Finding Heterogeneous-Unsafe Configuration Parameters in Cloud Systems

Sixiang Ma, Fang Zhou, Michael D. Bond, Yang Wang
The Ohio State University
Heterogeneous Configurations Are Prevalent

• Heterogeneous hardware calls for heterogeneous configuration

• Online reconfiguration, e.g., reconfig command, rolling restart
  • Consequence: short window of heterogeneous configuration
Heterogeneous Configuration Can Cause Error

• Errors can happen even if each node has valid configuration locally.

HomoConf(F1) is valid
HomoConf(F2) is valid
HeterConf(F1,F2) is invalid
Heterogeneous Configuration Can Cause Error

- Errors can happen even if each node has valid configuration locally.

We call $\text{HeterConf}(F_1, F_2)$ **Invalid Heterogeneous Configuration**, if it causes errors but $\text{HomoConf}(F_1)$ and $\text{HomoConf}(F_2)$ do not.

- $\text{HomoConf}(F_1)$ is valid
- $\text{HomoConf}(F_2)$ is valid
- $\text{HeterConf}(F_1, F_2)$ is invalid
Errors can happen even if each node has valid configuration locally. We call HeterConf(F1, F2) **Invalid Heterogenous Configuration**, if it causes errors but HomoConf(F1) and HomoConf(F2) do not.

We call the corresponding parameter **Heterogenous-Unsafe Configuration Parameter**.
Example: `dfs.datanode.balance.bandwidthPerSec`

- Specify the maximum amount of bandwidth that a HDFS DataNode can use for balancing purpose.
Example: `dfs.datanode.balance.bandwidthPerSec`

- Specify the maximum amount of bandwidth that a HDFS DataNode can use for balancing purpose.

```
100 MB/s
DataNode 1 transfer data 100 MB/s
DataNode 2

Balancer

report progress
```
Example: `dfs.datanode.balance.bandwidthPerSec`

- Specify the maximum amount of bandwidth that a HDFS DataNode can use for balancing purpose.
Example: `dfs.datanode.balance.bandwidthPerSec`

- Specify the maximum amount of bandwidth that a HDFS DataNode can use for balancing purpose.

100 MB/s

DataNode 1

transfer data

10 MB/s

DataNode 2

Balancer

report progress
Related Work

This type of errors is different from the problem of erroneous configuration values [EnCore-ASPLOS’14, ConfValley-EuroSys’15, PCheck-OSDI’16, PracExtractor-ATC’20]

- Parameter values are valid.
- Errors happen when nodes communicate.
Overview

• Our goal: find heterogeneous-unsafe configuration parameters in cloud systems.

• ZebraConf: a testing framework that reuse existing unit tests

• It finds 41 true problems in HDFS, YARN, MR, HBase, Flink.
ZebraConf Uses Classic Software Testing Approach

• Challenge: some parameters may only take effect under specific workloads.

• Observation: mature cloud systems usually have rich unit tests.
  • High code coverage [Kairux-SOSP’19]
    • E.g., 90.1% statement coverage in HDFS
  • Many unit tests are using configuration
    • 3,628 unit tests in HDFS use config, covering 96.2% parameters
Reusing Existing Unit Tests for Our Purpose

- E.g., 90.1% statement coverage in HDFS
- Many unit tests are using configuration
  - 3,628 unit tests in HDFS use config, covering 96.2% parameters
ZebraConf: Major Challenges
ZebraConf: Major Challenges

• C1: How to reduce testing time?
  • Apps can have 1000s of tests, 100s-1000s of parameters.
  • A test runs for several seconds to several minutes.
ZebraConf: Major Challenges

- C1: How to reduce testing time?
  - Apps can have 1000s of tests, 100s-1000s of parameters.
  - A test runs for several seconds to several minutes.

