ScaleCheck
A Single-Machine Approach for Discovering Scalability Bugs in Large Distributed Systems

Cesar A. Stuardo, Tanakorn Leesatapornwongsa*, Riza O. Suminto, Huan Ke, Jeffrey F. Lukman, Wei-Chiu Chuang†, Shan Lu and Haryadi S. Gunawi
New Class of Bugs?

- “Classic” critical bugs
  - Concurrency
  - Configuration
  - Performance
  - Security

- Scalability bugs?
Scalability bugs?

- **Cassandra**
  - Bug #3831: "start exploding when start reading 100-300 nodes"
  - Bug #6127: "obvious symptom is with 1000 nodes"
  - Bug #6409: "With >500 nodes, ... have trouble"

- **Hadoop/HDFS**
  - Bug #1073: "1600 data nodes. randomWriter fails to run"
  - Bug #3990: "multi-thousand node cluster often results in [problem]"

- **HBase**
  - Bug #10209: "with 200+ nodes, [cluster] takes 50mins to start"
  - Bug #12139: "with >200 nodes, StochasticLoadBalancer doesn't work"

- **Riak**
  - Bug #926: "1000 node cluster is a larger question"
Latent bugs whose symptoms surface in large-scale deployments, but not necessarily in small/medium-scale deployments.
An Example: Cassandra Bug #3831

Decommissioning

Update Ring

N=6
**An Example: Cassandra Bug #3831**

In every node
\[
\text{for}(i=0 \text{ to } N) \\
\text{for}(j=0 \text{ to } N) \\
\text{for}(k=0 \text{ to } N) \\
\] \(O(N^3)\)

CPU intensive

0.1 – 4 seconds

In all nodes!
**(unstable cluster)**

\[N > 200\]
The “Flapping” Bug(s)

CASS-3831: Decommission

The bug symptom
Real-scale Testing

“For Apache Hadoop, testing at thousand-node scale has been one of the most effective ways of finding bugs, but it’s both difficult and expensive. It takes considerable expertise to deploy and operate a large-scale cluster, much less debug the issues. Running such a cluster also costs thousands of dollars an hour, making scale testing impossible for the solo contributor.

As it stands, we are heavily reliant on test clusters operated by large companies to do scale testing. A way of finding scalability bugs without requiring running a large-scale cluster would be extremely useful.”

— Andrew Wang (Cloudera and Apache Hadoop PMC Member and Committer)

Q1 What are the root causes of scalability bugs?

Q2 Can we do large-scale testing easily, e.g. in one machine?
ScaleCheck

Discovering scalability bugs
Democratizing large scale testing

Real-scale testing
What are the root causes of scalability bugs?

Scale Dependent Loops

Q1

ScaleCheck

SFind

Automatically find scale dependent data structures and loops by checking their in-memory growth tendency over small scale experiments
Can we do large-scale testing easily, e.g. in one machine?

Many Challenges!!!
Memory Overhead
Context switching
CPU contention
Network Overhead
Bad modularity

ScaleCheck

STest

SPC  Single Process Cluster
GEDA Global Event-Driven Architecture
PIL  Processing Illusion

Large scale testing in a single machine with little source code modification
Our Results

- Integrated to 4 systems
- Scale tested 18 protocols
- Accurately reproduced 10 known scalability bugs
- Found 4 new bugs

.....in just a single machine
Outline

- Introduction
- SFind
- STest
- Evaluations and Conclusions
Our Frequent Root Cause...

Scale Dependent Loops

iterating scale dependent data structures

- Any distributed system these days has...
  - > 100K LOC
  - > 100x source files
  - > 1000x maps, lists, queues, etc...

- We can find them manually...
  - /* ... holds one entry per peer ... */
  - /* ... maps partition to node... */
  - /* ... maps blocks to datanodes... */

- How can we find them automatically?
JVMTI and Reflection support

List “peers” growth tendency

Step 1

Step 2

Step N

Size

Step

N

2

1

1

2

1

2

1

N
Data structures that tend to grow when one or more dimensions grow are *scale dependent*

A’s on heap data structures *growth tendency after N steps*
Finding many scale-dep data structures

- Try out different axes of scale
  - Adding files
  - Adding partitions
  - tables, columns, files, directories...

- Try out different workloads
  - Snapshot directories
  - Removing nodes
  - migration, rebalance...

