Noncommittal Commits: Predicting Performance Slowdowns in Version Control History

Daniel Nichols¹, Dilan Gunawardana¹, Aniruddha Marathe², Todd Gamblin³, Abhinav Bhatele¹
¹University of Maryland, College Park, ²Lawrence Livermore National Laboratory

Abstract

Scientific software in high performance computing is becoming increasingly complex both in terms of its size and the number of external dependencies. Correctness and performance issues can become more challenging in actively developed software with increasing complexity. This leads to software developers having to spend larger portions of their time on debugging, optimizing, and maintaining code. Making software optimization and maintenance easier for developers is paramount to accelerating the rate of scientific progress. Fortunately, there is a wealth of data on scientific coding practices available implicitly via version control histories. These contain the state of a code at each stage throughout its development via commit snapshots. Commit snapshots provide dynamic insight into the software development process that static analyses of release tarballs do not. In this poster, we present a methodology for:

- Collecting performance data across version control history
- Pruning Abstract Syntax Trees (AST) based on profiled Calling Context Trees (CCT)
- Computing edit scripts that map one AST to another
- Using deep learning to predict performance degradation based on code changes

Data Collection & Preprocessing

We collect performance data at each commit for 2 proxy applications: Kripke¹ and Laghos². Each commit is run on the Quartz system at Lawrence Livermore National Laboratory (LLNL).

<table>
<thead>
<tr>
<th>Augmentation Method</th>
<th>Kripke Data Set #</th>
<th>Lagos Data Set #</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>107</td>
<td>1168</td>
</tr>
<tr>
<td>Reflection</td>
<td>212</td>
<td>2302</td>
</tr>
<tr>
<td>Commit Windows</td>
<td>1712</td>
<td>18,688</td>
</tr>
<tr>
<td>Splitting by Function</td>
<td>5071</td>
<td>31,140</td>
</tr>
</tbody>
</table>

To get more training samples we augment the data set with 3 methods:
- Reflection: swap commits and output label
- Commit Windows: use all commit pairs within a range of commits
- Splitting by Function: split commit by changes per function

Machine Learning Pipeline

Commit hash 1

Commit hash 2

AST Generation

Commit windows

Prune ASTs with run time data

AST

CCT & AST

Build & Run

Source Code

CCT & AST

GumTree Algorithm

Edit Script

LSTM

Model

References

[1] https://github.com/LLNL/Kripke

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-POST-838843).

Acknowledgements

We have shown that by using data engineering techniques we can utilize deep learning to predict performance degradation in version control history. In future work we plan to:

- Use transfer learning to extend to new applications
- Use causal learning to gain insight into what types of code changes cause performance degradation
- Improve end-result by predicting relative performance
- Suggest code improvements

Long short-term memory (LSTM) model classifies with up to 0.81 F₁ score and 78% accuracy when cross-validated on data set.

The model used is LSTM→Dropout→Dense with two 256 unit LSTM layers and 0.2 dropout.

The LSTM model outperforms simpler models and shows the representative capacity of the edit scripts.

Conclusion and Future Work

We have shown that by using data engineering techniques we can utilize deep learning to predict performance degradation in version control history. In future work we plan to:

- Use transfer learning to extend to new applications
- Use causal learning to gain insight into what types of code changes cause performance degradation
- Improve end-result by predicting relative performance
- Suggest code improvements

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-POST-838843).

References

[1] https://github.com/LLNL/Kripke

Build & Run

Source Code

CCT & AST

GumTree Algorithm

Edit Script

LSTM

Model