Collabora Productivity

Shrink and Load: Optimizing for speed and footprint

By Ashod Nakashian

Consultant Software Engineer at Collabora Productivity

@collabora.com

Overview

- Problem
- Opportunities
- Solution
- Results

The Problem

- Primarily in Online, but bonus points for Desktop improvements
- Slow loading == poor user experience
  - Perception matters!
  - Show something ASAP, continue in background / on demand
  - Users count time in spinning-wheel cycles
- Memory pressure real issue for scalability
- Very large documents might breach memory quota
- More CPU + memory → $$$

Approach and Tools

- Profile before coding
  - Three rules of optimization: Measure, Measure, Measure
- Know your target
  - Optimizing Gtk helps not Online performance (nor other platforms)
  - Common code optimization often win-win (but avoid regressions elsewhere)
- Valgrind: the tool of choice
  - Callgrind for CPU profiling
  - Massif for Memory footprint
- Test suite: a motley of documents
  - Use representative documents
  - 80/20 rules: 80% of resources used by 20% of code/cases
- See “Profiling with Callgrind” by Luboš Luňák on today at 16:30 (same room)

Themes

Loading

- Responsive UI on load critical all-around
  - But especially Writer load-time is crucial
- N.B.: Less memory ~= less CPU → Faster performance

Memory

- Optimize for peak-memory first
  - Reduces probability of swapping or exceeding memory quota
  - Often reduces steady-state footprint
- Caching can be costly
  - Support cache eviction and rebuilding on-demand
  - Double-edged sword: Hurts performance when overdone
- Reduce large objects → major gains


Loading / CPU
Optimizations

Initialize once

- Online forks from a pre-initialized instance
- Move all initialization to pre-forking stage
- Pre-load:
  - Spell-checker(s)
  - Locale data
  - Default fonts and scripts

Priority Inversion?

