Non-intrusive Performance Profiling for Entire Software Stacks based on The Flow Reconstruction Principle

Xu Zhao, Kirk Rodrigues, Yu Luo, Ding Yuan, Michael Stumm
Motivations

- Debugging distributed system stacks is difficult

- Existing tools are limited
  - **Intrusive**: manually built-in domain knowledge
  - **Machine learning** on logs: unable to reconstruct execution flow
  - **Static analysis [Zhao OSDI’14]**: cannot cross different software components

- Use programmers’ intuition in postmortem execution analysis
The Flow Reconstruction Principle

Programmers log sufficient information so that they can reconstruct the execution flow in distributed stacks

- **Three rules**
  - Log **events** at critical points in the control path
  - Log **object identifiers** to differentiate concurrent runs
  - Log **sufficient identifiers** to unambiguously identify an object

Thread1

11/02/16 18:00 Thread starts

Thread2

11/02/16 18:01 Thread starts
The Flow Reconstruction Principle

Programmers log sufficient information so that they can \textit{reconstruct the execution flow in distributed stacks}

- Three rules
  - Log \textit{events} at critical points in the control path
  - Log \textit{object identifiers} to differentiate concurrent runs
  - Log \textit{sufficient identifiers} to unambiguously identify an object

\begin{itemize}
  \item \texttt{Thread1}\newline
    11/02/16 18:00 \texttt{Thread 1} starts

  \item \texttt{Thread2}\newline
    11/02/16 18:01 \texttt{Thread 2} starts
\end{itemize}
The Flow Reconstruction Principle

Programmers log sufficient information so that they can reconstruct the execution flow in distributed stacks.

- Three rules
  - Log **events** at critical points in the control path
  - Log **object identifiers** to differentiate concurrent runs
  - Log **sufficient identifiers** to unambiguously identify an object

Thread 1:

11/02/16 18:00 [Process 2355] **Thread 1** starts

Thread 2:

11/02/16 18:01 [Process 1224] **Thread 2** starts
The Flow Reconstruction Principle

Programmers log sufficient information so that they can reconstruct the execution flow in distributed stacks

- **Three rules**
  - Log **events** at critical points in the control path
  - Log **object identifiers** to differentiate concurrent runs
  - Log **sufficient identifiers** to unambiguously identify an object

Thread1

11/02/16 18:01 **Process 2355** starts a Thread

11/02/16 18:01 **Process 1224** starts a Thread

11/02/16 18:01 **Thread 2** starts

11/02/16 18:02 **Thread 1** starts
Stitch: A Non-intrusive Profiler for Entire Server Stacks

- Works without any built-in domain knowledge
- Purely based on IDs, throws away constant text
- Reconstructs objects, infers their lifetimes and hierarchies

Node 1
2016-05-02 23:23:01,374 INFO

Node 2
2016-05-02 23:23:01,374 INFO

Visualization

User_Alice

Query 1
Query 2
Outline

- Real world example
- Design of Stitch
- User study and correctness evaluation
Real World Example

- User study on 14 programmers

- We reproduced a real world performance anomaly: YARN-4610
  - Symptom: user Alice reports her Hive query takes longer than expected

- Separate programmers into 2 groups: debug using raw log or with Stitch
  - 45 mins time limit
Debugging with Raw Log

- Only 2 users using raw log can diagnose within the time limit
- Reading each log message is infeasible: over 5,000 lines of logs
- Users take two approaches: bottom-up and top-down
  - **Bottom-up** results in a wild goose chase

```
$ grep “ERROR\|WARN” -R logs/
./node0/userlogs/application_1462245782384_0024/container_1462245782384_0024_01_000576/syslog:2016-05-02 23:46:37,010 ERROR [Thread-50]
org.apache.hadoop.mapreduce.v2.app.rm.RMContainerAllocator: Could not
deallocate container for task attemptId attempt_1462245782384_0024_r_000001_1
... [707 more]
```
Debugging with Raw Log

- Only 2 users using raw log can diagnose within the time limit
- Reading each log message is infeasible: over 5,000 lines of logs
- Users take two approaches: **bottom-up** and **top-down**
  - **Bottom-up** results in a wild goose chase
  - **Top-down** gets closer, but still cannot diagnose failure within time limit
    - Determine slowest map task
Debugging with Stitch

► All 7 Stitch users succeeded, spending 13 mins on average

► Stitch speeds-up debugging time by a factor of 3.5

► Demo
Outline

- Real world example
- Design of Stitch
- User study and correctness evaluation
Client: Log Parsing

- Detect log printing processes and log files
  - Periodically read /proc
- Log parsing: separate dynamic and constant parts of the raw log
  - Extract **string constants** from binaries, match against the log
  - Filter non-identifiers using a blacklist ("memsize", "ms", "progress", etc.)

```
16/04/02 00:58 MongoDB starting: pid=22925 port=27017 dbpath=/var/lib/mongodb

// Code that prints this msg:
l << "MongoDB starting : pid=" << pid << " port=" << serverGlobalParams.port << " dbpath=" << storageGlobalParams.dbpath;
```

