Dangers and Joys of Stock Trading on the Web: Failure Characterization of a Three-Tier Web Service

Fahad A. Arshad and Saurabh Bagchi
Dependable Computing Systems Lab (DCSL)
School of Electrical and Computer Engineering
Purdue University

Presented by: Zbigniew Kalbarczyk

Motivation

• Why web services are important?
  – A way to do e-business and communicate online
• Why is failure characterization needed?
  – To evaluate and improve robustness of a given service
• Why do failures occur in web services?
  – Also, how do failures manifest themselves
• Where do failures occur?
  – Network, OS, VM, application server, application
• How do we come up with a rigorous analysis?
  – Bug databases
  – Fault Injection
Target Domains

• Stock Trading Systems
• Banking Systems
• E-Stores
• Auction Systems
• Travel industry

Fault Injection to Emulate errors

• What kind of faults to inject?
  – Undeclared exceptions
  – Null-call variants
• When to inject the faults?
  – On method invocation
• Where to inject the faults?
  – EJB container
Fault Injection: Null-call and Unchecked Exceptions

- Null-call
  - Null-Return
  - Null-Object-Return
  - No-Op
- Unchecked Exceptions
  - Arithmetic Exception
  - IndexOutofBounds Exception
  - ClassCastException

Fault Emulation: example code

```
foo()
...
RObject x = bar();
...
RObject bar(){
    return RObject;
}
Original Code

foo()
    //RObject x = bar();
    RObject x = null;
...
Null-Return Code
```

```
foo()

    //RObject x = bar();
    RObject x = RObject.getClass();
    RObject x = RObjectClass.cast(null);
...
Null-Object-Return Code
```

```
foo()
...
RObject x = bar();
...
RObject bar(){
    throw new java.lang.RuntimeException();
    ...
    return RObject;
}
Unchecked Exception Code
```

```
foo()
...
RObject x = bar();
...
...
No-Op Code
```

Application: Three-tier Web Service (DayTrader)

- Front-end presentation in web container
- Middle-tier business logic in EJB container
- Back-end data-source in Derby database
- Faults injected in business logic in EJB container

Failure Manifestation and Classification

- Non-silent
  - Explicit error messages from infrastructure to user
  - E.g.: HTTP 5xx, blank page
- Non-silent-interactive
  - Partially correct response, only noticeable interactively
  - 3 results returned for 5 getQuote requests
- Silent
  - Unnoticeable to user or admin
  - E.g.: I buy 100 stocks of IBM and tomorrow I do not find them in my portfolio
  - Most worrisome class
### Failure Distribution

**By Error Class**

- Null-Return: 40% No-Silent, 40% Silent, 20% Null-Object Return
- Null-Object Return: 24% No-Silent, 33% Silent, 43% Null-Object Return
- No-Op: 36% No-Silent, 27% Silent, 37% Unchecked Exception
- Unchecked Exception: 33% No-Silent, 40% Silent, 27% Null-Object Return

**By where failures are logged**

- No-Op: 73% Non-Silent
- Null-Object Return: 67% Non-Silent Interactive, 33% Non-Silent
- Null-Return: 33% Non-Silent Interactive, 67% Non-Silent
- Null-Return: 33% Non-Silent Interactive, 67% Non-Silent

- No-Op causes significant silent errors
- Unchecked Exceptions cause significant non-silent errors
- Majority of non-silent failures are logged by CONTAINER

### How to detect these failures?

- **Detection Checks**
  - Application Generic
    - Null-call check
  - Application Specific
    - Call-Length check
    - Head-Tail check

- Detection example:

  ```java
  foo()
  bar()
  If(null) {
    Detect
  }

  If(HEAD != “foo” && TAIL != “baz”){
    Detect
  }

  Normal Call-length = 3
  ```
ACCURACY: Application Generic Null-call Check

- Null-Return is caught in all 3 failure classes (100% accuracy)

- No-Op is not detected by this check except for Logout

<table>
<thead>
<tr>
<th>Type of Injection</th>
<th>Failure Class</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Null-Return</td>
<td>Silent</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>Non-Silent</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>Non-Silent-Interactive</td>
<td>100%</td>
</tr>
<tr>
<td>Null-Object-Return</td>
<td>Silent</td>
<td>30%</td>
</tr>
<tr>
<td></td>
<td>Non-Silent</td>
<td>22%</td>
</tr>
<tr>
<td></td>
<td>Non-Silent-Interactive</td>
<td>35%</td>
</tr>
<tr>
<td>No-Op</td>
<td>Silent</td>
<td>1%</td>
</tr>
<tr>
<td></td>
<td>Non-Silent</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>Non-Silent-Interactive</td>
<td>0%</td>
</tr>
</tbody>
</table>

Application Specific: Call-Length Check

- Data-independent
  - Login=5

- Data-dependent
  - Variable call-length, due to different number of stocks owned or searched by a user
    - DB: Portfolio ≥ 2
    - USER: Quotes ≥ 1

```
  Call-length
  Web Request

  Constant
  Data-independent

  Variable
  Data-dependent

  DB
  USER
```
ACCURACY: Application Specific Call-Length Check

- Low detection accuracy for silent failures
- 60% of non-silent failures from No-Op detected
- Data-dependent web requests not detected

