Towards Pre-Deployment Detection of Performance Failures in Cloud Systems

Riza Suminto, Agung Laksono*, Anang Satria*, Thanh Do†, Haryadi Gunawi
Cloud Systems

Cassandra
A highly scalable, eventually consistent, distributed, structured key-value store.

redis

Hadoop

MongoDB

CouchDB

Amazon DynamoDB

Hive
Demands

- Users demand high dependability, reliability, and performance stability
- Amazon found that every 100ms of latency cost them 1% in sales
- Google found an extra 0.5 second in search page generation time dropped traffic by 20%

Speed Matters!
Performance failures happen

What Bugs Live in the Cloud? A Study of 3000+ Issues in Cloud Systems, SOCC’14
Performance Bug

- Jobs take multiple times than usual to finish
  - Improper speculative execution
    - JCH₁ & TPL₁ & FPL₂ & FTY₁
  - Unnecessary repeated recovery
    - TPL₁ & TPL₄ & FTY₄ & TOP₁

<table>
<thead>
<tr>
<th>Scenario Type</th>
<th>Possible Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>DLC: Data Locality</td>
<td>(1) Read from remote disk, (2) read from local disk, ...</td>
</tr>
<tr>
<td>DSR: Data Source</td>
<td>(1) Some tasks read from same datanode, (2) all tasks read from different datanodes, ...</td>
</tr>
<tr>
<td>JCH: Job Characteristic</td>
<td>Map-reduce is (1) many-to-all, (2) all-to-many, (3) large fan-in, (4) large fan-out, ...</td>
</tr>
<tr>
<td>JSZ: Job Size</td>
<td>(1) 1 GB jar file, (2) 1 MB jar file, ...</td>
</tr>
<tr>
<td>LSZ: Load Size</td>
<td>(1) Thousands of tasks, (2) small number of tasks, ...</td>
</tr>
<tr>
<td>FTY: Fault Type</td>
<td>(1) Slow node/NIC, (2) Node disconnect/packet drop, (3) Disk error/out of space, (4) Rack switch, ...</td>
</tr>
<tr>
<td>FPL: Fault Placement</td>
<td>Slowdown fault injection at the (1) source datanode, (2) mapper, (3) reducer, ...</td>
</tr>
<tr>
<td>FGR: Fault Granularity</td>
<td>(1) Single disk/NIC, (2) single node (deadnode), (3) entire rack (network switch), ...</td>
</tr>
<tr>
<td>FTM: Fault Timing</td>
<td>(1) During shuffling, (2) during 95% of task completion, ...</td>
</tr>
<tr>
<td>TOP: Topology Scenario</td>
<td>(1) 30 nodes per rack, (2) 3 nodes per rack, ...</td>
</tr>
<tr>
<td>TPL: Task Placement</td>
<td>(1) Mappers and reducers are in different nodes, (2) AM and reducers in different nodes, (3) Mappers are in the same node, (4) Most of reducers placed in the same rack, ...</td>
</tr>
</tbody>
</table>
Untriggered SpecExec

**DLC**<sub>A</sub> Map read locally

**TPL**<sub>A</sub> Mappers and reducers in different nodes

**JCH**<sub>A</sub> All-to-All

**FPL**<sub>A</sub> Fault at map node

**FTY**<sub>A</sub> Slow NIC

All reducers slow!

No straggler = No SpecExec

DLC<sub>A</sub> & TPL<sub>A</sub> & JCH<sub>A</sub> & FPL<sub>A</sub> & FTY<sub>A</sub>
Untriggered SpecExec, cont

- DLC_A & TPL_A & JCH_A & FPL_A & FTY_A

Mappers

- M1
- M2
- M3

Reducers

- Reducer

- Reducer

- Reducer

- Reducer

DN

Mappers

- M1

- M2

- M3

DLC_B = read remote

Straggler!
Untriggered SpecExec, cont

- $\text{DLC}_A \& \text{TPL}_A \& \text{JCH}_A \& \text{FPL}_A \& \text{FTY}_A$
- $\text{DLC}_A \& \text{TPL}_A \& \text{JCH}_A \& \text{FPL}_B \& \text{FTY}_A$

**Mapper** $\text{M1}$, $\text{M2}$, $\text{M3}$ to **Reducer**

Slow reducer $\Rightarrow$ Straggler!

