Extracting Configuration Parameter Interactions using Static Analysis
Chelsea Metcalf, Farhaan Fowze, Tuba Yavuz, and Jose Fortes
University of Florida
Problem

- Complex software systems come with a huge number of configuration parameters
- Understanding their impact on run-time behavior as well as their interaction with other parameters is a challenge
- Example: Apache Hadoop
Problem

- Using default settings causes performance penalties [8]
- Finding interactions between parameters can improve performance of the software
- Also aids product based modification by specifying which configuration parameters need to be tuned
- Only a few interactions have been reported in the literature [1], [2], [4], [5], [6], [11]
Contribution

- Find interacting parameters in scope of a software component using static slicing technique
  - Thin Slicing: missing control flow dependencies
  - Scalability

- Display interactions in a formalism we call “interaction graphs”
Static Analysis

- Starts by building on-the-fly call graph
  - Nodes represent methods
  - Edges represent method calls

- **Our definition:** Two parameters **interact** if they impact the **same program statement** through **data** and/or **control flow** dependency

- Use **WALA** program analysis framework to extract static control-flow and data-flow dependencies

- Use **Thin Slicing** to find configuration parameters that may interact
Example

```java
public void main(String[] args) {
    1: int v1 = cp.get("p1");
    2: int v2 = cp.get("p2");
    3: int v3 = cp.get("p3");
    4: if (v1 > 0 && v2 > 0)
    5:     foo2(v3);
}

void foo2(int a1) {
    6:    int f = a1;
}
```
---
= data flow
———> = control flow
Algorithm: *FindInteractingParameters*

- **Input:** Software component
- **Goal:** Find configuration parameters that may interact
- Visit every node of call graph and every program statement in call graph node
- Find control flow statements that are controlling the target statement up to a given depth (control depth)
Algorithm: FindInteractingParameters

- Backwards Thin Slicing is performed on each statement, the target statement, of a given software component along with the control statements.
- Slicing results consists of all nodes reachable from slicing seed.
- Bounded data-flow dependency chain length (slicing depth).
- After bounded slice is computed, algorithm selects statements that represent configuration parameters and extracts names of impacting parameters.
The data goes through the following phases:

- **Input Splits:** Input to a MapReduce job is divided into input splits
- **Mapping:** Data in each split is passed to a mapping function to produce output values
  - Example: WordCount, prepare a list in the form <word,frequency>
- **Shuffling:** Consolidates relevant records from Mapping phase output
- **Reducing:** Combines values from Shuffling phase and returns a single output value
Case Study

- **HADOOP-2095**: “Reducer failed due to Out of Memory”
- Hadoop was generating too many codec objects at run-time and consuming more memory than what was anticipated
- Culprit program statement in `init()` method of class `SequenceFile$Reader`:
  - `this.codec = (CompressionCodec)ReflectionUtils.newInstance(codecClass, conf);`
- **Three configuration parameters that interact**:
  - `mapred.compress.output`
  - `mapred.inmem.merge.threshold`
  - `io.sort.factor`
Case Study

Input: SequenceFile
Seed Statement: newInstance
Output: io.sort.factor, mapred.inmem.merge.threshold
### Results

<table>
<thead>
<tr>
<th>Configuration Parameter</th>
<th>Control Depth</th>
<th>Slicing Depth</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>io.sort.factor</code></td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td><code>mapred.reduce.parallel.copies</code></td>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td><code>mapred.reduce.copy.backoff</code></td>
<td>1</td>
<td>41</td>
</tr>
<tr>
<td><code>io.file.buffer.size</code></td>
<td>1</td>
<td>43</td>
</tr>
<tr>
<td><code>mapred.userlog.limit.kb</code></td>
<td>1</td>
<td>50</td>
</tr>
<tr>
<td><code>io.seqfile.compress.blocksize</code></td>
<td>2</td>
<td>14</td>
</tr>
<tr>
<td><code>fs.local.block.size</code></td>
<td>2</td>
<td>26</td>
</tr>
<tr>
<td><code>mapred.inmem.merge.threshold</code></td>
<td>2</td>
<td>30</td>
</tr>
<tr>
<td><code>io.sort.mb</code></td>
<td>3</td>
<td>11</td>
</tr>
</tbody>
</table>

The set of configuration parameters that impact the codec creation in the Merge phase of the Reduce stage in Hadoop 0.15.0, which manifested the out of memory error as reported in HADOOP-2095.
## Call Graph Statistics

