #4

ZooKeeper Service

Asynch. | Synch.
---|---

Customize | Standard

RPC

Socket

ZK Coordinator

RPC-call

EnQ

Update

Push

create

Thd

Thd

Thd
Distributed rules

<table>
<thead>
<tr>
<th>Synchronous</th>
<th>Asynchronous</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dist. While-loop</td>
<td>Zookeeper Service</td>
</tr>
<tr>
<td>Hadoop</td>
<td>HBase</td>
</tr>
<tr>
<td>RPC</td>
<td>Socket</td>
</tr>
<tr>
<td>Customized</td>
<td>Standard</td>
</tr>
</tbody>
</table>

- Dist. While-loop
- Zookeeper Service
- Hadoop
- HBase
- RPC
- Socket
- Customized
- Standard
Local rules

Local  Distributed
Local rules

<table>
<thead>
<tr>
<th>Synchronous</th>
<th>Asynchronous</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>n/a</td>
<td>Event-related</td>
</tr>
<tr>
<td>Customized</td>
<td>Standard</td>
</tr>
</tbody>
</table>
Local rules

<table>
<thead>
<tr>
<th></th>
<th>Synchronous</th>
<th>Asynchronous</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>While-loop</strong></td>
<td>n/a</td>
<td>Event-related</td>
</tr>
<tr>
<td><strong>Thread fork/join</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Local rules

<table>
<thead>
<tr>
<th>Synchronous</th>
<th>Asynchronous</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>While-loop</strong></td>
<td><strong>n/a</strong></td>
</tr>
<tr>
<td><strong>Thread fork/join</strong></td>
<td><strong>Event-related</strong></td>
</tr>
</tbody>
</table>

- **Customized**
- **Standard**
Outline

• Motivation
• DCatch Happens-before Model
• DCatch tool
• Evaluation
• Conclusion
C1: How to handle the huge amount of mem accesses?

C2: What’s the happens-before model?

C3: How to estimate the distributed impact of a race?

C4: How to trigger with distributed time manipulation?

Selective tracing: only mem accesses in Event/message handlers and their callee.
C1: How to handle the huge amount of mem accesses?

C2: What’s the happens-before model?

C3: How to estimate the distributed impact of a race?

C4: How to trigger with distributed time manipulation?

Challenges

C1: How to handle the huge amount of mem accesses?

C2: What’s the happens-before model?

C3: How to estimate the distributed impact of a race?

C4: How to trigger with distributed time manipulation?

---

C1: How to handle the huge amount of mem accesses?

C2: What’s the happens-before model?

C3: How to estimate the distributed impact of a race?

C4: How to trigger with distributed time manipulation?

---

Machine B

```java
//RPC thread
Task getTask(jID){
    ...
    return jMap.get(jID);
}
```

```java
//UnReg thread
void unReg(jID){
    jMap.remove(jID);
    ....
}
```
C1: How to handle the huge amount of mem accesses?

C2: What’s the happens-before model?

C3: How to estimate the distributed impact of a race?

C4: How to trigger with distributed time manipulation?

**Challenges**

```
while(!getTask(jID)) {
    Task getTask(jID) {
        ...
        return jMap.get(jID);
    }
    //RPC thread
    //UnReg thread
    void unReg(jID) {
        jMap.remove(jID);
        ...
    }
}
```
C1: How to handle the huge amount of mem accesses?

C2: What's the happens-before model?

C3: How to estimate the distributed impact of a race?

C4: How to trigger with distributed time manipulation?

### Challenges

#### Machine A

```java
while(!getTask(jID)) {
}
```

#### Machine B

```java
// RPC thread
Task getTask(jID) {
    ...
    return jMap.get(jID);
}
```

```java
// UnReg thread
void unReg(jID) {
    jMap.remove(jID);
    ....
}
```
C1: How to handle the huge amount of mem accesses?

C2: What’s the happens-before model?

C3: How to estimate the distributed impact of a race?

C4: How to trigger with distributed time manipulation?

Challenges

Machine A

while(!getTask(jID)){
    
}

Machine B

//RPC thread
Task getTask(jID){
    ...
    return jMap.get(jID);
}

//UnReg thread
void unReg(jID){
    jMap.remove(jID);
    ....
}
C1: How to handle the huge amount of mem accesses?

C2: What’s the happens-before model?

C3: How to estimate the distributed impact of a race?

