Leveraging Linux kernel tracing to classify and detail application bottlenecks

This paper uses Linux kernel tracing to improve the features of GAPP, a tracing tool that is used to identify lines of source code that cause reduced parallelism bottlenecks. Most relevant to our study, the authors add a feature that allows GAPP to identify what type of bottleneck the application is experiencing. There are three categories that a bottleneck can be classified into: synchronization, I/O, and unknown. The authors begin with an extensive summary of the background information needed for this topic, including tracing, task-based parallelism, recent profiling tools, and synchronization. Next, they describe the extension of the GAPP tool. The classification of different types of bottlenecks, as mentioned before, is the most relevant to our study, and the authors explain how they achieved this function and some challenges they faced while attempting to get it to work. Finally, the authors detail their approach in testing the system. Testing was done using the Bodytrack tool within the Parsec benchmarking application, as well as Cuberite, a fan-made Minecraft game server. Through this testing, the authors were able to confirm that their extension worked, but also identified a few key issues. The study is concluded with some possible avenues for continued research being identified.

This paper is very extensive, which is one of its best features. The authors go into detail when discussing how they were able to identify the different types of bottlenecks, as well as issues they encountered. For our study, we could define these bottlenecks as several new categories of bad smells: synchronization and I/O. Additionally, as it was tested on a Minecraft game server, this project has shown that using kernel tracing to identify issues within the application's runtime is effective in real world applications. It is overall a very effective application that adds incredibly useful features to the GAPP tool.

One of the problems with relating this paper with our study is that this paper did not go into great detail when classifying the different types of bottlenecks. The authors identified I/O and synchronization bottlenecks. These are quite broad, and seem more like categories of tracing-based bad smells than bad smells themselves. A system to distinguish between threads that caused a critical stack trace by sleeping and those that caused a critical stack trace by being de-scheduled while waking a sleeping thread or switching to another thread right after waking it was implemented, but was not used to report different types of synchronization bad smells. One issue with the application discussed in the paper is that its upper bound for errors was around 15%, a relatively significant number. This upper bound was only approached while evaluating real world applications. Overall, this paper is related to our study and may prove useful as an example of an effective use of kernel tracing to identify and categorize issues; however, it only provides two broad categories of bad smells and does not go into further detail.

https://www.imperial.ac.uk/media/imperial-college/faculty-of-engineering/computing/public/1819-ug-projects/Davies-LyonsA-IO-bottleneck-detection-in-the-Linux-kernel.pdf