UPGRADVISOR: Early Adopting Dependency Updates Using Hybrid Program Analysis and Hardware Tracing

Yaniv David\textsuperscript{1}, Xudong Sun\textsuperscript{2}, Raphael J Sofaer\textsuperscript{1}, Aditya Senthilnathan\textsuperscript{3}, Junfeng Yang\textsuperscript{1}, Zhiqiang Zuo\textsuperscript{2}, Guoqing Harry Xu\textsuperscript{4}, Jason Nieh\textsuperscript{1} and Ronghui Gu\textsuperscript{1}

\textsuperscript{1}Columbia University  \textsuperscript{2}Nanjing University  \textsuperscript{3}IIT Delhi  \textsuperscript{4}UCLA
Modern Software Development is fast-paced

- Facebook(Meta) updates their front-end three times a day, and release new iOS and Android apps every week

- A key enabler for new features is pre-existing libraries

- Average of 12 direct dependencies 100+ transitive dependencies
  (Our survey of top-starred Python, JS, Ruby GitHub projects)
Dependency Update Adoption Is Slow

- Dependencies' developers are releasing updates frequently, too

- Currently averaging **400 days** in update adaption delay
Dependency Delays Lead to Bad Consequences

- Fixed bugs in dependencies continue to affect applications
- Closed security holes put dependent applications at risk
- Conflicts arise in transitive dependency graphs
  - Some can be resolved by using the oldest supported version
  - Other fall into a “dependency hell”
Dependency Delays Lead to Bad Consequences

- Fixed bugs in dependencies continue to affect applications
- Closed security holes put dependent applications at risk
- Conflicts arise in transitive dependency graphs
  - Some can be resolved by using the oldest supported version
  - Others fall into a “dependency hell”
Naïve Solutions Fall Short

- As dependency APIs change, blindly updating might fail
  - Noisy run-time crash
Naïve Solutions Fall Short

- As dependency APIs change, blindly updating might fail
  - Noisy run-time crash

[Code Snippet]

```python
TypeError: Traceback (most recent call last)
<ipython-input-7-ea5a420416fe> in <module>()
----> 1 wikiann_corpus: Corpus = WIKIANN(languages=['ar'])
      2 print(wikiann_corpus)

/usr/local/lib/python3.7/dist-packages/flair/datasets/sequence_labeling.py in __init__(self, languages, basi

2044       # download from google drive
2045   gdown.download(url, str(language_folder / language) + '.tar.gz')
2046
```

`TypeError: download() missing 1 required positional argument: 'quiet'`
Naïve Solutions Fall Short

- As dependency APIs change, blindly updating might fail
  - Noisy run-time crash
  - Or worse, fail silently

```python
- async def spawn(coro, *args, loop=None, report_crash=True):
-   return spawn_sync(coro, *args, loop=loop, report_crash=report_crash)
+ async def spawn(coro, *args, loop=None, daemon=False):
+   return spawn_sync(coro, *args, loop=loop, daemon=daemon)
```
Naïve Solutions Fall Short

- As dependency APIs change, blindly updating might fail
  - Noisy run-time crash
  - Or worse, fail silently

- Integration tests fail or don’t even try covering dependency interfaces\(^1\)

\(^1\)Joseph Hejderup & Georgios Gousios, “Can we trust tests to automate dependency updates? A case study of Java Projects”, Journal of Systems and Software
Naïve Solutions Fall Short

- As dependency APIs change,
  - Noisy run-time crash
  - Or worse, fail silently
- Integration tests fail or don’t even try covering dependency interfaces
- Manual inspection is not feasible

\[ \times \text{[#updates]} \times \text{[#deps]} \]
UPGRADVISOR: Upgrade-Advisor

Application Code

Dependency Code

Average Overhead 3% (Max 6%)

UPGRADVISOR

Production Servers

56%
Upgrade SAFE

90%
Reduced Diff

Upgrade not safe

Built UPGRADVISOR-Python3 and evaluated on 172 dependency updates
UPGRADVISOR: Upgrade-Advisor

eddiebergman commented on Dec 12, 2021

Hi @Yanivmd, thanks for the personal response. We do some extra steps due to the kinds of shared environments our users sometimes have when using this library, the extra output can help them debug things.

