How To Keep Your Head Above Water While Detecting Errors

Ignacio Laguna, Fahad A. Arshad, David M. Grothe, Saurabh Bagchi

Dependable Computing System Lab
School of Electrical and Computer Engineering
Purdue University
Impact of Failures in Internet Services

• Internet services are expected to be running 24/7
  – System downtime can cost $1 million / hour
    (Source: Meta Group, 2002)

• Service degradation is the most frequent problem
  – Service is slower than usual; almost unavailable
  – Can be difficult to detect and diagnose

• Internet-based applications are very large and dynamic
  – Complexity increases as new components are added
Complexity of Internet-based Applications

- Each tier has multiple components
- Components can be stateful
The Monitor Detection System (TDSC ’06)

- **Non-intrusive** — observe messages between components
- **Online** — faster detection than offline approaches
- **Black-box detection** — components treated as *black boxes*
  - No knowledge of components’ internals
Stateful Rule-based Detection

**Monitor**

- Packet Capturer
- Finite State Machine (event-based state transition model)
- Normal-Behavior Rules

**Detection Process**

1. A message is captured
2. Deduce current application state
3. Match rules based on the deduced state
4. If a rule does not satisfy, signal *alarm*
The *Breaking Point* in High Rate of Messages

- After breaking point, latency increases sharply
- True alarms rate decreases because packets are dropped
- Breaking point expected in any stateful detection system
Avoiding the Breaking Point: 
*Random Sampling* (SRDS ‘07)

• *Processing Load* in Monitor
  \[ \delta = R \times C \]
  
  \(R\): Incoming message rate, \(C\): Processing cost per message

• *Processing Load* \(\delta\) is reduced by reducing \(R\)
  – Only a portion of incoming messages is processed
  – \(n\) out of \(m\) messages are *randomly sampled*

• Sampling is activated if \(R \geq \text{breaking point}\)
The **Non-Determinism Caused by Sampling**

A portion of a Finite State Machine

![Finite State Machine Diagram](Image)

Events in Monitor

1. \( SV = \{ S0 \} \)
2. A message is dropped
3. \( SV = \{ S1, S2 \} \)
4. A message is sampled
5. The message is \( m5 \)
6. \( SV = \{ S5 \} \)

**Definitions:**

**State Vector \((SV)\)** — The state(s) of the application from Monitor’s point of view (deduced state(s))

**Non-Determinism** — Monitor is no longer aware of the exact state the application is in (because of dropped messages)
Remaining Agenda

I. Addressing the problem of non-determinism
   A. Intelligent Sampling
   B. Hidden Markov Model (HMM) for state determination

II. Experimental Test-bed

III. Performance Results

IV. Efficient Rule Matching and Triggering
Challenges with Non-Determinism

- **Decrease Detection Latency**
  (Large $SV$ increases rule matching time)

- **Increase True Alarms**
  (Reduce effect of incorrect messages)

- **Decrease False Alarms**
  (Reduce incorrect states in $SV$)

**Non-Determinism**

**Intelligent Sampling**

**Hidden Markov Model**

$SV$ reduction
Intelligent Sampling

Random Sampling

Finite State Machine

State Vector

Sampled Messages

Incoming Messages

Intelligent Sampling

Sampled Messages

Incoming Messages

Message with *desirable* property

Slide 11/27
What is a Desirable Property of a Message?

Suppose, $SV = \{ S1, S2, S3 \}$, Sampling Rate $= 1/3$

**A portion of a Finite State Machine**

<table>
<thead>
<tr>
<th>Sampled Message</th>
<th>SV</th>
<th>Discriminative Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m1$</td>
<td>${ S4, S5, S6 }$</td>
<td>3</td>
</tr>
<tr>
<td>$m2$</td>
<td>${ S7, S8 }$</td>
<td>2</td>
</tr>
<tr>
<td>$m3$</td>
<td>${ S9 }$</td>
<td>1</td>
</tr>
</tbody>
</table>

*Discriminative Size* — Number of times a message appears in transitions to different states in the FSM

- Desirable property is a small *discriminative size*
## Benefits of Intelligent Sampling

<table>
<thead>
<tr>
<th>Random Sampling</th>
<th>Intelligent Sampling</th>
</tr>
</thead>
<tbody>
<tr>
<td>• $SV$ can grow into large size</td>
<td>• $SV$ is kept small</td>
</tr>
<tr>
<td></td>
<td>– Detection latency reduction</td>
</tr>
<tr>
<td>• Multiple incorrect states</td>
<td>• Less incorrect states in $SV$</td>
</tr>
<tr>
<td>– Increase of false alarms</td>
<td>– False alarms reduction</td>
</tr>
</tbody>
</table>
The Problem of Sampling an *Incorrect Message*

- What if an *incorrect* message is sampled?
  - The message is *incorrect* in current states, e.g., a message from buggy component

- Suppose $SV = \{ S1, S2 \}$ and $m3$ is changed to $m5$
  $\Rightarrow SV = \{ S5 \}$ (incorrect $SV$!, it should be $\{ S3 \}$)
Probabilistic State Vector Reduction: A Hidden Markov Model Approach

- Hidden Markov Model (HMM) used to reduce SV
  - An HMM is an extended *Markov Model* where states are not observable
  - States are hidden as in the monitored application

- Given an HMM, we can ask:
  
  \[ \text{The probability of the application being in any state, given a sequence of messages?} \]