C4: How to trigger with distributed time manipulation?
C1: How to handle the huge amount of mem accesses?
C2: What’s the happens-before model?
C3: How to estimate the distributed impact of a race?
C4: How to trigger with distributed time manipulation?

Challenges
C1: How to handle the huge amount of mem accesses?

C2: What’s the happens-before model?

C3: How to estimate the distributed impact of a race?

C4: How to trigger with distributed time manipulation?
C1: How to handle the huge amount of mem accesses?

C2: What’s the happens-before model?

C3: How to estimate the distributed impact of a race?

C4: How to trigger with distributed time manipulation?

---

**Challenges**

**Trace**

**HB**

**Triage**

**Trigger**

Machine A

Thd

RPC-call

Machine B

Thd

Event thread

EnQueue

r

e1

EnQueue

w

e2

RPC-call

Machine C

Thd
C1: How to handle the huge amount of mem accesses?

C2: What’s the happens-before model?

C3: How to estimate the distributed impact of a race?

C4: How to trigger with distributed time manipulation?

Challenges
C1: How to handle the huge amount of mem accesses?

C2: What’s the happens-before model?

C3: How to estimate the distributed impact of a race?

C4: How to trigger with distributed time manipulation?
Outline

• Motivation
• DCatch Happens-before Model
• DCatch tool
• Evaluation
• Conclusion
Methodology

• Benchmarks:
  – 7 real-world DCbugs from TaxDC[1]
  – 4 distributed systems

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

<table>
<thead>
<tr>
<th>BugID</th>
<th>Detected?</th>
<th>#. Bugs</th>
<th>#. Benign</th>
<th>#. false-pos</th>
</tr>
</thead>
<tbody>
<tr>
<td>CA-1011</td>
<td>✔️</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>HB-4539</td>
<td>✔️</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>HB-4729</td>
<td>✔️</td>
<td>4</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>MR-3274</td>
<td>✔️</td>
<td>2</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>MR-4637</td>
<td>✔️</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>ZK-1144</td>
<td>✔️</td>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ZK-1270</td>
<td>✔️</td>
<td>6</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>
## Overall results

<table>
<thead>
<tr>
<th>BugID</th>
<th>Detected</th>
<th>#. Bugs</th>
<th>#. Benign</th>
<th>#. false-pos</th>
</tr>
</thead>
<tbody>
<tr>
<td>CA-1011</td>
<td>✔️</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>HB-4539</td>
<td>✔️</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>HB-4729</td>
<td>✔️</td>
<td>4</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>MR-3274</td>
<td>✔️</td>
<td>2</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>MR-4637</td>
<td>✔️</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>ZK-1144</td>
<td>✔️</td>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ZK-1270</td>
<td>✔️</td>
<td>6</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>
## Overall results

<table>
<thead>
<tr>
<th>BugID</th>
<th>Detected?</th>
<th>#. Bugs</th>
<th>#. Benign</th>
<th>#. false-pos</th>
</tr>
</thead>
<tbody>
<tr>
<td>CA-1011</td>
<td>✔</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>HB-4539</td>
<td>✔</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>HB-4729</td>
<td>✔</td>
<td>4</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>MR-3274</td>
<td>✔</td>
<td>2</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>MR-4637</td>
<td>✔</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>ZK-1144</td>
<td>✔</td>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ZK-1270</td>
<td>✔</td>
<td>6 = 12 + 8</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>
# Overall results

<table>
<thead>
<tr>
<th>BugID</th>
<th>Detected?</th>
<th>#. Bugs</th>
<th>#. Benign</th>
<th>#. false-pos</th>
</tr>
</thead>
<tbody>
<tr>
<td>CA-1011</td>
<td>✔️</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>HB-4539</td>
<td>✔️</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>HB-4729</td>
<td>✔️</td>
<td>4</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>MR-3274</td>
<td>✔️</td>
<td>2</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>MR-4637</td>
<td>✔️</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>ZK-1144</td>
<td>✔️</td>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ZK-1270</td>
<td>✔️</td>
<td>6</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>
Other results in our paper

• Performance overhead
• Trace compositions
• HB model impact
  – False-positive
  – False-negatives
• ...

Outline

• Motivation
• DCatch Happens-before Model
• DCatch tool
• Evaluation
• Conclusion
Conclusion

- **A HB Model** for distributed systems

- **DCatch** detects DCbugs from **correct runs** with **low false positive** rates.
Thank you!

Q&A