Next Generation Local Storage Presentation

(authored by Jan Varga)

Short description of LocalStorage

- LS is a simple web API for storing data locally
- It’s basically a key/value store
- There are two basic methods: getItem and setItem
- Everything happens synchronously, for example if you call getItem, you get the result immediately, you don’t have to pass any callback or use a promise
- This is very convenient for web developers, but unfortunately it has been pain in the ass for browser developers for long time
- We can’t get rid of this API since it’s used a lot, we just have to live with it (KV Storage may change this)

The old implementation

- Great at the time it was written (to my knowledge fastest among most popular web browsers)
- IndexedDB wasn’t used much, there was no DOM Cache at that time
- Usage reporting and permission control was spread across FF UI
- The old implementation was designed and optimized for non-e10s and e10s, in other words for single process case and multi process case where we had one main process and one content process (we couldn’t have more content processes, that came later)
- So LS implementation reflected Gecko at that time
- Bug for old impl 600307 (landed 7 years ago)
- Written by Necko team members from Czech republic :)

Gecko changed

- We now have multiple content processes (also for the same origin, but this is going to change with Fission probably)
- User interface also changed, especially preferences which now has a centralized place for storage usage reporting and site management (By site management I mean that you see all the sites that stored some data, you see how much data is stored for given origin, you can see when the origin was accessed last time and you can delete data for selected origins)

Other long standing problems in old LS implementation

- Not unified with other storage APIs (IndexedDB, DOM cache)
  - Quota
  - Usage reporting
- Data stored in one big database
  - Prone to database corruption
  - Disk space is not freed when an origin is deleted
- Data never evicted
- …

- ...
- Inefficient e10-multi support
  - Bigger memory footprint, each process must contain a cache for given origin
  - Problems with synchronization (must be synchronized even when other processes don’t need data, but this was actually mitigated in the old implementation in the end)
  - New private windows can’t get existing data from other private windows
- Data must be preloaded in a content process to be able to receive storage event notifications
- Different code paths for e10s and non-e10s
- 5MB limit for entire eTLD+1 domain
- ...

- ...
- Data clearing via observer service
- Preloaded data exists even if it is not used
- All data must be sent across IPC, even when just 1 byte is needed
- Data is not compressed (pages very often store stuff that can be efficiently compressed)
- Preloading triggered later during page load (could be triggered much sooner in the parent process)
- navigator.storage.estimate() doesn’t include LS (required for compliance with the Storage spec)

Analysis

When we analyzed these problems we came to a conclusion, that most of them can be fixed by moving the cache (the cache is a memory object which contains all LS data for given origin) to parent process and by plugging LS to quota manager.

However, it would be really hard to do it in the old implementation, so we decided to rewrite it from scratch.

The new implementation was split into several patches to make it easier to review it and also to track its features and decisions made during development. We ended up with 54 patches :)

Here comes new implementation

- We actually had to wait for several architecture changes that were needed elsewhere as well
    - PBackground (in non-e10s case, it would be impossible to use QM, when the main thread is blocked)
        - originally required for new IndexedDB implementation
    - Getting PBackground child actors synchronously
        - required for moving old LS implementation to PBackground)
    - Nested event queues
        - originally required for rationalizing event queues (the times of frankenventloop :)
- …

- ...
- As I mentioned most of the problems are solved by moving the cache to main process, using sync IPC calls to access it and using quota manager for unification with other storage APIs
- Quota checks and usage reporting is shared with IDB and Cache)
- We have per principal/origin databases
- Data is evicted along with IDB and Cache
- Data is preloaded only in main process, it means:
    - We don’t need to synchronize caches across content processes
    - LS in private browsing works correctly even with multiple content processes
- There’s no 5MB for entire eTLD+1 group (wikipedia happier, filldisk less :)
- ...

- …
- We have unified origin clearing
- Preloaded data is released/freed if page doesn’t use it in 20s after page has been loaded
- We send data through IPC only when it’s needed
- Data preloading is triggered much sooner
- Size of LS data is included in navigator.storage.estimate()

Challenges that we faced ...

How we plugged a synchronous API to asynchronous quota manager

Two ways for sync blocking:

- If we always use IPC then we can just use standard sync IPC calls
    - This works well only if you already have an answer for given request in the parent process, so we only use it when we already initialized quota manager and when we have data preloaded
    - When we don’t have data preloaded we can’t use it because we would have to spin the event loop in the parent process on the PBackground thread or just block PBackground until everything is prepared and preloaded, it’s usually a bad idea to spin the event loop or block the thread, especially in the main process and PBackground which is used by other subsystems

- So we decided to use a special technique for data preparation. We communicate with the parent asynchronously using a special thread in content process (DOM File thread) and while this happens we synchronously block the main thread. Originally we just used a monitor for the blocking, but then we had to change it to a nested event queue.
- During testing, we found out that some tests still use unsafe CPOWs (this was fixed by cancelling the blocking of the main thread when we detected other sync IPC messages, but in the end we had to get rid of the main thread completely)

How we optimized it for e10s-multi

- Preloaded data lives in main process
    - No synchronization needed anymore

How we lowered down number of synchronous IPC calls

Snapshotting:

- Much less IPC calls
- Also provides consistent view of the database during an implicit microtask
- If the database is small (under 16 KB) all data is sent to content which provides very fast access to data and greatly reduces number of IPC messages
- Snapshot are reused if data didn’t change in parent
- Explicit snapshots for testing (not available to web content)
- Preloading other data (up to 4KB) with each sync request to load a new item

CPOWs and LS

- Sync messages from parent process to content process while we synchronously block main thread in content process, may lead to a dead lock

How we handled migration of old data

- Old database webappstore.sqlite can be big, 150 MB (or even 500 MB) with thousands of unique origins (for example 3000 origins in my profile)
- Would be very slow to convert the old database into per origin databases
- Solution: lazy data migration on demand
- An archive database is created for migration
- SQLite supports transactions across databases

How to make the switch

- LS is used a lot on the web, if there’s a hidden bug, especially in the sync blocking, users could observe unpleasant hangs
- Solution: keep both implementations, have a pref for switching and keep old database in sync when the new implementation is enabled

Enhancements done after initial landing

- QM and quota clients (including LSNG) don’t use the main thread anymore
  - This was required to prevent dead locks caused by LS and accessibility doing sync blocking at the same time
- Data stored in UTF8 (stored in UTF16 before)
- Compression of data
- Lazy sqlite database creation (internal affairs)
- Using the write optimizer in content too (this significantly reduced SetLength OOM crashes and also max IPC message size crashes)
- Quota caching (1:10/1:6 improvement of storage initialization time)
- Reduction of shutdown hangs and storage init fails
- Various performance and correctness improvements

LSNG will ship once we address QuotaManager initialization failures (bug 1482662)