Understanding and Detecting Timing Bugs in Cloud Systems

Haopeng Liu, Shan Lu

UCHICAGO

Background & Motivation

1. Timing bugs in distributed systems:

- Unexpected timing among dist. events:
  - -- Message: time-of-execution bugs
  - -- Fault: time-of-fault bugs
- Common in dist. systems [1, 2]
- Difficult to avoid and tackle
- Little tool support

2. State of the art – Model checking

Challenge: Complex manual specifications

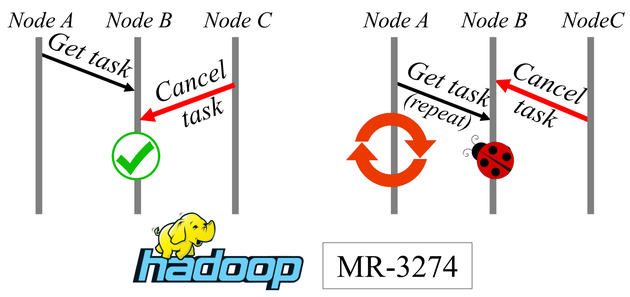
Q1: Can we judge what are timing bugs without manual specifications?

2. State of the art – Random fault injection

Challenge: Many fault injection runs

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

Time of execution (TOE) bugs — ASPLOS-2017



MR-3274

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

//UnReg thread
void cancel(jID){
  jMap.remove(jID);
  ....
}
```

Key insights: opportunities

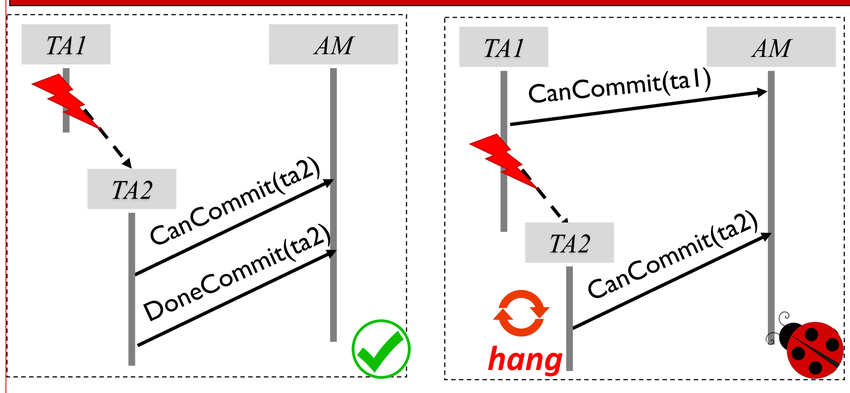
- **Distributed** timing causes conflicting accesses on **one** machine

→ Can we apply local concurrency bug detection techniques?

Key insights: challenges to detect TOE bugs

- What's the Happens-Before model?
- How to handle the huge # of accesses?
- How to manipulate distributed timing?
- How to estimate distributed impact?

Time of fault (TOF) bugs — ASPLOS-2018