- C2: How to assign heterogeneous configuration in unit tests?
  - We can specify the config when starting a node as process.
    - E.g., hadoop-daemon.sh --config [CONFIG_PATH] start
  - However, this approach doesn’t work in unit tests.
ZebraConf Overview

TestGen: generate test instances

TestRunner: conduct a test

ConfAgent: assign configs to nodes
ZebraConf Overview

TestGen: generate test instances
- Selective value assignment
- Pre-run profiling
- Pooled tests

TestRunner: conduct a test
- Supporting pooled testing
- Concurrent testing
- Hypothesis testing

ConfAgent: assign configs to nodes

C1: How to reduce testing time

C2: How to assign heter config in unit tests
ZebraConf Overview

TestGen: generate test instances
- Selective value assignment
- Pre-run profiling
- Pooled tests

TestRunner: conduct a test
- Supporting pooled testing
- Concurrent testing
- Hypothesis testing

ConfAgent: assign configs to nodes

C1: How to reduce testing time

C2: How to assign heter config in unit tests
ZebraConf Overview

**Parameters**

- TestGen: generate test instances
  - Selective value assignment
  - Pre-run profiling
  - Pooled tests

- TestRunner: conduct a test
  - Supporting pooled testing
  - Concurrent testing
  - Hypothesis testing

C1: How to reduce testing time

C2: How to assign heter config in unit tests

ConfAgent: assign configs to nodes
TestGen: Dimensions in Test Instances

Test instance = Unit test + HeterConf(F1, F2)
TestGen: Dimensions in Test Instances

Test instance = Unit test + Assignment of F1, F2 to nodes
TestGen: Dimensions in Test Instances

Test instance = Unit test + Assignment of F1, F2 to nodes

\[ u \times \binom{2^p}{p} \times 2^n \]

where
- \( u \) is the number of unit tests,
- \( p \) is the number of paras,
- if each para has 2 values,
- if each test has \( n \) nodes.
TestGen: Dimensions in Test Instances

Test instance = Unit test + Assignment of F1, F2 to nodes

• F1, F2 $\rightarrow$ a parameter with two values

• Value-to-node assignment $\rightarrow$ rep. assignment for node type

$u \times \binom{2^p}{2} \times 2^n$, where $u$ is # of unit tests, $p$ is # of paras, if each para has 2 values, if each test has $n$ nodes

$u \times 2p \times 3t$, where $t$ is # of node types, if each para has 2 values, max # of rep. assignment is 3
TestGen: Dimensions in Test Instances

Test instance = Unit test + Assignment of F1, F2 to nodes

• F1, F2 $\rightarrow$ a parameter with two values

• Value-to-node assignment $\rightarrow$ rep. assignment for node type

Test number is still very large:
For HDFS, $u=6175$, $p=914$, $t=6$, there will be 387 millions of tests instances.

$u \times \binom{2^p}{2} \times 2^n$, where
- $u$ is # of unit tests,
- $p$ is # of paras,
- if each para has 2 values,
- if each test has $n$ nodes

$u \times 2p \times 3t$, where
- $t$ is # of node types,
- if each para has 2 values,
- max # of rep. assignment is 3
TestGen: Dimensions in Test Instances

Test instance = \textbf{Unit test} + Assignment of F1, F2 to nodes

• F1, F2 $\rightarrow$ a \textbf{parameter} with two values

• Value-to-node assignment for node type

Test number is still very large:

For HDFS, $u=6175$, $p=914$, $t=6$, there will be 387 millions of tests instances.

Many combinations can be ineffective when testing.

$u \times C_{2p}^{2} \times 2^{n}$, where

- $u$ is \# of unit tests,
- $p$ is \# of paras,
- if each para has 2 values,
- if each test has \(n\) nodes has \(n\) nodes

$u \times 2p \times 3t$, where

- $t$ is \# of node types,
- if each para has 2 values,
- max \# of rep. assignment is 3
TestGen: Pre-run to Profile Unit Tests

• Run each unit test once and collect test meta info.
  • For each test, record which parameter read by what node with what value

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Node</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>encrypt</td>
<td>NodeA.1</td>
<td>true</td>
</tr>
<tr>
<td>encrypt</td>
<td>NodeA.2</td>
<td>true</td>
</tr>
<tr>
<td>timeout</td>
<td>NodeB.1</td>
<td>1s</td>
</tr>
</tbody>
</table>
TestGen: Pre-run to Profile Unit Tests