**Mapper** $\text{M1}$, $\text{M2}$, $\text{M3}$ to **Reducer**
O(n) Recovery

\[ TPL_A \] Mappers and Reducers in different nodes

\[ TPL_B \] Mappers and Reducers in different racks

\[ TOP_A \] Large number of nodes per rack

\[ FTY_B \] Slow inter-rack switch

\[ TPL_A \] & \[ TPL_B \] & \[ TOP_A \] & \[ FTY_B \]

Rack 1 \[ \rightarrow \] \[ R \]

Rack 2

Slow!
Conditions lead to performance bug

- **Untriggered Speculative Execution**
  - MR-70001 = JCH₁ & TPL₁ & FPL₂ & FTY₁
  - MR-70002 = DSR₁ & DLC₁ & FPL₁ & FTY₁
  - MR-5533 = FTY₂ & FPL₃ & TPL₃
  - ...

- **O(n) Recovery**
  - MR-525₁ = FTY₃ & FPL₃ & FTM₁
  - MR-5060 = TPL₁ & TPL₃ & FTY₁ & FPL₂
  - MR-1800 = TPL₁ & TPL₄ & FTY₄ & TOP₁
  - ...

- **Long lock contention**
  - MR-919₁ = FTY₃ & FPL₃ & FTM₁
  - MR-929₂ = TPL₁ & TPL₃ & FTY₁ & FPL₂
  - MR-939₃ = TPL₁ & TPL₄ & FTY₄ & TOP₁
  - ...

<table>
<thead>
<tr>
<th>Scenario Type</th>
<th>Possible Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>DLC: Data Locality</td>
<td>(1) Read from remote disk, (2) read from local disk, ...</td>
</tr>
<tr>
<td>DSR: Data Source</td>
<td>(1) Some tasks read from same datanode, (2) all tasks read from different datanodes, ...</td>
</tr>
<tr>
<td>JCH: Job Characteristic</td>
<td>Map-reduce is (1) many-to-all, (2) all-to-many, (3) large fan-in, (4) large fan-out, ...</td>
</tr>
<tr>
<td>JSZ: Job Size</td>
<td>(1) 1GBjarfile, (2) MBjarfile, ...</td>
</tr>
<tr>
<td>LSZ: Load Size</td>
<td>(1) Thousands of tasks, (2) small number of tasks, ...</td>
</tr>
<tr>
<td>FTY: Fault Type</td>
<td>(1) Slow node/NIC, (2) Node disconnect/packet drop, (3) Disk error/out of space, (4) Rack switch, ...</td>
</tr>
<tr>
<td>FPL: Fault Placement</td>
<td>Slowdown fault injection at the (1) source datanode, (2) mapper, (3) reducer, ...</td>
</tr>
<tr>
<td>FGR: Fault Granularity</td>
<td>(1) Single disk/NIC, (2) single node (deadnode), (3) entire rack (network switch), ...</td>
</tr>
<tr>
<td>FTM: Fault Timing</td>
<td>(1) During shuffling, (2) during 95% of task completion, ...</td>
</tr>
<tr>
<td>TOP: Topology</td>
<td>(1) 30 nodes per rack, (2) 3 nodes per rack, ...</td>
</tr>
<tr>
<td>TPL: Task Placement</td>
<td>(1) Mappers and reducers are in different nodes, (2) AM and reducers in different nodes, (3) Mappers are in the same node, (4) Most of reducers placed in the same rack, ...</td>
</tr>
</tbody>
</table>
Outline

Performance Bug

System Performance Verifier
Current Approach

- Benchmarking
- **Hundreds** benchmark for every scenario
- Injecting slowdowns and failures
- Take **days to weeks!!**
What we want...

