Visualizing Software Evolution in Linux: A Hierarchical Graph-Based Heat Map

Adrian Volpe
College of Science and Math
Belmont University
Nashville, USA
adrian.volpe@bruins.belmont.edu

Esteban Parra
College of Science and Math
Belmont University
Nashville, USA
esteban.parrarodriguez@belmont.edu

***Abstract*—This project presents a visualization of Linux commit activity using a hierarchical, heat-map based graph. Using a large dataset of commit data from Zenodo, we model the Linux structure as a directed hierarchy. Each node in the graph represents a subsystem, and edges show a parent-child relationship within the Linux architecture. The graph is rendered radially in Unity with a color gradient applied to nodes to indicate the volume of commits made to each subsystem. The result is an interactive, intuitive view of development hotspots in the Linux kernel, aimed at supporting further software evolution analysis.**

***Index Terms*—commit, graph, heat-map**

I. Introduction

Linux is a family of open source operating systems that are based on the Linux kernel. The Linux kernel is a core component of a device that manages the hardware and resources. It's responsible for the CPU, memory and peripherals. Linux is versatile and is used everywhere from powering smartphones to operating smart TV's. Linux is fast, secure and highly scaleable, so it is important to learn how Linux changes over time [1].

Linux is an open source system, meaning that the source code is available, for free, to be modified and redistributed. However, the main Linux operating system (OS) is only modified by specific authors and each modification is checked thoroughly [2]. These changes to the Linux OS source code are called commits. Each commit contains certain data including: the author of the commit, the time the commit was made, every line changed, and the number of added and deleted lines. Commits are used to keep track of what code was changed, when it was changed and by who.

Analyzing commit history is useful for developers because it can show areas in the Linux kernel that have not been modified recently or that have been heavily modified. Areas that have recently been modified often are more likely to contain bugs or errors [**?**]. Thus, commit history visualization can strengthen our ability to find bugs in the Linux OS. We created a graph visualization of Linux subsystems using commit frequency as a heat metric. This visualization is specifically useful because it provides an interactive and clear way of seeing commit history. This approach has not been done before, in this way, and can provide a new way of seeing the relationship between the Linux OS structure and how commits are distributed across it [3].

In section II we explain works that inspired this tool as well as similarities and differences in adjacent works. Section III presents the architecture and design of our tool. Finally, section IV will conclude the paper with ideas for future work and limitations of the tool.

II. Related Work

I have added something to the LaTeX file, now will it show up on GitHub? Im gonna add even more chnages like this one here. Let me make a change here and see where it appears when I hit save.

III. Methodology

In this section we will present an overarching view of the architecture and design of our tool, from how we collecting the data to constructing and visualizing the graph.

A. Data Collection

The data we used came from the Zenodo dataset provided by VISSOFT [4]. This dataset is comprised of was given to us in the form of a JavaScript Object Notation (JSON) file. To read this file we created a script in Java to parse through the data and output a text file. The text file held information on the relationships between the subsystems, the number of commits on each subsystem, and a color corresponding with each subsystem determined by its number of commits. This text file is what we use to construct and render the graph.

B. Color Encoding

At first, we colored each node by taking a ratio between each subsystems number of commits by the maximum number of commits. This heavily skewed the nodes to be green because the range of commits made to subsystems was huge. The maximum number of commits made to a subsystem was 500,000 while the minimum was 0. The average was somewhere around 1000 commits per subsystem. Almost all nodes took on a nearly indistinguishable shade of green, except for a select few nodes. To fix this, we implemented a logarithmic scale making it so that the color of each node is determined by the log of that previous ratio, except in cases where the ratio becomes

0 or 1. Those cases are handled individually, since $\log(0)$ is undefined and $\log(1)$ returns a blue color which is the opposite of what is desired when the ratio is 1. This logarithmic scaling fixed the issue with most nodes looking identical. Nodes now take on a range of colors from blue to red. The more red a node is, the more commits that have been made to it. The more blue a node is, the less commits a that have been made to it. This heat-map approach is useful because it makes it immediately obvious to the user which nodes (or subsystems) have been modified the most.

C. Graph Construction

We utilized C# in Unity to visualize the graph. First, we read the text file output from Java. The text file contained the parent-child relationships between all of the subsystems, the exact number of commits made to each subsystem, and a color corresponding to each node. Our code created a set of nodes and a set of vertices based on this text file. A node is created to represent each subsystem and its corresponding color, and an edge connects two nodes if and only if one is a child of the other. This creates a hierarchical graph layout that not only shows us the relationships between the subsystems, but also the relative commit frequency between them as well. This graph is then visualized using a radial layout so that the nodes are spaced evenly along different radii.

D. UI/UX

Our visualization implementation is interactive. In particular, the user is able to zoom in/out and pan the camera. The user is also able to click on each node to view information about that subsystem. On click, the user can see the name of the subsystem and the exact number of commits made to that subsystem.

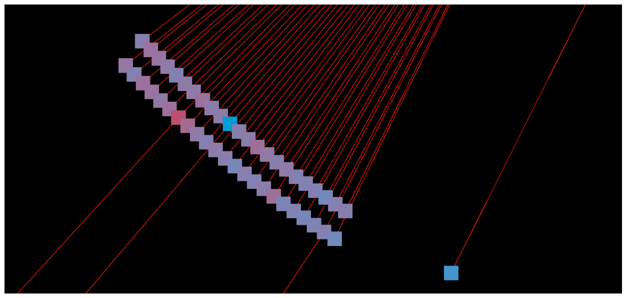

Fig. 1. This image shows a variety of colors attributed to nodes based on commit frequency of that subsystem.

IV. Results and Discussion

- talk about scalability issues, groups of high commit frequency, and think of other things to talk about as well.

- talk about future directions except you plan on doing some of those future directions so i don teven know what man. talk about using git clone instead of java text file reading output roundabout stupid way.

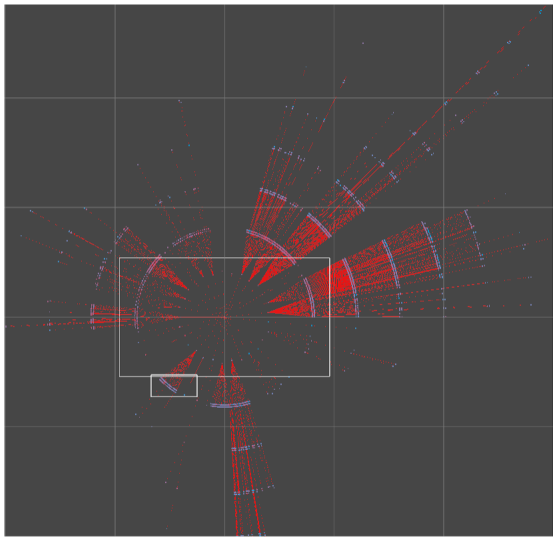

Fig. 2. wow this is image

V. Future Directions and Limitations

References

[1] R. Love, *Linux kernel development*. Pearson Education, 2010.

[2] A. Israeli and D. G. Feitelson, "The linux kernel as a case study in software evolution," *Journal of Systems and Software*, vol. 83, no. 3, pp. 485–501, 2010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121209002519

[3] K. Perumalla, A. Soni, R. Dey, and S. Rich, "Zeroin: Characterizing the data distributions of commits in software repositories," 2022. [Online]. Available: https://arxiv.org/abs/2204.07863

[4] M. Maes Bermejo, J. M. Gonzalez-Barahona, M. Gallego, and G. Robles, "A dataset of linux kernel commits," 2024, doi: 10.5281/zenodo.10654193.