  Cost is \( O(N^3L) \), \( N \): number of states, \( L \): sequence length

- The HMM is trained (offline) with application traces
State Vector Reduction with the HMM

- Monitor asks the HMM: \( \{p_1, p_2, \ldots, p_N\} \)
  
  \[ p_i = P(S_i | O), \quad S_i: \text{application state } i, \quad O: \text{observation’s sequence} \]

**Messages** (observations) \[\rightarrow\] **HMM** \[\rightarrow\] Sort by Probabilities

\[ p_1, \quad p_2, \quad \ldots, \quad p_N \]

\[\begin{aligned} p_3, \quad p_5, \quad \ldots, \quad p_1 \end{aligned}\]

\( \alpha = \text{Top } k \) probabilities \[\rightarrow\] \( \alpha \cap \text{SV} = \text{new SV} \)

New SV is **robust** to incorrect messages
Experimental Testbed: Java Duke’s Bank Application

- Simulates multi-tier online banking system

- User transactions:
  - Access account information, transfer money, etc.

- Application stressed with different workloads
  - Incoming message rate at Monitor varies with user load
Web Interaction: A Sequence of *calls* and *returns*

Distributed E-commerce System

Components:
Java servlets, JavaBeans, EJBs
### Error Injection Types

<table>
<thead>
<tr>
<th>Error Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response Delays</td>
<td>a response delay in a method call</td>
</tr>
<tr>
<td>Null calls</td>
<td>a call to a component that is never executed</td>
</tr>
<tr>
<td>Unhandled Exceptions</td>
<td>exception thrown by execution that is never caught by the program</td>
</tr>
<tr>
<td>Incorrect Message Sequences</td>
<td>change randomly the web interaction structure</td>
</tr>
</tbody>
</table>

- Errors are injected in components touched by web interactions
  - A web interaction is faulty if at least one of its components is faulty
### Performance Metrics Used in Experiments

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Accuracy</strong> (True Alarms)</td>
<td>% of <em>true</em> detections out of web interactions where errors were injected</td>
</tr>
<tr>
<td><strong>Precision</strong> (False Alarms)</td>
<td>% of <em>true</em> detections out of the total number of detections</td>
</tr>
<tr>
<td><strong>Detection Latency</strong></td>
<td>time elapsed between the error injection and its detection</td>
</tr>
</tbody>
</table>

**Example:**

<table>
<thead>
<tr>
<th>Web Interactions</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Error Injected</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td><strong>Detection</strong> (An alarm is signaled)</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Accuracy** = $2/3 = 0.67$  
**Precision** = $2/4 = 0.5$
Results: State Vector Reduction

• Peaks are not observed in intelligent sampling (IS)
  – IS capability of selecting messages with small discriminative size
• SV of size 1 is more frequent in IS
Results: Monitor vs. Pinpoint (Accuracy, Precision)

- **Pinpoint** (*NSDI ‘04*), traces paths through multiple components
- Use of **PCFG** to detect abnormal paths

- Monitor and Pinpoint expose similar levels of accuracy
- Precision in Monitor (1.0) is higher than in Pinpoint (0.9)
Results: Monitor vs. Pinpoint (Detection Latency)

- Detection latency in Monitor is in the order of **milliseconds**, while in Pinpoint is in **seconds**
- The PCFG has a high space and time complexity
Results: Memory Consumption (MB)

<table>
<thead>
<tr>
<th></th>
<th>Virtual Memory</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monitor</td>
<td>282.27</td>
<td>25.53</td>
</tr>
<tr>
<td>Pinpoint-PCFG</td>
<td>933.56</td>
<td>696.06</td>
</tr>
</tbody>
</table>

- Monitor doesn’t rely in large data structures
- PCFG in Pinpoint has high **space** and **time** complexity
  - $O(RL^2)$ and $O(L^3)$
    - $R$: number of rules in the grammar
    - $L$: size of a web interaction
- Pinpoint thrashes due to high memory requirements
Efficient Rule Matching

- Selectively match computationally expensive rules
  - Expensive rules don’t have to be matched all the time
- Rules are matched only if instability is present
Efficient Rule Matching Example: *Detecting Memory Leak*

- Efficiently detect memory leak in Apache web server
  - Memory leak injected probabilistically with web requests

- Expensive ARIMA-based rule to detect abnormal memory usage
  - Average matching latency reduced

<table>
<thead>
<tr>
<th>Rule Matching Criteria</th>
<th>Memory Leak Detected</th>
<th>Average Matching Latency (msec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Always matched</td>
<td>yes</td>
<td>19.283</td>
</tr>
<tr>
<td>$\sigma \geq 0.5$</td>
<td>yes</td>
<td>7.115</td>
</tr>
<tr>
<td>$\sigma \geq 1.0$</td>
<td>no</td>
<td>1.25</td>
</tr>
</tbody>
</table>
Concluding Remarks

• Contributions:
  – **Sampling** used to **scale** stateful detection system under high-rate of messages
  – **Intelligent Sampling** reduces non-determinism caused by sampling
  – **HMM-approach** handles incorrect messages
  – Techniques can be applied to any stateful detection system
  – Monitor performs better than other approaches

• Future Work:
  – **Efficient Rule Matching** technique will be extended
  – **Sampling** only sequences of messages that lead to errors
  – Automatic generation of rules from traces