Cool bot by the way, looking forward to seeing what future progress you make!

Manually submitted a sample of 9 PR 7 already merged

alanakbik commented 26 days ago

@rsofaer wow very interesting project! Managing dependencies is definitely a lot of overhead, so anything that helps us here is greatly appreciated!
Insight: What You Can’t Reach Won’t Hurt You

Application Code

Updated Dependency Code

Production
Insight: What You Can’t Reach Won’t Hurt You

Requirements for production run:

- Study the update \textbf{without} applying it
- No interruption
- Incur low overhead
Insight: What You Can’t Reach Won’t Hurt You

Requirements for production run:

- Study the update **without** applying it
- No interruption
- Incur low overhead

Tracing production environment over time can serve as **ground truth** for dependency usage
Key Ideas Driving UPGRADEVISOR’s Design

- Safely discard non-reachable changes via hybrid program analysis
  - Static analysis to discard \textit{never-reachable} changes
  - Dynamic analysis to test \textit{maybe-reachable} changes

- Achieve low-overhead by employing hardware-based tracer
Key Ideas Driving UPGRADEVISOR’s Design

- Safely discard non-reachable changes via hybrid program analysis
  - Static analysis to discard never-reachable changes
  - Dynamic analysis to test maybe-reachable changes
- Achieve low-overhead by employing hardware-based tracer

The 2021 State of the

Octoverse

<table>
<thead>
<tr>
<th>Year</th>
<th>JavaScript</th>
<th>Python</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2019</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2020</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2021</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- npm: 2.42 M
- Python Package Index: 455 K
- RubyGems: 179 K
Key Ideas Driving UPGRADVISOR’s Design

- Safely discard non-reachable changes via hybrid program analysis
  - Static analysis to discard *never-reachable* changes
  - Dynamic analysis to test *maybe-reachable* changes
- Achieve low-overhead by employing hardware-based tracer
- Design for dynamic languages to maximize usability
Analyzing Our Motivating Example
Analyzing Our Motivating Example

QLib 0.7.1

HyperOpt v0.1.1

Call-Graph

Call-Graph
Analyzing Our Motivating Example

QLib 0.7.1

HyperOpt v0.1.1

App → Dependency Call
Analyzing Our Motivating Example

QLib 0.7.1

HyperOpt v0.1.1

HyperOpt v0.1.2
```python
def main_worker_helper(...):
    if os.name != 'nt':
        + signal(SIGUP, hdlr_shutdown)
        - signal(SIGUP, hdlr_shutdown)
        signal(SIGINT, hdlr_shutdown)
```

Classify Changes In Motivating Example

QLib 0.7.1

HyperOpt v0.1.1

HyperOpt v0.1.2

Never-reachable
Classify Changes In Motivating Example

QLib 0.7.1

Only trace maybe-reachable changes

Never-reachable

HyperOpt v0.1.1

HyperOpt v0.1.2
Classify Changes In Motivating Example

```python
def serial_evaluate(self, ...):
    for trial in self.dyn_trials:
        if trial['state'] == NEW:
            trial['state'] = RUNNING
...
Classify Changes In Motivating Example

<table>
<thead>
<tr>
<th>Changes</th>
<th>Initial Count</th>
<th>Discarded</th>
<th>Left</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functions</td>
<td>72</td>
<td>68 (94%)</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

QLib 0.7.1

HyperOpt v0.1.1

HyperOpt v0.1.2
Classify Changes In Motivating Example

v0.8.1

github-actions released this Jan 15, 2022 · 164 commits to main since this release

Changes

Features

- pylint code refine & Fix nested example @you-n-g (#848)
- chore: remove hard code input dimension of model pytorch_tcts @PalanQu (#843)
- [840] - Test case for operators. @ChiahungTai (#841)
- DDG-DA paper code @you-n-g (#743)
- Update BCEloss in MLP model @cuicorey (#756)
- solve VERSION.txt bug @b4thesunrise (#732)
- **Hyperopt upgrade @upgradvisor-bot (#741)**
- Add method parameter for volume @you-n-g (#734)
Key Challenges for Designing UPGRADVISOR

- **Hybrid program analysis to safely discard non-reachable changes**
  - Safely discard non-reachable changes via hybrid program analysis
    - Create sound call-graphs
    - Reachable-but-non-affecting changes
  - Dynamic analysis to test maybe-reachable changes

- **Achieve low-overhead using a hardware-based tracer**
  - Low-overhead & selective tracing
Key Challenges for Designing UPGRADVISOR

- Hybrid program analysis to safely discard non-reachable changes
  - Safely discard non-reachable changes via hybrid program analysis
    - Create sound call-graphs
    - Reachable-but-non-affecting changes
  - Dynamic analysis to test maybe-reachable changes

- Achieve low-overhead using a hardware-based tracer
  - Low-overhead & selective tracing
void foo(int a) {
    if (a == 0) {
        // something
    } else {
        // something else
    }
}

Jump Not Taken

cmp rdi, 0
jne .EL
nop
jmp RET
EL
nop
.RET
ret

Recreate Trace Offline

Cyclic-write RAM buffer (Usually dumped to disk)
def foo(a):
    if a == 0:
        # something
    else:
        # something else

LOAD_GLOBAL 0 (a)
LOAD_CONST 1 (0)
COMPARE_OP 2 (==)
POP_JUMP_IF_FALSE 10

switch (bc){
    case LOAD_GLOBAL:
        // do load global
        break;
    case LOAD_CONST:
        // do load const
        break;
    case ....:
}

JPortal: Precise and Efficient Control-Flow Tracing for JVM Programs with Intel Processor Trace [PLDI ’21]
Hardware Tracing for Interpreter Code