<table>
<thead>
<tr>
<th>Version</th>
<th>Component</th>
<th># of Nodes</th>
<th># of Edges</th>
<th>Bytecode Size (KB)</th>
<th>Time (hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.6.0</td>
<td>MapTask</td>
<td>17749</td>
<td>67648</td>
<td>577</td>
<td>3.8</td>
</tr>
<tr>
<td></td>
<td>Shuffle</td>
<td>18433</td>
<td>71277</td>
<td>537</td>
<td>3.3</td>
</tr>
<tr>
<td></td>
<td>Merge</td>
<td>18264</td>
<td>69725</td>
<td>533</td>
<td>5.1</td>
</tr>
<tr>
<td>0.15.0</td>
<td>MapTask</td>
<td>11789</td>
<td>32164</td>
<td>399</td>
<td>0.4</td>
</tr>
<tr>
<td></td>
<td>ReduceTask</td>
<td>13617</td>
<td>42393</td>
<td>410</td>
<td>1.6</td>
</tr>
</tbody>
</table>
Presenting the Data
Quantifying Interactions

- For a software component \( \text{comp} \)
- Parameters \( p_1 \) & \( p_2 \) that impact \( \text{comp} \)
- \( \text{Impact}_{\text{comp}}(p_1) = \# \text{ of program statements in } \text{comp} \text{ that } p_1 \text{ impacts} \)
- \( \text{Impact}_{\text{comp}}(p_1, p_2) = \# \text{ of common program statements in } \text{comp} \text{ that } p_1 \text{ and } p_2 \text{ impact} \)
- \( \text{Inter}_{\text{comp}}(p_1, p_2) = \text{Interaction of parameter } p_1 \text{ with } p_2 \text{ in the context of } \text{comp} \)
$T = \text{Inter}_{\text{comp}}(p_1, p_2) = \frac{\text{Impact}_{\text{comp}}(p_1, p_2)}{\text{Impact}_{\text{comp}}(p_1)}$

$p_1 \xrightarrow{T \geq \theta} p_2$

$p_1 \rightarrow p_2 \not\equiv p_2 \rightarrow p_1$
Hadoop Version: 2.6.0
Component: Shuffle phase of ReduceTask
Threshold: 70%
## Interaction Graph Statistics

<table>
<thead>
<tr>
<th>Version</th>
<th>Component</th>
<th># of Nodes</th>
<th># of Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.6.0</td>
<td>MapTask</td>
<td>30</td>
<td>199</td>
</tr>
<tr>
<td></td>
<td>Shuffle</td>
<td>40</td>
<td>146</td>
</tr>
<tr>
<td></td>
<td>Merge</td>
<td>25</td>
<td>263</td>
</tr>
<tr>
<td>0.15.0</td>
<td>MapTask</td>
<td>8</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>ReduceTask</td>
<td>9</td>
<td>37</td>
</tr>
</tbody>
</table>
## Reported Interactions

<table>
<thead>
<tr>
<th>Source</th>
<th>Configuration Parameter Interactions Reported in the Literature</th>
<th>Interaction Graph Results</th>
</tr>
</thead>
</table>
| Principal Component Analysis [22]    | *io.sort.factor, mapred.compress.map.output*  
 *io.sort.mb, mapred.child.java.opts, reduce.input.buffer.percent*  
 *mapred.reduce.tasks, mapred.map.tasks, mapred.reduce.parallel.copies* | Component | Average Interaction (InterComp) |
|                                       |                                                                                                                               | MapTask | 95%                           |
|                                       | *io.sort.mb, mapred.reduce.tasks, io.sort.spill.percent, io.sort.record.percent*                                                | Merge    | NA                            |
|                                       | *io.sort.factor, min.num.spills.for.combine*                                                                                | MapTask | 97%                           |
 *mapred.job.reduce.input.buffer.percent, mapred.child.java.opts, io.sort.factor*                                      | Shuffle | 95%                           |
|                                       | *mapreduce.tasktracker.http.threads, mapred.reduce.parallel.copies, mapred.job.shuffle.input.buffer.percent*                  | MapTask | 86%                           |
|                                       | *mapred.job.shuffle.merge.percent, mapred.child.java.opts, mapred.inmem.merge.shuffle.threshold, mapred.job.shuffle.merge.percent, io.sort.factor* | Shuffle | 91%                           |
|                                       | *mapred.job.reduce.input.buffer.percent*                                                                                     | Merge    | 93.9%                         |
Filtered Interaction Graph
Related Work

- Precomputing possible configuration error diagnosis
  - Ariel Rabkin and Randy Katz

- Automated diagnosis of software configuration errors
  - Sai Zhang and Michael Ernst

- iTREE: efficiently discovering high-coverage configurations using interaction trees
  - Charles Song, Adam Porter, and Jeffrey Foster
Conclusion & Future Work

- Finding configuration parameter interaction
- Our algorithm can find the most relevant parameter, although it might be missing some of the relevant ones

Future Work:
- Integrate static analysis with dynamic analysis to extract interactions that cross component boundaries
- Provide more parameters for visualization
Thank You & Questions

Chelsea Metcalf: chelseametcalf@ufl.edu
Farhaan Fowze: farhaan104@ufl.edu
Tuba Yavuz: tuba@ece.ufl.edu
Jose Fortes: fortes@acis.ufl.edu
References


