lprof : A Non-intrusive Request Flow Profiler for Distributed Systems

Xu Zhao, Yongle Zhang, David Lion, Muhammad FaizanUllah, Yu Luo, Ding Yuan, Michael Stumm

Oct. 8th 2014
Performance analysis tools are needed

• Poor performance of distributed systems leads to
  o Increase of user latency
  o Increase of data center cost

• Distributed system behavior is hard to understand
  o Concurrent requests being processed by multiple nodes

• To diagnose poor performance, tools are needed to
  o Reconstruct the request control flow
  o Understand system behavior
Existing tools are intrusive

• Instrument systems to infer request control flow
  o *E.g.* MagPie, Project 5, X-Trace, Dapper, etc.
  o Incur performance overhead
  o Instrumentations are often system specific
System logs contain rich information

• Rich information in logs is not coincidence
  o Developers rely on logs to perform manual debugging

• Distributed systems generate lots of logs
  o During normal execution

7TB/month [Cloudera’13]
25TB/day [Rothschild’09]
Existing log analyzers are limited

- Cannot infer request control flow
  - Machine learning based log analyzers
    - E.g. [Xu’09], DISTALYZER, Synoptic, etc.
    - Only detect system anomalies
  - Commercial tools
    - E.g. splunk, VMWare LogInsight
    - Require users to perform key-word based searches
Infers request control flow from system logs
  - Along with timing information
  - Group logs printed by the same request on multiple nodes
  - Use information generated by static analysis

System logs

Request control flow

Node 1

Node 2

2014-04-19 08:37:00,322 INFO org.apache.de: Receiving block BP-1811987486-1094451935804_8660 src: /172.31.42.16

Node 1

Node 2

Thread1

Thread2

Thread3

Thread4

lprof: a non-intrusive profiler

Node 1

Node 2
Outline

• Introduction
• Case Study
• Design
• Evaluation
A real-world example

- Performance regression – HDFS-4049

![Latency for each type of request]

- `writeBlock` is suspicious

```
Time          Latency
04:00          0 ms
08:00          500 ms
12:00          1000 ms
16:00          1500 ms
```
Intra-node latency doesn’t increase while inter-node does.

Conclusion: unnecessary network communication.
Outline

• Introduction
• Case Study
• Design
• Evaluation
Overview

[Byte code] → [Static analysis] → [Model] → [Log analysis] → [Request database]

[Logs] → [Log analysis] → [Model] → [Static analysis] → [lprof]

Visualization
Challenges

- Goal: to stitch log messages with respective requests

- Logs are interleaved
  - From different request types
  - From different request instances of the same type
- Perfect identifiers doesn’t always exist
- Distributed across multiple nodes

log snippet from HDFS data nodes
Code snippet in HDFS

```java
1 dataXCeiver() {
2     switch (opCode) {
3         case WRITE_BLOCK:
4             blk_id = getBlock();
5             writeBlock(blk_id, ...);
6             break;
7         case READ_BLOCK:
8             blk_id = getBlock();
9             readBlock(blk_id, ...);
10            break;
11     }
12 }
13 writeBlock(blk_id, ...) {
14     log("Receiving block " + blk_id);
15     new PacketResponder().start();
16 }
17 readBlock(blk_id, ...) {
18     log("HDFS_READ block " + blk_id);
19 }
20 PacketResponder.run() {
21     log("Received block " + blk_id);
22     ...
23     log(blk_id + " terminating");
24 }
```

- Top level method - starting method to process a request
- Request identifier - logged variable not modified in one request
- Log temporal order - possible order between log statements
- Communication behavior - communication between threads
Request analysis

- Find top level method
- Find request identifiers
- Intuition
  - Request identifiers already exist for manual debugging
  - Not modified within one request
  - Once modified, outside of the request
Request analysis example

• Bottom-up analysis on call graph
  o Logged variables – identifier candidates (IC)
  o Number of times they got printed – count

```java
1  dataXceiver () {
2      switch(opCode){
3          case WRITE_BLOCK:
4              blk_id = getBlock();
5              writeBlock(blk_id, …);
6              break;
7          case READ_BLOCK:
8              blk_id = getBlock();
9              readBlock(blk_id, …);
10             break;
11      }
12  }
```

### Call Graph

- **dataXceiver()**
  - `IC: {}` count: 0
- **writeBlock()**
  - `IC: {blk_id}` count: 8
- **readBlock()**
  - `IC: {blk_id}` count: 7