```python
def foo(a):
    if a==0:
        # something
    else:
        # something else
```

```
for (ByteCode bc : allcode):
    switch (bc) {
        case LOAD_GLOBAL:
            // do load global
            break;
        case LOAD_CONST:
            // do load const
            break;
        case ...:
    }
```
def foo(a):
    if a==0:
        # something
    else:
        # something else

LOAD_GLOBAL 0 (a)
LOAD_CONST 1 (0)
COMPARE_OP 2 (==)
POP_JUMP_IF_FALSE 10

def foo(a):
    if a==0:
        # something
    else:
        # something else

LOAD_GLOBAL 0 (a)
LOAD_CONST 1 (0)
COMPARE_OP 2 (==)
POP_JUMP_IF_FALSE 10

for (ByteCode bc, allcode):
    switch
    case LOAD_GLOBAL:
        // do load global
        break;
    case LOAD_CONST:
        // do load const
        break;
    case ...:

Recreate Bytecode Trace

Recreate Interpreter Trace

High overhead & Data loss
Hardware Tracing for Interpreted Code

Only trace this code segment

Only Selected Functions

code segment

Only trace maybe-reachable changes
Hardware Tracing for Interpreted Code

```java
for (ByteCode bc : all_code)
switch (bc){
    case Op_Code_1:
        // do Op_Code_1
        break;
    case POP_JUMP_IF_F:
        jump_to_trace1();
        // do POP_JUMP_IF_F
        break;
    case ... :
        jump_back_trace2();
        jump_back_trace3();
    }
```
Hardware Tracing for Interpreted Code

```java
for (ByteCode bc : all_code)
    switch (bc) {
        case Op_Code_1:
            // do Op_Code_1
            break;
        case POP_JUMP_IF_F:
            jump_to_trace1();
            // do POP_JUMP_IF_F
            break;
        case : ... 
    }
```

Selectivity:
- Only trace this code segment
- Low-overhead
Evaluation – Facilitating Dependency Updates

Updateable: 172

Static-Safe: 98 (56%)

Tracing Required: 74 (44%)

For 5 Projects: Production-Like Tracing
The Dynamic Tracing Contribution

<table>
<thead>
<tr>
<th>Project (Dependency)</th>
<th>Diff (LOC)</th>
<th>% Discarded</th>
<th>% Left</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Static</td>
<td>Dynamic</td>
</tr>
<tr>
<td>AutoML(Distributed)</td>
<td>820</td>
<td>95</td>
<td>5</td>
</tr>
<tr>
<td>Electrum (qdarkstyle)</td>
<td>641</td>
<td>88</td>
<td>8</td>
</tr>
<tr>
<td>Flair (gdown)</td>
<td>1500</td>
<td>71</td>
<td>29</td>
</tr>
<tr>
<td>Qlib (Hyperopt)</td>
<td>828</td>
<td>90</td>
<td>9</td>
</tr>
<tr>
<td>Scylla (requests)</td>
<td>449</td>
<td>90</td>
<td>8</td>
</tr>
</tbody>
</table>
Tracer Overhead Testing Setup

- Selected Python projects with robust test-suites from our data-set
  - For Django’s also running in parallel: using 1, 8, and 16 cores
Tracer Overhead Testing Setup

- Selected Python projects with robust test suites from our data set
  - For Django’s also running in parallel: using 1, 8, and 16 cores
- UPGRADEVISOR-Targeted
Tracer Overhead Testing Setup

- Selected Python projects with robust test suites from our data-set
  - For Django’s also running in parallel: using 1, 8, and 16 cores

- UPGRADVISOR-Targeted
- UPGRADVISOR-ALL
- Jportal4Py

