"Performance Measurements Before Releases vs. Each Commits" – Experiment Idea

Seminar: Systems Benchmarking

Lukas Abelt

Advisors: Prof. Sven Apel,
Christian Hechtl

Saarland Informatics Campus,
Saarland University

1 General

This document will briefly outline the experiment idea for the topic "Performance Measurements Before Releases vs. Each Commits" in the seminar "Systems Benchmarking". The goal of this experiment and the corresponding paper is to showcase, how continous performance measurements throughout the whole development process can be used to detect and fix performance regressions at a very early stage.

To achieve this, the experiment will first analyze the performance of public releases an open-source project to find a performance regression between two releases. Then, a performance measurement will be done for every commit between these two releases. We will use the data aquired by these performance measurements to demonstrate how or if various change detection measurements could havedetected this regression early on.

2 Experiment Setup

For my experiment I will use the open-source compression library `lz4`[1] as it has a active commit and release history. For the performance measurement itself we will use the latest version of the open-source project `lzbench`[2]. Lzbench is an in-memory compression benchmarking tool that is originally designed to compare different compression algorithms. However for our case we will use it to simply only measure `lz4` at different releases/commits. For this case we will disable all other compression algorithms that `lzbench` supports at compile-time. First preliminary tests have shown that `lzbench` can easily use a different version of `lz4` than the shipped one by simply replacing the corresponding source files.

For the compression itself we will use the Silesia Data Corpus[3]. The Sileasia Corpus combines a variety of different data in order to create a diverse dataset for compression measurements.

[1] `https://github.com/lz4/lz4`
[2] https://github.com/inikep/lzbench
[3] http://sun.aei.polsl.pl/ sdeor/index.php?page=silesia

After identifying a performance regression between two releases we will collect data for the commits between these releases. To analyze the data collected for these commits we will compare the two following methods:

- Comparing based on a static threshhold to the last commit/release
- Using Change Point Detection (Specifically E-Divisive Means)[4]

For the change point detection we will use the implementation provided by MongoDB[5]. Since we will also store the raw performance data as returned by `lzbench` it is also possible to add further methods later on to get a better overview of different methods.

3 Challenges and possible risks

In this section I will briefly outline challenges, risks that might arise during the experiment and also outline possible solutions to them.

3.1 Detecting regression between releases

Defining a regression threshhold One of the main challenges will be to define a threshhold for a performance regression between commits. As a first starter we will work with a static threshhold of 10% as compared to the previous release. However if this value seems to many/less releases we might need to change the threshhold or overthink out strategy overall.

No regression detected It might happen that we will detect no noticable performance changes whatsoever between the individual releases. One likely reason for this might be that the actual performance regressions happened while testing the release and where therefore fixed in the actual release.

One solution to this might be to select some commit for measurement before the release commit to benchmark on. However this would require a lot of manual triaging to select the "correct" commit based on e.g. the commit or issue history of the project.

Then still it could happen that with that strategy we do not detect any performance regressions (or improvements) that we can use as a guideline where to perform our finer measurement. In that case we could:

- Select an arbitrary release range for our finer analysis – However this itself bears the risk that we do not detect any change points in this range
- Select a release range based on issues submitted on the GitHub page that indicate that there at least was some performance regression during this period
- Select another project for our experiment

[4] As described in `https://dl.acm.org/doi/pdf/10.1145/3358960.3375791`
[5] https://github.com/mongodb/signal-processing-algorithms

3.2 Measuring individual commits

Commits that won't build If we measure all commits between two releases it is very likely that not all commits will be able to compile. We can easily take this into account by simply skipping these commits and not perform any measurements if we detect a failure during build.

Number of commits Even when restricting us to the duration between two releases it might still happen that we end up with a lot of commits we need to benchmark individually. Depending how long one benchmark takes this could easily take too much time to finish in a reasonable time frame. Additionally we have to remember that also others might want to use the cluster environment for their experiments, so we should not clog it with hundreds of our jobs.

One possible measure to counteract this is to not measure each individual commit but squash them based on some criterion, e.g. only measuring one commit per day. In practice we will have to take a close look how long an individual performance measurement (including build etc.) takes so that we can esitmate how long our measurements would take in total.

4 Possible Extensions

There are a few possible extensions that could be appended to this experiment idea if time allows it:

- Using the data collected by individual commits to show how individual commits can be compared effectively[6]
- Extend the regression detection method
- Evaluate the impact on sustainability

[6] As shown in `https://arxiv.org/pdf/2101.10231.pdf`