```
Boolean CanCommit(taID) {          TA1
  ...
  T.commitID = taID;
}

Boolean CanCommit(taID) {          TA2
  if (T.commitID) {
    return T.commitID == taID;
  }
}
```

MR-5476

Key insights

A new model of TOF bug

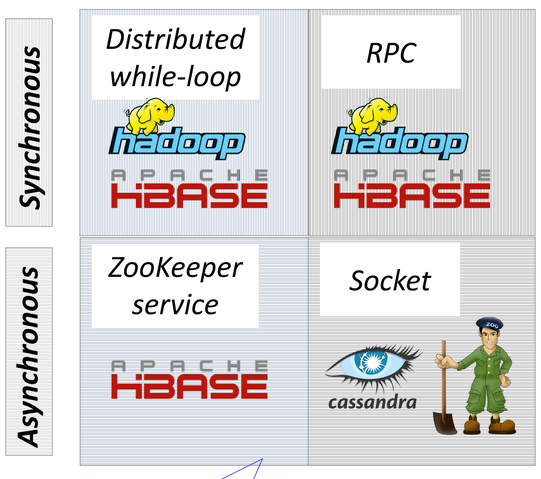
- Conflicting accesses on shared states (heap, file, ...)
- From a crash node and non-crash nodes
- Data dependency changes with TOF change

A new fault-aware logical time model

- Predict data flow changes caused by TOF changes
- Consider both synch. and fault-tolerance ops

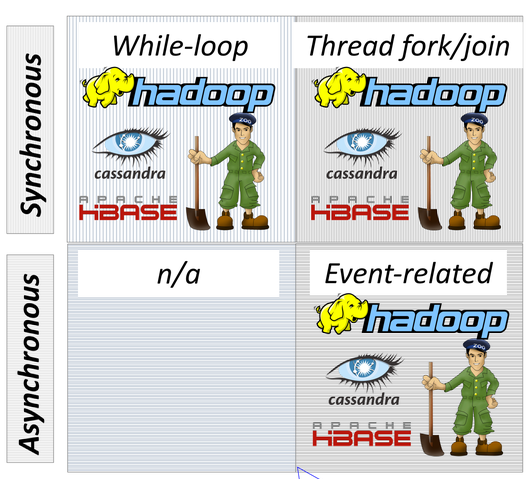
Logical time model for Distributed Systems

Distributed HB Rules



- Rule-M^{rpc}: *Crt(r, n1)* → *Begin(r, n2)*
- Rule-M^{rpc}: *End(r, n2)* → *Join(r, n1)*
- Rule-M^{soc}: *Send(m, n1)* → *Recv(m, n2)*
- Rule-M^{pull}: *Update(s, n1)* → *Pull(s, n2)*
- Rule-M^{push}: *Update(s, n1)* → *Push(s, n2)*

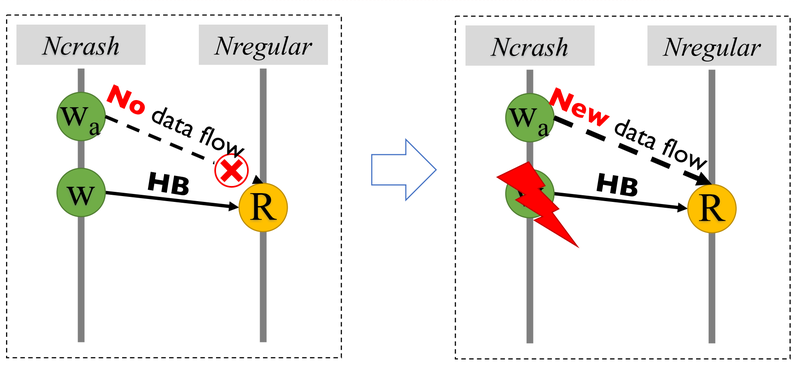
Local HB Rules



- Rule-T^{fork}: *Create (t)* → *Begin (t)*
- Rule-T^{join}: *End (t)* → *Join (t)*
- Rule-E^{enq}: *Create(e)* → *Begin(e)*
- Rule-E^{serial}: *End (e1)* → *Begin(e2) if...*
- Rule-T^{pull}: *Update (f)* → *Pulled (f)*

Fault-aware logical time model

1. What is new data flow introduced by timing of faults?

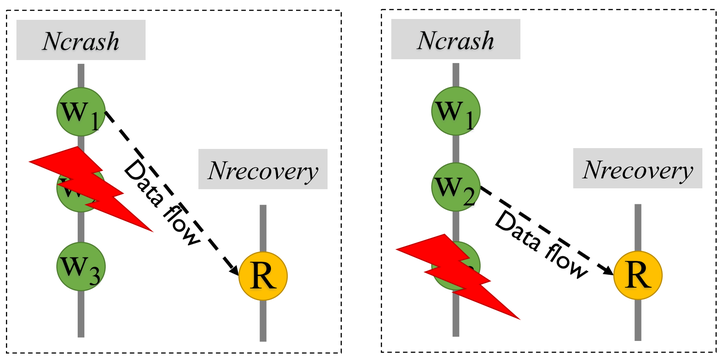


Crash-regular TOF bug

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

```
obj.wait(long timeout); //R
```

2. How about data flow between recovery and crash nodes?



Data flow is totally determined by TOF!

Crash-recovery TOF bug

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

