Mojave: A Recommendation System for Software Upgrades

Rekha Bachwani, Olivier Crameri, Ricardo Bianchini, Willy Zwaenepoel
Motivation

- Modern software is complex; requires frequent updates
  - Fix bugs
  - Patch security vulnerabilities
- Software upgrade failures are frequent
  - 5-10% of all upgrades fail [SOSP’07]
  - 41% of bugs reported in OpenSSH due to upgrades
- Users’ environment and input cause upgrade failures
  - Application-specific configurations
  - System environment settings
Current Techniques

- Deploy upgrades as packages
  - Package management systems check for static dependencies

- Delay installation till the upgrade is “mature”
  - Wait for positive feedback from (many) other users

None of these approaches is ideal
Approach

Developer and user collaboration

- Integrate users in upgrade deployment cycle
  - Test upgrade in (many) user environments with their input

- Collect data from the (willing) users
  - Environment settings
  - Dynamic execution behavior
  - Success or failure flags

- Leverage data from many users
- Prevent failures for new users
Contributions

- **Mojave**: Recommendation system for upgrades
  - Provides accurate recommendations
  - Predicts the likelihood of an upgrade failure
  - Uses machine learning, environment & run time data
  - Evaluation with two OpenSSH upgrade failures
Outline

- Overview
- Mojave: A Recommendation System
- Evaluation
- Conclusion
Mojave - Key Idea

- Upgrades fail mostly because of users’ attributes
  - Environment settings
  - Inputs (execution behavior)
- Users similar to other users where upgrade failed
  - Likely to experience similar failures
- “Alike” before the upgrade → similar behavior after it

- Learns failure characteristics
- User similarity to predict failure likelihood
Mojave - Learning Phase

Developer

Initial Users

Upgrade

Upgrade

Upgrade
Mojave - Learning Phase

Environment data
Call sequences
Success/Failure flags

Developer

Initial Users
Mojave - Learning Phase

**Developer**

- Environment data
  - Call sequences
  - Success/Failure flags
- Feature Selection
- Source Analysis

Filtered call sequences and success/failure flags

Filter

Longest Common Subsequence (LCS)

90^{th} percentile length of LCS

Environment and success/failure data

Logistic Regression

FSimilarity, SSimilarity

Prediction Model

Suspect Routines

Suspect Environment Features
Mojave - Recommendation Phase

- Call sequences
- Environment data

Prediction Model
Mojave - Recommendation Phase

Developer

Call sequences
Environment data

Past users' call sequences and success/failure flags

Longest Common Subsequence (LCS)

90th percentile length of LCS

FSimilarity, SSimilarity

Prediction Model

Recommend FOR

Recommend AGAINST

New User
Summary

- Collects environment and run time data from users
- Learns user attributes correlated with the failure
  - Machine learning, call sequence similarity, and static and dynamic analyses
- Compares new user’s attributes to those of past users
  - Call sequence similarity and machine learning
- Recommends in favor or against an upgrade
Outline

- Overview
- Mojave: A Recommendation System
- Evaluation
- Conclusion
Failures – Port Forwarding

- Large data transfers abort when using port forwarding
  - Regression bug in ssh version 4.7
  - Abort not reproducible at developer site

- Abort characteristics
  - Users had port forwarding (Tunnel) enabled
  - Default window size increased from 128KB to 2MB
  - Port forwarding code advertising window size as packet size
  - sshd limits maximum packet size to 256KB
Failures – X11 Forwarding

- X forwarding won't start when executed in background
  - Regression bug in `sshd` version 4.2

- Failure characteristics
  - Users had X11 forwarding (`X11Forwarding`) enabled
  - X11 forwarding code modified to fix channel leaks
  - Destroys X11 connections whose session has ended
  - Connections started in background close session immediately
Experimental Setup

- **Upgrade deployment**: environment data from 87 machines
- **8 real application configs**: 3 have failure settings
- **8 inputs**: 3 inputs that activate failures
- **Training set has 57 profiles, remaining 30 test profiles**
- **Feature selection**
  - 20 fail profiles, 67 success profiles
  - Features within 30% of the top-ranked feature considered suspect

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Type of values</th>
<th>No. of profiles</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>System</td>
<td>Application Specific</td>
</tr>
<tr>
<td>Perfect (100%)</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td>Imperfect(60%)</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td>Imperfect(20%)</td>
<td>Real</td>
<td>Real</td>
</tr>
</tbody>
</table>

Upgrade deployment: environment data from 87 machines. 8 real application configs: 3 have failure settings. 8 inputs: 3 inputs that activate failures. Training set has 57 profiles, remaining 30 test profiles. Feature selection - 20 fail profiles, 67 success profiles. Features within 30% of the top-ranked feature considered suspect.
## Recommendation Results

<table>
<thead>
<tr>
<th>Bug</th>
<th>Experiment</th>
<th>Initial Users (Training Data)</th>
<th>New Users (Test Data)</th>
<th>Recommendation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Success</td>
<td>Failure</td>
<td>Success</td>
</tr>
<tr>
<td>Port Forwarding</td>
<td>Perfect (100%)</td>
<td>42</td>
<td>15</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>Imperfect (60%)</td>
<td>48</td>
<td>9</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>Imperfect (20%)</td>
<td>34</td>
<td>3</td>
<td>29</td>
</tr>
<tr>
<td>X11 Forwarding</td>
<td>Perfect (100%)</td>
<td>42</td>
<td>15</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>Imperfect (60%)</td>
<td>48</td>
<td>9</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>Imperfect (20%)</td>
<td>34</td>
<td>3</td>
<td>29</td>
</tr>
</tbody>
</table>

- Produces accurate recommendations: **96-100% accuracy**
- Mispredicts one failures: closer to success profiles
- Prevents upgrade failures for most new users
Outline

➢ Overview

➢ Mojave: A Recommendation System

➢ Evaluation

➢ Conclusion
Conclusion

Mojave: first upgrade recommendation system

- Integrates users in the upgrade deployment cycle
- Leverages past similarity between user attributes
- Uses a novel combination of techniques
  - Machine learning
  - Static and dynamic analyses
  - Program behavior similarity
- Prevents upgrade failures for most new users
Thanks for your time!

Questions?