- Run each unit test once and collect test meta info.
  - For each test, record which parameter read by what node with what value
- Use this info to filter ineffective tests.
  - Result: filtered 97% of tests in HDFS (387M -> 10M)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Node</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>encrypt</td>
<td>NodeA.1</td>
<td>true</td>
</tr>
<tr>
<td>encrypt</td>
<td>NodeA.2</td>
<td>true</td>
</tr>
<tr>
<td>timeout</td>
<td>NodeB.1</td>
<td>1s</td>
</tr>
</tbody>
</table>

- Parameter: encrypt
  - Test N
  - Test1

- TestGen: Pre-run to Profile Unit Tests

- Parameter: encrypt

<test1, encrypt, NodeA,...>

<test1, encrypt, NodeB,...>

encrypt not used by NodeB
ConfAgent: Challenges

In distributed setting, we can specify the config file when starting a node as process.
ConfAgent: Challenges

In distributed setting, we can specify the config file when starting a node as process.
In distributed setting, we can specify the config file when starting a node as process.
ConfAgent: Challenges

In distributed setting, we can specify the config file when starting a node as process.

This doesn’t work in unit tests, as nodes are often created as threads in a single process (i.e., minicluster testing)
ConfAgent: Challenges

Key idea: attribute config objects to nodes & hack return values

Challenges:
• Each node can have multiple config objects.
• Config objects can be shared among nodes.
• Values in one config object seen by multiple nodes.

It doesn't work in unit tests, as nodes are often created as threads in a single process (i.e., minicluster testing)
ConfAgent: Challenges

Key idea: attribute config objects to nodes & hack return values

Challenges:
• Each node can have multiple config objects.
• Config objects can be shared among nodes.
  • Values in one config object seen by multiple nodes.

A node may read inconsistent values, causing false positives.

This doesn't work in unit tests, as nodes are created as threads in a single process (minicluster testing)
ConfAgent’s Solution

• Clone config objects that can be shared.
ConfAgent’s Solution

• Clone config objects that can be shared.
ConfAgent’s Solution

• Clone config objects that can be shared.

• Track config creation flow and attribute config objects to nodes by rules.
  • E.g., Conf b = new Conf(a) & a belongs to node1 → b belongs to node1
ConfAgent’s Solution

• Clone config objects that can be shared.

• Track config creation flow and attribute config objects to nodes by rules.
  • E.g., Conf b = new Conf(a) & a belongs to node1 → b belongs to node1

• Track uncertain config objects.
  • Avoid testing parameters read from them.
ConfAgent’s Solution

• Clone config objects that can be shared.

• Track config creation flow and attribute config objects to nodes by rules.
  • E.g., Conf b = new Conf(a) & a belongs to node1 → b belongs to node1

• Track uncertain config objects.
  • Avoid testing parameters read from them.

• Manipulate config parameter values.
Evaluation

• Hardware setting
  • We run all the experiments on CloudLab
  • Intel Xeon 10-core CPUs, 192 GB DRAM, 480 GB SATA SSD

• Applications
  • Five app: HDFS, YARN, MR, HBase, Flink
  • Modification overhead: 18 to 38 LOC
  • Totally 4,652 machine hours with up to 100 physical machines, each running 20 Docker containers
• ZebraConf reports 57 heterogeneous-unsafe parameters.
• Our manual analysis finds 41 are true problems.
• Categories of these parameters:
  • Data transfer format related
  • Max limit related
  • Timeout related
  • Task numbers related
  • ...
  • Unexpected ones
dfs.datanode.balance.max.concurrent.moves

- Limits the max number of threads that a DataNode can use for balancing.
**dfs.datanode.balance.max.concurrent.moves**

- Limits the max number of threads that a DataNode can use for balancing.

*Moves*=$value

DataNode 1

Balancer

DataNode N

Balancer
dfs.datanode.balance.max.concurrent.moves

- Limits the max number of threads that a DataNode can use for balancing.

