Rake: Semantics Assisted Network-based Tracing Framework

Yan Chen

Lab for Internet and Security Technology (LIST)
Northwestern Univ.

Joint work with Yao Zhao, Yinzhi Cao, Anup Goyal (NU), and Ming Zhang (MSR)
Motivation

• Large distributed systems involve thousands of or even 100s of thousands of nodes
  – E.g. search system, CDN
• Host-based monitoring cannot infer the performance or detect bugs
  – Hard to translate OS-level info (such as CPU load) into application performance
  – App log may not be enough
  – Hard to collect all the logs
• Task-based approach adopted in many diagnosis systems
  – WAP5, Magpie, Sherlock
Task-based Approaches

• The Critical Problem – Message Linking
  – Link the messages in a task together into a path or tree
Task-based Approaches

- The Critical Problem – Message Linking
  - Link the messages in a task together into a path or tree

- Challenges
  - Accuracy
  - Non-invasiveness
  - Scalability to large computing platforms
  - Applicability to a large number of applications
Existing Approaches

- Black-box approaches
  - Do not need to instrument the application or to understand its internal structure or semantics
  - Time correlation to link messages
    - Project 5, WAP5, Sherlock
  - Rely on time Correlation
  - Accuracy affected by cross traffic
Existing Approaches

• White-box approaches
  – Extracts application-level data and requires instrumenting the application and possibly understanding the application's source codes
  – Insert a unique ID into messages in a task
    • X-Trace, Pinpoint
  – Invasive due to source code modification
## Related Work

<table>
<thead>
<tr>
<th>Invasiveness</th>
<th>Non-Invasive</th>
<th>Invasive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Knowledge</td>
<td>Network Sniffing</td>
<td>Interposition</td>
</tr>
<tr>
<td>Black-box</td>
<td>Project 5, Sherlock</td>
<td>WAP5</td>
</tr>
<tr>
<td>Grey-box</td>
<td>Rake</td>
<td>Magpie</td>
</tr>
<tr>
<td>White-box</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Rake

- Key Observations
  - Generally no unique ID linking the messages associated with the same request
  - Exist polymorphic IDs in different stages of the request

- Semantic Assisted
  - Use the semantics of the system to identify polymorphic IDs and link messages
Message Linking Example
Questions on Semantics

• What Are the Necessary Semantics?
  – Use data flow analysis to automatically extract the invariants (and its transformation)

• How Does Rake Use the Semantics?
  – Naïve design is to implement Rake for each application with specific application semantics

• How Efficient Is the Rake with Semantics?
  – Can message linking to accurate?
  – What’s the computational complexity of Rake?
Potential Applications

- Search
  - Verified by Microsoft collaborator
- CDN
  - CoralCDN is studied and evaluated
- Chat System
  - IRC is tested
- Distributed File System
  - Hadoop DFS is tested
Conclusions

• Feasibility/Applicability
  – Rake works for many popular applications in different categories

• Easiness
  – Rake allows user to write semantics via XML
  – Necessary semantics are easy to obtained given our experience

• Accuracy
  – Much more accurate than black-box approaches and probably matches white-box approaches

• Non-invasiveness
Necessary Semantics

- Intra-node linking
  - The system semantics
- Inter-node link
  - The protocol semantics
Utilize Semantics in Rake

• Implement Different Rakes for Different Application is time consuming
  – Lesson learnt for implementing two versions of Rake for CoralCDN and IRC

• Design Rake to take general semantics
  – A unified infrastructure
  – Provide simple language for user to supply semantics
Example of Rake Language (IRC)