- Data structures:
  - 26 data structures (Cassandra, HDFS)
    - inodes
    - tokens

  - 18 data structures (Cassandra, HDFS)
    - snapshots
    - deadNodes
Prioritizing Scale-Testing

Map our data structures to loops using data-flow
Create **inter-procedural call graphs** for all methods

<table>
<thead>
<tr>
<th>Priority</th>
<th>Method</th>
<th>Complexity</th>
<th>#IO</th>
<th>#Lock</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>sendToEndpoints</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>?</td>
<td>applyStateLocally</td>
<td>2</td>
<td>$N^2$</td>
<td>$N^2$</td>
</tr>
</tbody>
</table>

Prioritize harmful scalability bottlenecks

<table>
<thead>
<tr>
<th>Priority</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>applyStateLocally</td>
</tr>
<tr>
<td>2</td>
<td>sendToEndpoints</td>
</tr>
</tbody>
</table>
Outline

- Introduction
- SFind
- STest
- Evaluations and Conclusions
Democratizing Large Scale Testing with STest

<table>
<thead>
<tr>
<th>Priority</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>applyStateLocally</td>
</tr>
</tbody>
</table>

```
public void testApplyStateLocally(...){
    //...
}
```

Many Challenges!!!
- Memory Overhead
- Context switching
- CPU intensity
- Network Overhead
- Bad modularity
Try #1: Naïve Packing (NP)

- Deploy **nodes as processes** in a single machine
  - One process = one node = one system’s instance

- Main challenges
  - Per node runtime overhead
  - Process context switching
    - **Event Lateness** (e.g. send message)

- Max Colocation factor: **50 nodes**

Prohibits large scale-testing!
Try #2: Single Process Cluster (SPC)

- Deploy **nodes as processes threads** in a single process

- **Challenges**
  - Lack of modularity (e.g. bad design patterns)
    - Automatic per-node *isolation* (e.g. ClassLoader isolation)
  - Large amount of threads
    - Simplify inter-node communication

- Max Colocation factor: **120 nodes**
Per-Node Services

- Frequent Design pattern
  - Service = worker pool + event queue

But N nodes mean $N \times (S_1 + \ldots + S_s)$ threads!!!
Try #3: Global Event Driven Architecture (GEDA)

- One **global event handler** per service
  - total # of threads = # of cores
- Identify **event constrains**
  - Multi-threaded versus single-threaded
- Max Colocation factor: **512 nodes**
Using our Framework for HDFS-9188

**RPC Queue full = no other request can be served**

- **Namenode’s RPC Queue**
  - IBR processed in serial
  - Max size = 1000
  - FULL!

- **Datanodes**
  - N = 32
  - N = 64
  - N = 128
  - N = 256

**Accurate!**

---

HDFS-9198: RPC Queue Size (x1000)

- SCk
- Real

![Graph showing queue size vs. number of nodes](image)
But Meanwhile in CASS-3831...

Inaccurate!
CPU Contention...

Try #4: Processing Illusion (PIL)

- Replace CPU intensive computation with sleep
- “The key to computation is the execution time and eventual output”
- What code blocks can be safely replaced by sleep?
(Easy) CPU intensive blocks

Non pertinent operations that do not affect processing semantics

```javascript
function intense()
for(...){
  for(...){
    for(...){
      write(“hello!”);
    }
  }
}
```

Can be PILed...

```javascript
function intense()
if(!ScaleCheck){
  for(...){
    for(...){
      for(...){
        write(“hello!”);
      }
    }
  }
}
else{
  t = getTime(...)
  sleep(t)
} // O(1)
```

Offline profiling
(Hard) CPU intensive blocks

**Pertinent operations that affect processing semantics**

```javascript
function relevantAndRelevant(input)
  for(...){
    for(...){
      for(...){
        modifyClusterState(...)
      }
    }
  }
}
```

Can be PILEd...

```javascript
function relevantAndRelevant(input)
  if(!ScaleCheck){
    for(...){
      for(...){
        for(...){
          modifyClusterState(...)
        }
      }
    }
  }
  else{
    t, output = getNextState(input)
    sleep(t)
    clusterState = output
  }
```

Offline/Online memoization
How about now?

CASS-3831: Decommission

CASS-5456: Scale-Out

RIAK-3926: Bootstrap
Outline

- Introduction
- SFind
- STest
- Evaluations and Conclusions
Reproducing 8 known bugs accurately

[see paper for details]
Finding New Bugs

Meanwhile in latest Cassandra...

\[ O(N^3) \]

\[ O(N) \text{ IO} \]

\[ O(N) \text{ Lock} \]
Finding New Bugs

HDFS: Snapshot Diff

- Snapshot diff time (s) vs. Snapshot diff size

HDFS: MetaSave

- Metasave time (s) vs. Under-replicated blocks

Real
Colocation Factor

How many nodes can we collocate in a single machine maintaining accuracy?

Need all techniques for a high colocation factor.

- Cassandra decommission
- HDFS IBR
- Voldemort rebalance

- PIL
- GEDA
- NetStub
- SPC
- Naive
Limitations and Future Work

- Focus on scale dependent CPU/Processing time
  - E.g. cluster size, number of files, etc...
  - Scale of load, Scale of failure

- Colocation factor limited by a single machine
  - Akin to unit testing
  - Multiple machines

- PIL might require real scale memoization
  - Offline memoization takes too long (days)
  - Explore other CPU contention reduction techniques

- New code = New maintenance costs
  - Low cost (about 1% of code base)
  - Going for zero-effort emulation
Conclusions

Real-scale testing

ScaleCheck:

Discovering scalability bugs

Democratizing large scale testing
Thank you!
Questions?

http://ucare.cs.uchicago.edu
http://cs.uchicago.edu
https://ceres.uchicago.edu