```c
for (ByteCode bc : allcode)
    switch (bc){
    case LOAD_GLOBAL:
        // do load global
        break;
    case LOAD_CONST:
        // do load const
        break;
    case ....:
    }```


Tracer Overhead Testing Setup

- Selected Python projects with robust test-suites from our data-set
  - For Django’s also running in parallel: using 1, 8, and 16 cores

- UPGRADEVISOR-Targeted
- UPGRADEVISOR-ALL
- Jportal4Py
- UPGRADEVISOR-SW
UPGRADVISOR’s Tracer Incurs Low-Overhead
UPGRADVISOR’s Tracer Incurs Low-Overhead

In the paper: Compare to cProfile & Coverage.py
Instagram’s Django performance workbench
Conclusion

- We presented UPGRADVISOR: a system for reducing developer effort and error risk in adopting dependency updates
- Want to know more? See our website! https://upgradvisor.github.io
- Want to use UPGRADVISOR-Python3? Install our free GitHub App
Conclusion

- Want to use UPGRADVISOR? Install our free GitHub App
Hi there! Upgrador has identified that one of your repository's dependencies has a newer version available, and we recommend you upgrade.

Your code currently pins requests 2.26.0, when requests 2.27.1 is available. Our analysis indicates that the impacts from this upgrade may fix a bug in scylla. The attached graph shows the dependency path of your repository relative to requests. Your code is shown in green (each node is a method), and your code calling requests is shown in orange. Changes between version 2.26.0 and 2.27.1 are shown as starred.
Hi there! Upgradvisor has identified that one of your repository's dependencies has a newer version available, and we recommend you upgrade.

Your code currently pins requests 2.26.0, but this upgrade may fix a bug in Scylla. The next version of requests is 2.26.1. Your code is shown in green (e.g., `validator:Validator.validate_proxy`). Changes between version 2.26.0 and 2.26.1 can be seen in the diagram.

Upgradvisor is a research project from Columbia University's RCS lab. Our goal is to provide meaningful information for developers (like you) on how to upgrade their software dependencies. If you have any questions or feedback please reach out to the Upgradvisor team at yaniv.david@columbia.edu.
Thank You! Questions?

**UPGRADVISOR**: a system for reducing developer effort and error risk in adopting dependency updates

- **Application Code**
- **Dependency Code**
- **Updated Dependency Code**

- **UPGRADVISOR**
  - 56% Upgrade SAFE
  - 90% smaller
  - Average Overhead 3% (Max 6%)

- **Production Servers**

**HW-based Tracing is Production-Ready**

```
for (ByteCode bc : all_code)
    switch (bc){
    case Op_Code_1:
        // do Op_Code_1
        break;
    case POP_JUMP_IF_F:
        jump_to_trace1();
        // do POP_JUMP_IF_F
        break;
    case : ...
    }
```

**Dependency Update Adoption Is Slow**

- **Python**
- **Java**
- **Ruby**

**Average of Delay Days**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Ruby</td>
<td>200</td>
<td>300</td>
<td>400</td>
</tr>
<tr>
<td>Java</td>
<td>300</td>
<td>400</td>
<td>500</td>
</tr>
<tr>
<td>Python</td>
<td>400</td>
<td>500</td>
<td>600</td>
</tr>
</tbody>
</table>

**UPGRADVISOR**

- 56% Upgrade SAFE
- 90% smaller
- Average Overhead 3% (Max 6%)

**Production Servers**