<table>
<thead>
<tr>
<th>Parsed Log</th>
<th>Timestamp</th>
<th>Identifier</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>16/04/02 00:58</td>
<td>22925</td>
<td>pid</td>
<td></td>
</tr>
<tr>
<td></td>
<td>27017</td>
<td>port</td>
<td></td>
</tr>
<tr>
<td></td>
<td>/var/lib/mongodb</td>
<td>filePath</td>
<td></td>
</tr>
</tbody>
</table>
Server: Identifier Relations

- **1:1 relation**
  - Can be used interchangeably

  ![Diagram showing a 1:1 relation between Host1 and IP 192.168.0.100]

- **1:n relation**
  - Hierarchical relationship

  ![Diagram showing a 1:n relationship between User Bob and Queries 20161102_0001 and 20161102_0002]

- **m:n relation**
  - Combination is required to unambiguously identify an object

  ![Diagram showing a m:n relationship between Process 42, Process 43, Thread 1, and Thread 2]
m:n Relation

- **Example:** Host, Process and Thread

- Not every combination is meaningful
  - Meaningful ones: \{Host\}, \{Host, Process\}, \{Host, Process, Thread\}
  - Meaningless combination: \{Host, Thread\}

- Developers only log the meaningful combinations
  - Developers will never log \{Host, Thread\} without Process
Lifetime and Hierarchy

- Lifetime of objects inferred from event timestamps
- Hierarchy is inferred from 1:n relation

1. 16/04/02 00:58 User Bob creates Hive query: Query 20161102_0001
2. 16/04/02 01:03 User Bob creates Hive query: Query 20161102_0002
3. 16/04/02 01:06 User Bob Hive query Query 20161102_0001 finishes
4. 16/04/02 01:10 User Bob Hive query Query 20161102_0002 finishes
Outline

► Real world example
► Design of Stitch
► **User study and correctness evaluation**
Evaluation

► How much time does Stitch save in debugging?

► How accurate is Stitch in identifying objects?

► Do real-world systems follow the Flow Reconstruction Principle?
User Study Evaluation

- 14 users work with 3 cases
  - *Performance debugging* – An anomaly caused by a YARN scheduler bug
  - *Bottleneck identification* – Identify slow node in a MapReduce job
  - *Hierarchy identification* – OpenStack components involved in request

- At least **4.6x** faster debugging with Stitch
Correctness Evaluation

- Hive, Spark and OpenStack workload: 200 nodes run for 24 hours
- Production workload: 24-node cluster running for four months
- Together there are 19 software components

<table>
<thead>
<tr>
<th>System</th>
<th>Objects</th>
<th>Object Types</th>
<th>Object Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hive</td>
<td>295,042</td>
<td>32</td>
<td>90%</td>
</tr>
<tr>
<td>Spark</td>
<td>192,969</td>
<td>31</td>
<td>94%</td>
</tr>
<tr>
<td>OpenStack</td>
<td>214,882</td>
<td>14</td>
<td>100%</td>
</tr>
<tr>
<td>Production</td>
<td>8,141</td>
<td>24</td>
<td>100%</td>
</tr>
<tr>
<td>Total</td>
<td>711,034</td>
<td>101</td>
<td>96%</td>
</tr>
</tbody>
</table>

- Stitch correctly identifies 96% of all objects in four software stacks
Limitations

► Stitch does not capture causal relations

► Stitch’s efficacy relies on developers following the Flow Reconstruction Principle
  ► Violations of the principle will mislead human debugging

► Stitch’s accuracy is sensitive to the quality of identifiers and type-extraction
  ► Log parsing is a solved problem
  ► Industrial solutions exist, e.g., Splunk, VMWare LogInsight, LogStash
Related Work

- **Intrusive tools** [Aguilera SOSP’03] [Barham OSDI’04] [Chanda EuroSys’07]
  - Captures causal relationships
  - Built-in domain knowledge

- **Static analysis tools** [Zhao OSDI’14] [Yuan ASPLOS’10]
  - Captures causal relationships
  - Cannot cross layers

- **Machine learning solutions** [Xu SOSP’09] [Nagaraj NSDI’12] [Yu ASPLOS’16]
  - Cannot reconstruct execution flow
Conclusions

**The Flow Reconstruction Principle**

- Log **events** at critical points in the control path
- Log **object identifiers** to differentiate concurrent runs
- Log **sufficient identifiers** to unambiguously identify an object

**Stitch: non-intrusive profiler for distributed stacks**
Q & A
Source of Inaccuracy

- Hive ID: Stage- : violating Flow Reconstruction Principle
  - Stage- and Query ID in separate log messages

- Spark ID: inode and HostName - small scale of workload
  - inode and HostName are 1:1 instead of m:n
  - Appear together in an error log message