<table>
<thead>
<tr>
<th>Type of Injection</th>
<th>Failure Class</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Null-Return</td>
<td>Silent</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>Non-Silent</td>
<td>48%</td>
</tr>
<tr>
<td></td>
<td>Non-Silent-Interactive</td>
<td>30%</td>
</tr>
<tr>
<td>Null-Object-Return</td>
<td>Silent</td>
<td>1%</td>
</tr>
<tr>
<td></td>
<td>Non-Silent</td>
<td>57%</td>
</tr>
<tr>
<td></td>
<td>Non-Silent-Interactive</td>
<td>11%</td>
</tr>
<tr>
<td>No-Op</td>
<td>Silent</td>
<td>0.2%</td>
</tr>
<tr>
<td></td>
<td>Non-Silent</td>
<td>60%</td>
</tr>
<tr>
<td>Unchecked</td>
<td>Silent</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>Non-Silent</td>
<td>41%</td>
</tr>
</tbody>
</table>

Application Specific: Head-Tail Check

- Match first and last EJB request names
  - `getCloredOrders` → `getHoldings` → `getQuote` → `getQuote` → `getQuote`
- Implemented using ThreadLocal API
- Able to detect some data-dependent requests i.e. “Portfolio”
  - Detects Portfolio only when target of injection is either `getCloredOrders` or `getHoldings` and request is cut-short.
  - Expected tail ejb-request is `getQuote` from learning which will fail to satisfy the check if the web request is cut short.
ACCURACY: Application Specific Head-Tail Check

- Unable to detect silent failures
- Able to detect data-dependent (DB) web requests like Portfolio with low accuracy

<table>
<thead>
<tr>
<th>Type of Injection</th>
<th>Failure Class</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Null-Return</td>
<td>Silent</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>Non-Silent</td>
<td>45%</td>
</tr>
<tr>
<td></td>
<td>Non-Silent-Interactive</td>
<td>36%</td>
</tr>
<tr>
<td>Null-Object-Return</td>
<td>Silent</td>
<td>1%</td>
</tr>
<tr>
<td></td>
<td>Non-Silent</td>
<td>60%</td>
</tr>
<tr>
<td></td>
<td>Non-Silent-Interactive</td>
<td>11%</td>
</tr>
<tr>
<td>No-Op</td>
<td>Silent</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>Non-Silent</td>
<td>68%</td>
</tr>
<tr>
<td>Unchecked</td>
<td>Silent</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>Non-Silent</td>
<td>42%</td>
</tr>
</tbody>
</table>

Parameter Learning: Call-length and Head-Tail check

Login: Call-Length Distribution

Login: Tail EJB Request Distribution
Lessons Learned

• Caller should flag a returned *null-call*
  – Send a failure notification to end user to make it non-silent
  – Log the flagged null-call for better log quality

• Sanity checks at caller for *Null-Object-Return*
  – E.g., Check whether the size of the returned object is greater than a threshold

• *No-Op* are hard to detect in app-generic way
  – Application specific checks requiring low implementation overhead help to detect No-Op

Lessons Learned

• Explicit catch blocks for common Unchecked exceptions
  – Arithmetic Exception (Unforeseen calculation error)
  – IndexOutOfBoundsException Exception (Unintended Array manipulation)
  – ClassCastException (Unintended wrong Object casting)

• Mechanisms to make silent errors non-silent
  – E.g., Log analysis

• Data-dependent request are hard to detect
  – Much more deep application specific checks that require additional runtime information
Future Directions: Detect Data-dependent Requests

- Each data-dependent request has a *constant* and a *variable* part in normal case
  
  \[ \text{Portfolio}: \text{getClosedOrders( )}\rightarrow \text{getHoldings- } \]
  
  \[ \rightarrow \text{getQuote()}* \]

- Extract data-dependent information at runtime
- Match expected runtime call-length to observed call-length

Future Directions

- Failures due to concurrency in java based web services
- Failures due to using different design and architectural patterns in a given three-tier web service
- Identifying design patterns that lead to robust web services
Thank you

Backup Slides
Implementation: Call-Length Check

- Use ThreadLocal API
- Monitor no of EJB requests (call-length) invoked for a given web request
- Detect at the end of a given web request

```
sell : getClosedOrders, sell, updateQuotePriceVolume
quotes : getClosedOrders, getQuote
update_profile : getClosedOrders, updateAccountProfile, getAccountData, getAccountProfileData
quotes : getClosedOrders, getQuote, getQuote, getQuote, getQuote
register : register, login, getAccountData, getHoldings, getMarketSummary
login : getClosedOrders, login, getAccountData, getHoldings, getMarketSummary
buy : getClosedOrders, buy, updateQuotePriceVolume
logout : logout
home : getClosedOrders, getAccountData, getHoldings, getMarketSummary
portfolio : getClosedOrders, getHoldings, getQuote, getQuote
account : getClosedOrders, getAccountData, getAccountProfileData
portfolio : getClosedOrders, getHoldings, getQuote
```

Sample Web and EJB requests

Application Generic: Null-call Check (PRECISION)

- Logout results in low precision since it expects a null on return

![Precision Chart]

- Precision (Null-Return)
- Precision (Null-Object-Return)
- Precision (No-Op)
PRECISION of Application Specific Call-Length Check

- Data-dependent web requests like portfolio and quotes are not detected.

Application Specific Head-Tail Check PRECISION

- DB Data-dependent web requests like portfolio are detected with reasonable precision.