Design Doc: Multi Session Restore

Authors: bekell@microsoft.com
October 2020

One-page overview

Summary

User dissatisfaction with the useability of session restore only retaining the last session has resulted in requests for a multi-session storage approach. This change allows for Session & Tab Restore features to have an independent variable amount of session files stored.

Platforms

Mac, Windows, Linux, Chrome OS.

Team

bekell@microsoft.com

Bug

130066 - "Recently closed tabs" are forgotten after Chrome restarts a second time - chromium

Code affected

- Session Restore
- Tab Restore

Design

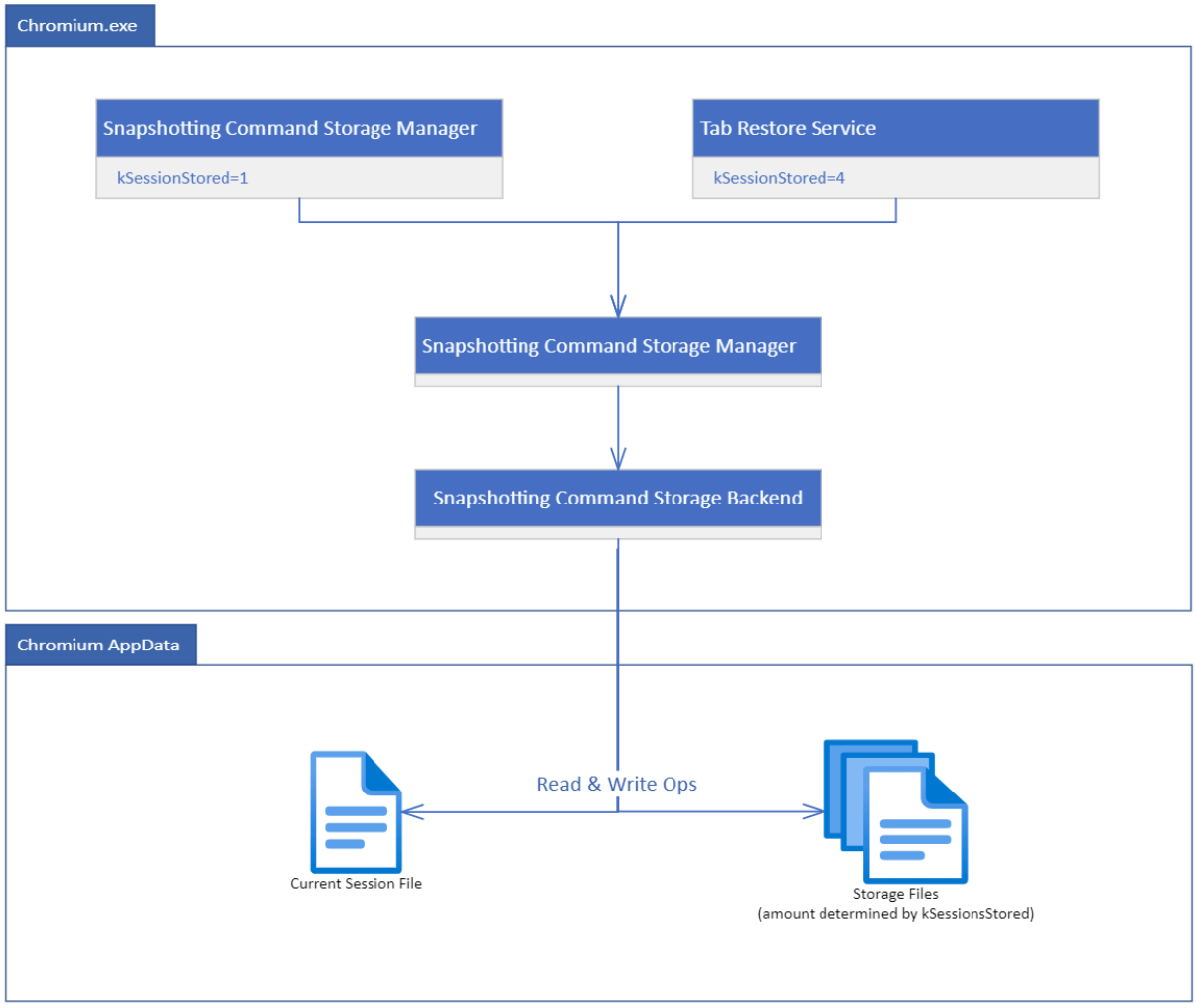
Currently, the Snapshotting Command Storage Manager has the idea of a “current session” and a “last session”. When the browser starts, a new current session is created and the previous current session becomes the last session. Any other files, such as the previous last session, are deleted from disk and the data is lost. While this makes sense in some UX’s, users have expressed a desire to get information from more than just the last session.

The proposed change is to move from retaining a “current session” & “last session” to a “current session” & “stored sessions” architecture, where “stored sessions” is a variable amount of session files. Storage will act as a FIFO queue, with the amount of sessions retained in storage determined by session and tab restore services. Operations remain largely the same, with notable changes listed below:

- When the browser starts, the previous current session will move into “storage” instead of becoming the “last session”. At this time, the storage files will be pruned (deleted) to ensure that the correct amount of files remain. These pruned files are thought of as expired, since they are the oldest session files. Regular behaviour should only be pruning one file each time, as the incoming current session expires the oldest session file in storage.
- Reading commands from storage reads all commands from all files within storage. The owning service can determine how many commands should be kept in memory (currently tab restore retains 25 commands).

Session Restore will be unchanged to only retain one session file in storage, and tab restore will keep four session files in storage. The UX for each restore service differs, where session restore is used to restore the most recent session, and tab restore can be useful to restore individual tabs & windows from multiple previous sessions.

A simplified view of the new architecture (only relevant components shown):



Metrics

Success metrics

This change has been driven by customer feedback, and we will therefore rely on further feedback to determine success of this work. Specifically, we will monitor user reaction to this change in the associated bug, as well as customer feedback via the various feedback platforms.

Regression metrics

Speed launch metrics will be monitored, specifically session restore metrics. SessionRestore.ForegroundTabFirstPaint3, which aims to measure roughly when the visible

tab is ready for user consumption. Additionally, user feedback will be closely monitored for session data loss impact.

Experiments

There is a potential avenue in which we launch experiments for the amount of sessions retained in storage for tab restore. This would allow us to determine differences in performance & usability of the feature, however this hasn't been thoroughly fleshed out and we will likely proceed with a waterfall rollout.

Rollout plan

Waterfall

Core principle considerations

Speed

This work has the potential to impact performance of the read operations of the stored files. During session or tab restoration, commands are read from the session files and used to restore the tabs or windows. This change increases the amount of session files for tab restore, so there is a potential impact on increasing read times. While we should monitor this (via regression metrics mentioned about), it is not overly concerning as the impact should be negligible because of the small increase in the amount of files (1 to 4). The highest impact would be on users who have very large session files, however this is already the case with the current implementation.

Furthermore, there will be a slight impact to disk usage, since we are storing more session files. This should be negligible since session files are relatively small (KB to a few MB) and disk space is not at a large premium. There is no memory impact, as the service determines how many commands should be kept in memory.

Security

No changes to the fundamentals of the read & write operations.
Command_Storage_Backend, which handles the I/O for session files, remains untouched.

Privacy considerations

We are storing tab session data for a longer amount of time, however, the session files will still be user-removable by clearing browsing history.

Testing plan

SnapshottingCommandStorageBackend unit tests have been modified to adapt to this change. A new test has been added to ensure reading commands reads from multiple session files.

Followup work

Monitoring user feedback & associated bug.