```
//Recovery node
if (f.valid()) { //R1: sanity check
  dt = f.read(); //R2: read
}
```

DCatch: predict TOE bugs

Overview: a. Customized for distributed systems and TOE bugs;
b. Aims **S**calability, **C**overage, and **A**ccuracy.

1. Runtime Tracing [S, C, A]

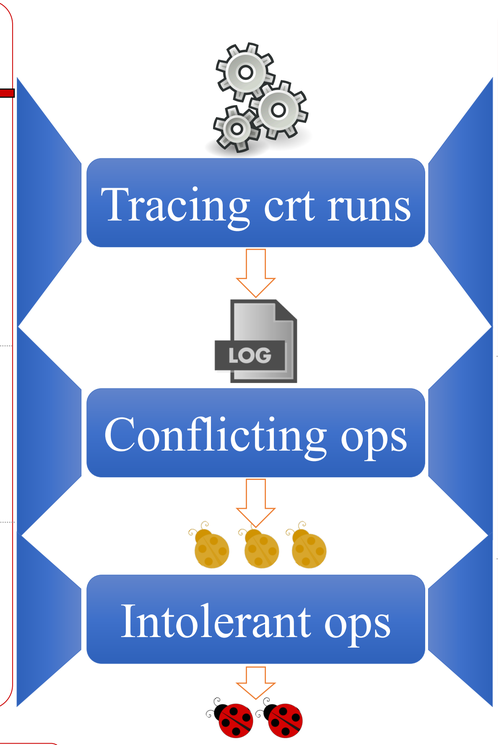
- Heap accesses related to dist. computation/communication
- Happens-before related operations following HB model

2. HB Analysis [S, C, A]

- HB graph construction following the HB model above
- Data race detection from HB graph[3]

3. Static Pruning [S, A]

- Statically estimate distributed & local impact of each race
- Prune out races unlikely to cause failures



FCatch: predict TOF bugs

Q. Only fault-free correct run is enough?

Crash-regular TOF bug

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

Crash-recovery TOF bug

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

Crash-regular TOF bugs:

- Analyze fault-free run trace
- W & R are from different nodes
- W & R have HB relation

Crash-recovery TOF bugs:

- Analyze fault-free &faulty traces
- W is from *Ncrash* in the fault-free
- R is from *Nrecovery* in the faulty

Crash-regular TOF bugs:

- Timeout (statically check R)

Crash-recovery TOF bugs:

- Sanity check + impact analysis

Evaluation

DCatch	CA	HB-1	HB-2	MR-1	MR-2	ZK-1	ZK-2	Total
Detected?	✔	✔	✔	✔	✔	✔	✔	
#. Harmful bugs	3	3	4	2	1	5	6	20
#. Benign bugs	0	0	1	0	2	1	2	5
#. False positives	0	1	0	4	4	1	0	7

Include 8 new bugs

Evaluation

FCacth	CA1&2	HB-1	HB-2	MR-1	MR-2	ZK	Total
#. Bench (harmful)	2 + /	1 + /	/ + 1	/ + 1	/ + 2	/ + 1	3 + 5
#. Unknown (harmful)	1 + 0	0 + 0	2 + 2	1 + 1	1 + 1	0 + 0	4 + 4
#. False positives	0 + 2	3 + 6	2 + 0	0 + 0	0 + 0	0 + 2	5 + 10

More details: http://fcatch.cs.uchicago.edu/

[1]. T. Leesatapornwongsa, J. Lukman, S. Lu, and H. Gunawi. TaxDC: A Taxonomy of Non-Deterministic Concurrency Bugs in Datacenter Distributed Systems. In ASPLOS, 2016
[2]. Zhenyu Guo, Sean McDirmid, Mao Yang, Li Zhuang, Pu Zhang, Yingwei Luo,Tom Bergan, Peter Bodik, Madan Musuvathi, Zheng Zhang, and Lidong Zhou.Failure recovery: When the cure is worse than the disease. In HotOS, 2013
[3]. V. Raychev, M. Vechev, and M. Sridharan. Effective Race Detection for Event-Driven Programs. In OOPSLA, 2013