Table:

<table>
<thead>
<tr>
<th>Configs</th>
<th>Balancing Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Balancer: 50, DN: 50</td>
<td>14s</td>
</tr>
<tr>
<td>Balancer: 1, DN: 1</td>
<td>16.7s</td>
</tr>
</tbody>
</table>

Diagrams:
- DataNode 1
- DataNode N
- Balancer
- $moves=value$
- $moves=value$
`dfs.datanode.balance.max.concurrent.moves`

- Limits the max number of threads that a DataNode can use for balancing.

```
Balancer: 50, DN: 50  14s
Balancer: 1, DN: 1    16.7s
Balancer: 50, DN: 1   154s
```

- `moves=1`
  - DataNode 1
  - DataNode N
  - Balancer

10x slower than just using 1 thread
We find the problem has been discussed in the community.

HDFS-7466: “Allow different values for moves per DataNode”

- DataNodes on different storage tiers, i.e., DISK and ARCHIVE
- Different thread numbers on different nodes for better performance
- Proposed solution: balancer fetches moves value from each DataNode.
Evaluation – Test Filtering

- How many ineffective tests can be filtered in pre-run phase?
- How many tests are filtered due to uncertainty?

<table>
<thead>
<tr>
<th># of tests</th>
<th>Flink</th>
<th>HBase</th>
<th>HDFS</th>
<th>MR</th>
<th>YARN</th>
</tr>
</thead>
<tbody>
<tr>
<td>w/o pre-run</td>
<td>7,193M</td>
<td>557M</td>
<td>387M</td>
<td>284M</td>
<td>705M</td>
</tr>
</tbody>
</table>
Evaluation – Test Filtering

- How many ineffective tests can be filtered in pre-run phase?
- How many tests are filtered due to uncertainty?

<table>
<thead>
<tr>
<th># of tests</th>
<th>Flink</th>
<th>HBase</th>
<th>HDFS</th>
<th>MR</th>
<th>YARN</th>
</tr>
</thead>
<tbody>
<tr>
<td>w/o pre-run</td>
<td>7,193M</td>
<td>557M</td>
<td>387M</td>
<td>284M</td>
<td>705M</td>
</tr>
<tr>
<td>after pre-run filter</td>
<td>2,019K</td>
<td>6,145K</td>
<td>10,404K</td>
<td>482K</td>
<td>668K</td>
</tr>
<tr>
<td></td>
<td>-99.9%</td>
<td>-98.9%</td>
<td>-97.3%</td>
<td>-99.8%</td>
<td>-99.9%</td>
</tr>
</tbody>
</table>
Evaluation – Test Filtering

- How many ineffective tests can be filtered in pre-run phase?
- How many tests are filtered due to uncertainty?

<table>
<thead>
<tr>
<th># of tests</th>
<th>Flink</th>
<th>HBase</th>
<th>HDFS</th>
<th>MR</th>
<th>YARN</th>
</tr>
</thead>
<tbody>
<tr>
<td>w/o pre-run</td>
<td>7,193M</td>
<td>557M</td>
<td>387M</td>
<td>284M</td>
<td>705M</td>
</tr>
<tr>
<td>after pre-run filter</td>
<td>2,019K</td>
<td>6,145K</td>
<td>10,404K</td>
<td>482K</td>
<td>668K</td>
</tr>
<tr>
<td></td>
<td>-99.9%</td>
<td>-98.9%</td>
<td>-97.3%</td>
<td>-99.8%</td>
<td>-99.9%</td>
</tr>
<tr>
<td>after uncertain filter</td>
<td>1,972K</td>
<td>6,033K</td>
<td>10,242K</td>
<td>430K</td>
<td>640K</td>
</tr>
<tr>
<td></td>
<td>-2.4%</td>
<td>-1.8%</td>
<td>-1.6%</td>
<td>-10.7%</td>
<td>-4.2%</td>
</tr>
</tbody>
</table>
Conclusion

• ZebraConf reuses existing unit tests to find unsafe parameters.
• We find 41 heterogeneous-unsafe parameters with ZebraConf.
• Need better support for heterogeneous configurations.
• We made ZebraConf publicly available: https://github.com/StarThinking/ZebraConf/