- Four goals in performance verification
  - Fast
  - Covers many deployment scenario
  - Runs in pre-deployment
  - Directly checks implementation code

Formal modeling tools!
System Performance Verifier (SPV)

Target system (e.g., Hadoop code)

Auto-generated model (in Colored Petri Net)

Performance Verification
Colored Petri Nets (CPN)

```plaintext
Tasks

(input(node,task);
output(assignment);
action
  let val (id,type) = task
  in
    (node,id,type)
end;

Node

Schedule Task

Task to Run

(A,"T1",map) @10

A

node

node

A

node

Task

(“T1”,map)

@+10

@0

@10

SPV @ HotCloud '15
Challenges: Two Different World

CPN

Java

```java
public synchronized void launchTask(RunningJob rjob) throws IOException {
    if (this.taskStatus.getRunState() == TaskStatus.State.UNASSIGNED) {
        this.taskStatus.setRunState(TaskStatus.State.FAILED_UNCLEAN);
        this.taskStatus.setRunState(TaskStatus.State.KILLED_UNCLEAN);
        localizableTask(task);
    } else if (this.taskStatus.getRunState() == TaskStatus.State.UNASSIGNED) {
        this.taskStatus.setRunState(TaskStatus.State.RUNNING);
    }
    setTaskRunner(task.createRunner(TaskTracker.this, this, rjob));
    this.runner.start();
}

long now = System.currentTimeMillis();
this.taskStatus.setStartTime(now);
this.lastProgressReport = now;
}
```
Our Approach

- Java ➔ SysJava
  - Data flattening
  - Code modularization
  - Annotation tagging

- SysJava ➔ Model compiler
Data Flattening

- Java system states = ArrayList, Map, Tree, ...
- CPN states = multisets

List&lt;JobInProgress&gt; runningJobs;

public class JobInProgress {
    JobID jobId;
    TaskInProgress maps[];
    ...
}

class TaskInProgress {
    TaskID id;
    double progress;
    ...
}
# Code Modularization

## Modular function

```java
@ProcessState
private void initCheck() {
    synchronized (taskTrackers) {
        ...
    }
}
```

## Control Flow logic

```java
@ForEach
private void updateStatuses(
    TaskTrackerStatus trackerStats) {
    for (TaskStatus ts: trackerStats) {
        tasks.get(ts.id).updateStatus(ts);
    }
    ...
}
```

## CRUD Logic

```java
@GetState
private TaskInProgress getTask(TaskID id) {
    tasks.get(ts.id);
}
@UpdateState
private void tipUpdate(TaskInProgress tip, TaskStatus ts) {
    tip.updateStatus(ts);
}
```
Annotation Tagging

- Assist compiler
- Annotation Category:
  - Data Structure
  - I/O
  - CRUD & Process
  - Miscellaneous

```java
public class JobInProgress {
    JobID jobId;
    TaskInProgress maps[];
    ...
}

@Data
public class JobInProgress {
    JobID jobId;
    TaskInProgress maps[];
    ...
}

@IO
public HeartbeatResponse heartbeat (HeartbeatData hd) {
    ...
}
```
Model Checking

- SPV Compiler ➔ Executable XML
- Define configurations, assertions, and specifications
- Explore every non-deterministic choices
  - Task to node mapping
Preliminary Result

- 5305 lines of code on top of WALA & Access/CPN
- Hadoop MapReduce 1.2.1, with 1067 lines code change
- 20x larger than hand-made model
- 34 scenario, 30 assertion violation, 4 performance bug
- 1.5 hour model checking

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Worker Node</td>
<td>Node A, B</td>
</tr>
<tr>
<td>Data Node</td>
<td>Node A, B, C</td>
</tr>
<tr>
<td>Tasks</td>
<td>2 Task</td>
</tr>
<tr>
<td>Fault Type</td>
<td>Slow Data Node</td>
</tr>
<tr>
<td>Fault Placement</td>
<td>Node C</td>
</tr>
</tbody>
</table>
Thank you! Questions?

http://ucare.cs.uchicago.edu
Discussion

- Is it time for pre-deployment detection of performance bugs?
- Bridging system code and formal methods
- Future of data-centric languages
- Beyond Hadoop
- Root cause anatomy of performance bugs
- Beyond performance bugs