Managing User Impatience

Nick Dell’Orco, Yilun Liang, Keith Stebler

It is important to reiterate the purpose of the User Impatience governor; to provide the performance that the user wants, when the user wants it. Why is it important to reiterate? The reason is that we have to answer two questions. One, are we achieving that goal? Two, is that goal actually worth achieving? However, how do you answer those questions? There are various users with varying levels of patience. There are users that are willing to wait a millennium, users that wanted the task done before it was even started, and all sorts in-between. How well does the governor achieve its task for varying users? Furthermore, how does its functionality compare to preexisting governors? If the User Impatience governor does provide catered performance, but saps the battery life more so than, for example, the interactive governor, then we need to reevaluate the functionality of the User Impatience governor. However, how do you address these concerns? You cannot just provide one test case for using the governor, you have to have many different test cases of different users. You cannot just test the functionality of the governor, you have to stress the functionality of the governor and be able to observe the impact. This is the purpose of the testing system.

Before detailing the specifics of the testing system, it is important to note that we are not, at present, working on reading power consumption, so those components of the system will be omitted. As for the rest, you have two hardware components; the mobile device and the computer. On the device, you have the governor, the application that allows you to set and configure the governor that you want to run, and the various applications to be tested on. On the computer, there are three main software components; governor control, the scripts and the impatience system. In the case of the governor control, you input into the command line the configuration that you want to set the device's governor to. The software then sends this data to the device as a series of ADB commands. In the case of the scripts, the computer sends a script of ADB commands, such as tap and swipe, to simulate a user utilizing an application on the device. At present, there are five applications being used; Youtube, Gmail, Chrome, Spotify, and Photos. In the case of the impatience evaluation system, while the scripts are running, the user is being evaluated for the potential for impatience for each task within a script. If the user is found to be impatient, the script is paused, a series of commands are sent to the phone to indicate impatience, and then the script is resumed. This last software component is what we worked on improving.

Previously, the user impatience evaluation system was a strictly probability-based system. For each task within the script, 'roll a dice' to determine impatience. However, there were problems with this system. The results of evaluation did not reflect user intention, it acted independent of the applications being tested on, and it resulted in the CPU frequency maximizing or minimizing. We were tasked with coming up with a way to fix these problems.

The new user impatience evaluation system is a system that is based on evaluating the time of a task and comparing that to an amount of time that the user is willing to wait. However, there are problems with just having a direct comparison between task time and wait time. The first problem is that not all tasks are strictly dependent on CPU performance. For example, loading a website is also dependent on the speed of the network. The other problem is that, with this whole testing system, a significant amount of overhead has been added. For example, ADB commands need to be sent and processed by the device to do basic actions, like tap and swipe, that we take completely for granted. It is necessary to cancel out these problems in the evaluation process and to get a more accurate evaluation of the user.

To get a more accurate evaluation, an additional sub-component has been added to the evaluation process; the setup. The setup sets the governor to performance mode, maximizing the CPU. Then, it runs the scripts, without impatience evaluation. As it is doing so, it records the times of each task completion within a text file. By knowing what the maximum possible CPU performance is of the same task with the same overhead, you can eliminate those additional factors in the user impatience evaluation.

During the user impatience evaluation, for each task, the optimal time, taken from the setup, is subtracted from the currently running task time and that value is compared to the user's wait time. If the value is higher than the user's wait time, then the user, as to be expected, would be impatient. If the value is lower, then, obviously the user is happy and the governor can continue trying to reduce performance to determine where the user's patience level is actually at.

Furthermore, we have a model that will predict the user’s next complaint based on prior data to understand how we can minimize both average CPU frequency and the number of times a user complains. By doing this we have to find an efficient algorithm that optimizes the main variables, the amount to decrease CPU frequency, how long to remain at the current frequency and how much to increase frequency in the event of a complaint. One major problem has already been fixed as we already discussed as now the system does better to mimic a real user and eliminates the chances of an immediate complaint or of no complaints. As we are able to collect more data we will be able to dynamically adjust these settings to better serve the user. For example, currently the amount to drop CPU frequency is a fixed number so we will need to have ((max frequency - complaint frequency)/amount to decrease) number of drops at a fixed amount. Meanwhile, if we can predict where the next user complaint will take place we can optimize the initial drops to move faster and then slow as they approach the area where we expect a complaint. This process will drastically improve efficiency of the algorithm as long as the predictions are close to accurate. The second variable that we need to optimize is the amount of time we should remain at a certain CPU frequency before dropping it. Once again this is where our predicted complaint value is useful because as we are farther away from where we expect a user to complain, there is no reason to stay at that value for too long. Therefore, we are implementing a way for this value to increase as we approach the area where the user is expected to complain. Therefore, we don’t need to waste time keeping the CPU frequency in areas where the user likely won’t complain. The final variable to optimize is the amount to increase the CPU frequency in the event of a complaint. Currently, this value is also fixed. However, as we collect more data about the habits of a user and where they will likely complain we can gradually decrease this value after each complaint until a certain point (maybe 1-2 standard deviations away from the next expected complaint). This will eliminate the need to decrease too much after a complaint as we already know where the user is likely to complain next. Therefore, as complaints are logged the range that the CPU frequency stays within will become smaller and smaller unless the user’s trends change.

In regards to what still needs to be worked on, there are some problems that have come up, now that probability has been mostly eliminated. The first problem is specific to that, there is still a system in place that has a chance to ignore a complaint threshold set by a user, the purpose of it was to account for mistaken complaints. With the new system, this chance should either be reduced or completely eliminated as during testing there is often conflict between the user's intention and the governor. Another problem that has been revealed by the new system is the duration of time until the governor tries to drop the CPU frequency, if a task is taking a particularly long time to complete, the governor could have multiple drops in performance during this time span. However, for each task, there is only ever one opportunity that the user can issue a complaint. Furthermore, depending on how far the drop is, there is also additional time for dropping during the time it takes for the complaint to be issued. What all of this means is that the level that the complaint is issued at can potentially be much much lower than what the actual complaint should be at. So these are some problems that need to be evaluated going forward.