Pre-fetch Procrastinator for Android Apps

Team Members

1. Akshay Kamath
2. Amish Shah

Problem

Prefetching content is a methodology that developers use to download the content even before a user needs it. The primary benefit of prefetching content is that the app becomes more responsive. For instance, if there is a weather app and along with the homepage at the launch of the app, articles and images in other Views are also prefetched. So, the content is rendered quickly when the user wants to visit different Views inside the app. However, if the user usually works with the homepage and does not visit the other Activities/Views than the data consumed to prefetch content is wasted. To exacerbate things further, if a user has enrolled for a pay-per-byte plan, or he/she is near the end of billing cycle or currently under a roaming plan, the user may incur monetary loss due to pre-fetch.

Key challenges

From our initial plan, we believe that this work is not done in Android to the best of our knowledge and will be a first application of its kind. Hence, we face challenges that are specific to android. For example, we need to identify the code patterns that perform network fetch calls in Java code. This will be different from the existing work in [1] since they focus explicitly on Silverlight applications built for windows phone.

Another potential challenging area is the level of code analysis. Our current assumptions do not work at the byte code level. We plan to make changes at the source code level, hence the algorithm needs to be developed for the same. We need to check whether this is feasible and can be easily generalized to other android applications as well.

We also believe that measuring results for energy conservation is a challenging aspect as it would potentially require an energy model and would be difficult under the given time constraints.

Finally, we also need to ensure that the impact of procrastination on the default application does not have a negative performance impact. This could arise as a result of holding back potential data fetch that are critical for smooth functioning.

High-level Approach

We first plan to create an app and inject some prevalent prefetching techniques. Alongside, we plan to modify the algorithm mentioned in [2] that finds the correlation between off-screen background activities of an app and the user interacting with that app later during screen-on – BFC (Background to Foreground correlation) metric. For our work, in a similar fashion, we would like to capture the likelihood that the user will interact with a particular Activity in an app depending on the times the user interacted with that Activity during previous launches.

As per the algorithm, we would track the user movements in the activity until we calculate a confidence correlation (prefetch score) among activities. We will also maintain a UI- call map for a View/Activity that will contain all the UI elements with their respective pre-fetch network calls.

Once we calculate a prefetch score, when an app or a new view is launched, we traverse the respective UI-Call map and delay the calls for those activities that have prefetch scores lower than the threshold. When the delayed UI elements/Views are visited, we execute the network calls for those elements.

Expected Results

From our experiments and implementations, we hope to conserve network data that may be limited to a user, especially on cellular networks such as LTE.

Another expected result is that of battery improvement because of procrastinated batch requests instead of individual inconsistent request. This should ideally reduce the amount of state transitions in the cellular network and conserve battery. However, currently this feature will be good to have and we hope that we can achieve the same.

Road Map

Week 1 and 2: Create/Search an app and improvise algorithm

Week 3: Research patterns and maintain the global elements for restoration

Week 4: Design an algorithm

Week 5: Based upon the prefetch policy retrieve if UI elements appear frequently.

Week 6: Improve the scalability to include more elements for prefetch tasks

Week 7 and 8: Test mapping to real world applications.

Week 9: Evaluation various threshold values

Week 10: User feedback, report and analysis

References:

[1] Lenin Ravindranath, Sharad Agarwal, Jitendra Padhye, Chris Riederer. "Procrastinator: Pacing Mobile Apps’ Usage of the Network"

[2] Chen, Xiaomeng, Abhilash Jindal, Ning, Ding, Y. Charlie Hu, Rath Vannithamby and Maruti Gupta. "Smartphone Background Activities in the Wild: Origin, Energy Drain, and Optimization."