### Top Level Method Identifiers

<table>
<thead>
<tr>
<th>Top Level Method</th>
<th>Identifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>writeBlock</td>
<td>blk_id</td>
</tr>
<tr>
<td>readBlock</td>
<td>blk_id</td>
</tr>
</tbody>
</table>

- Once count decreases, pick top level method and identifier
Temporal order analysis

- Control flow analysis in each top level method

```java
20 PacketResponder.run() { 
21     log("Received block " + blk_id);
22     ...
23     log(blk_id + " terminating");
24 }
```

Temporal order
Communication pair analysis

• Communication between request top level methods
  o Intra-node: thread creation, shared objects
  o Network: socket, RPC
    • Pair serializing and de-serializing methods
Summary of static analysis output

- **Top level method & request identifier**
  
<table>
<thead>
<tr>
<th>top level method</th>
<th>identifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>writeBlock()</td>
<td>blk_id</td>
</tr>
<tr>
<td>readBlock()</td>
<td>blk_id</td>
</tr>
</tbody>
</table>

- **Log temporal order**
  
  PacketResponder.run()

- **Communication pair**
  
  writeBlock()
  
  type: Network
  id: blk_id

  writeBlock()

  type: Thread Creation
  id: blk_id

  writeBlock()

  PacketResponder.run()
Distributed log stitching

**Implemented as a MapReduce job**

Node 3

Node 2

Node 1

1 Receiving blk_01
2 Received blk_01
3 blk_01 terminating *Node 1’s log*

*writeBlock*[log1],blk_01
*PktRsp.run*[log2,3],blk_01

`writeBlock`

[9 logs from 3 nodes], *blk_01*

*Map*  *Combine*  *Reduce*
Outline

• Introduction
• Case Study
• Design
• Evaluation
Evaluation methodology

• Evaluated on logs from 4 distributed systems
  o HDFS, Yarn, HBase, Cassandra
  o Logs generated on 200 Amazon EC2 nodes
  o HiBench, YCSB workload

• Authors manually verified each unique log sequence

  - Receiving block ...
  - Received block ...
  - ... terminating

writeBlock log sequence
**Request attribution accuracy**

accuracy for all the log messages

<table>
<thead>
<tr>
<th>System</th>
<th>Correct</th>
<th>Incomplete</th>
<th>Failed</th>
<th>Incorrect</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDFS</td>
<td>97.0%</td>
<td>0.1%</td>
<td>2.6%</td>
<td>0.3%</td>
</tr>
<tr>
<td>Yarn</td>
<td>79.6%</td>
<td>19.2%</td>
<td>1.2%</td>
<td>0.0%</td>
</tr>
<tr>
<td>Cassandra</td>
<td>95.3%</td>
<td>0.1%</td>
<td>4.6%</td>
<td>0.0%</td>
</tr>
<tr>
<td>HBase</td>
<td>90.6%</td>
<td>2.5%</td>
<td>3.4%</td>
<td>3.5%</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>90.4%</strong></td>
<td><strong>5.7%</strong></td>
<td><strong>3.0%</strong></td>
<td><strong>1.0%</strong></td>
</tr>
</tbody>
</table>
Real-world performance anomalies

• Randomly selected 23 anomalies
  o Reproduced each one to collect logs

• lprof is helpful for identifying the root cause for 65%

• Reasons for the cases lprof cannot help
  o Abnormal requests don’t print any logs
  o The abnormal request only print 1 log
    • But latency is needed for debugging
Related work

• Intrusive tools
  o *E.g. MagPie, Project 5, X-Trace, Dapper, etc.*

• Existing log analyzers
  o *E.g. [Xu’09], DISTALYZER, Synoptic, etc.*

• The Mystery Machine [Chow’14]
  o Infers request flow across software layers
  o Analyzes critical path and slack
  o But requires instrumenting IDs into logs
Conclusions

• lprof: a profiler for distributed system
  o Infers request control flow along with timing information
  o Non-intrusive because entirely from system logs
  o Analyzes logs with information generated by static analysis

• lprof leverages the natural way developers do logging
Limitations

- Iprof benefits from good logging practice
  - Iprof cannot help when there’s no log
  - Timestamp is required for latency analysis
  - Good identifier can improve the accuracy

- Iprof cannot infer request across software layers

- Iprof currently works on Java byte code