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<Rake>
    <Message name="IRC PRIVMSG">
        <Signature>
            <Protocol> TCP </Protocol>
            <Port> 6667 </Port>
        </Signature>
        <Link_ID>
            <Type> Regular expression </Type>
            <Pattern> PRIVMSG\s+(.*) </Pattern>
        </Link_ID>
        <Follow_ID id="0">
            <Type> Same as Link ID </Type>
        </Follow_ID>
        <Query_ID>
            <Type> No Return ID </Type>
        </Query_ID>
    </Message>
</Rake>
```
Signature

- Signature to Classify Messages
  - <Signature>
    - <Protocol> TCP </Protocol>
    - <Port> 6667 </Port>
  - </Signature>

- Formats of Signatures
  - Socket information
    - Protocol, port
  - Expression for TCP/IP header
    - udp [10]&128==0
  - Regular expression
  - User defined function
Link_ID and Follow_ID

• Follow_IDs
  – The IDs will be in the triggered messages by this message
  – One message may have multiple Follow_IDs for triggering multiple messages

• Link_ID
  – The ID of the current message
  – Match with Follow_ID previously seen

• Linking of Link_ID and Follow_ID
  – Mainly for intra-node message linking
Query_ID and Response_ID

- **Query_IDs**
  - The communication is in Query/Response style, e.g. RPC call and DNS query/response.
  - The IDs will be in the response messages to this message

- **Response_ID**
  - The ID of the current message to match Query_ID previously seen
  - By default requires the query and response to use the same socket

- **Linking of Query_ID and Response_ID**
  - Mainly for inter-node message linking
Complicated Semantics

- The process of generating IDs may be complicated
  - XML or regular expression is not good at complex computations
  - So let user provide own functions
    - User provide share/dynamic libraries
    - Specify the functions for IDs in XML
    - Implementation using *Libtool* to load user defined function in runtime
Example for DNS

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<Rake>
    <Message name="DNS Query">
        <Signature>
            <Protocol> UDP </Protocol>
            <Port> 53 </Port>
            <Expression> udp[10] & 128 == 0 </Expression>
        </Signature>
        <Link_ID>
            <Type> User Function </Type>
            <Library> dns.so </Library>
            <Function> Link_ID </Function>
        </Link_ID>
        <Follow_ID id="0">
            <Type> Link_ID </Type>
        </Follow_ID>
        <Query_ID>
            <Type> Link_ID </Type>
        </Query_ID>
    </Message>
```

Extract the queried host
Accuracy Analysis

• One-to-one ID Transforming
  – Examples
    • In search, URL -> Keywords -> Canonical format
    • In CoralCDN, URL -> Sha1 hash value
  – Ideally no error if requests are distinct

• Request ambiguousness
  – Search keywords
    • Microsoft search data
    • Less than 1% messages with duplication in 1s
  – Web URL
    • Two real http traces
    • Less than 1% messages with duplication in 1s
  – Chat messages
    • No duplication with timestamps
Evaluation

• Application
  – CoralCDN
  – Hadoop

• Experiment
  – Employ PlanetLab hosts as web clients
  – Retrieve URLs from real traces with different frequency

• Metrics
  – Linking accuracy (false positive, false negative)
  – Diagnosis ability

• Compared Approach
  – WAP5
CoralCDN Task Tree
Message Linking Accuracy

- Use Log-Based Approach to Evaluate WAP5 and Rake Linking in CoralCDN
Diagnosis Ability

- Controlled Experiments
  - Inject junk CPU-intensive processes
  - Calculated the packet processing time using WAP5 and Rake

Obviously Rake can identify the slow machine, while WAP5 fails.
Semantics of Hadoop
Get operation

1 - getFileInfo (Filename)
2 - FileInfo (IPC ID)
3 - getBlockLocations (Filename)
4 - BlockLocation (IPC ID)
5 - Copy (Block ID)
6 - Data (TCP Flag)
Semantics of Hadoop
Grep operation

1 - getNewJobID

2 - JobID (Job ID)

3 - Create Files (Filename)

4 - IPC Returns (IPC ID)

5 - Upload blocks (inode ID)

6 - Finish uploading (Socket)

7 - submitJob (Job ID)

8 - Assign Tasks (Task ID)

9 - Finish Task (Task ID)

10 - Job Finish (Job ID)

Client

Job Scheduler

Task Scheduler

Name Node

Data Node
Abused IPC Call in Hadoop

It is a problem that we found in Hadoop source code.

Four “getFileInfo”s are used here, while only one is enough.
Running time of Hadoop steps

- GetNewJobID: 0.994s
- UploadJobFile: 0.23s
- SubmitJob
  - StartMapTask: 0.65s → CompleteTask: 55.4s
  - StartMapTask: 0.71s → CompleteTask: 60.6s
  - StartReduceTask: 0.85s
- heartbeat: 20.6s
- heartbeatResponse
- heartbeat: 22.1s
- heartbeatResponse
- CommitTask
- KillJob: 101.5s
Discussion

• Implementation Experience
  – How hard for user to provide semantics
    • CoralCDN – 1 week source code study
    • DNS – a couple of hours
    • Hadoop DFS – 1 week source code study

• Inter-process Communication

• Encryption
  – Dynamic library interposition
Q & A?

Thanks!
Backup