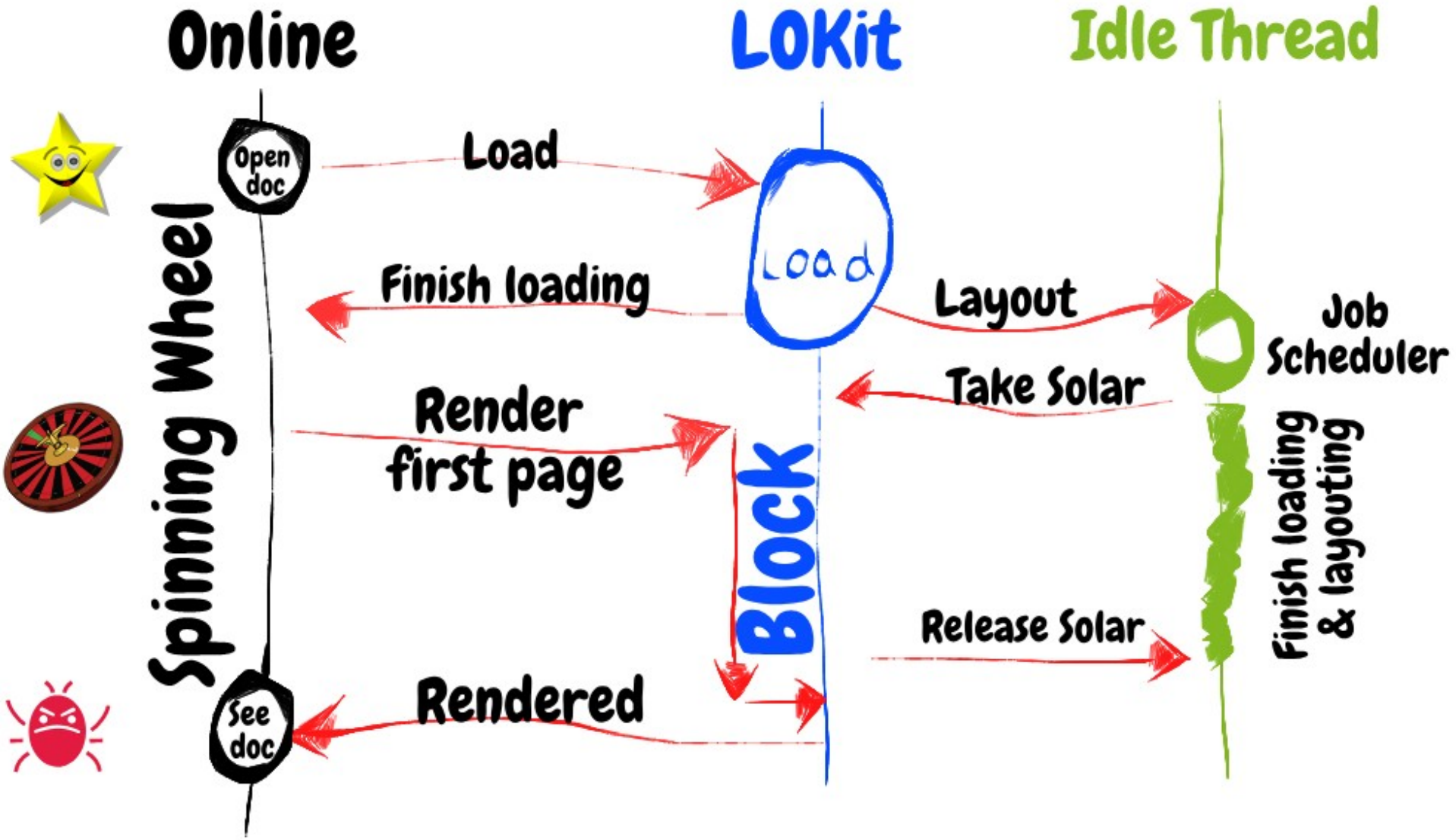
Symptom

- Document loads faster on desktop than in Online
- Online ~200% slower at loading documents?
  - A large document loads in ~17 seconds on desktop, but “35 wheel spins” in Online (user report)

Document actually loaded but not shown

- Idle jobs can hog CPU and monopolize SolarMutex
  - Textbook priority-inversion
- Remember: she who holds SolarMutex is queen

Priority Inversion Visualized



Credit: sketch.io

Priority Inversion Fix

Possible solutions

- Load and show first page immediately
- Prefer rendering request over loading and layouting the remainder of document
  - No prioritization between Idle jobs and LOKit API
  - No mechanism to request the release of SolarMutex

Solution

- Conditionally delaying idle-jobs until LOKit load-and-initialize logic is done and first page is rendered

  → Document loaded and visible in half the time
- Differed job is invoked on-demand → no delay due to this fix

GTK improvements

Improved desktop experience

- Avoid unnecessary work by caching states
  - Font styles are cached, but broken
  - Easy fix: remember last state, check before updating

- Excessive refresh requests costs many CPU cycles
  - Manage GUI notifications
  - Render when ready, not when told


Memory
Optimizations

Data-structures matter

Overview and Improvements

- Avoid storing copies of objects in multiple data-structures for performance benefits
  - Use-case: PDF GraphicsContext-ID maps
  - Instead of maintaining two maps for bi-directional lookup, use a single boost::bimap
- Avoid unnecessary indirection
  - A vector of (small) objects better than pointers to objects
- Reorder members to avoid/minimize padding → smaller footprint per instance

Results

- One PDF dropped the GraphicsContext maps' overhead from 160MB to 110MB
  - A reduction of >30%
- Faster lookups due to cache locality

Render PDF directly to raster

Overview and Improvements

- Importing PDF elements can be very slow for complex PDFs
- For view-only (in Online) individual elements are useless
  - And they reduce the accuracy of the positions
  - “PDFium for better PDF rendering and editing” talk by same author
- Potentially avoid creating large number of PDF elements saves memory

Results

- Memory reduction for complex PDFs
- Significantly faster and more accurate PDF viewing
- But works only for view-only
- Uses fixed-memory for images, can consume > memory for trivial PDFs

Cairo/Pixman caching

Overview and Improvements

- Pixmap glyphs are cached
- Cache size is limited by number of glyphs
  - Glyphs vary widely in size
- Track cache size by total memory footprint
  - Evict entries when exceeding memory footprint limit (currently 4MB)
- Also, evict cache more aggressively when many entries are stale

Results

- Significant reduction in steady-state memory footprint
  - Slight improvement / no decrease in run-time performance

Compact RGB images (Major win)

Overview and Improvements

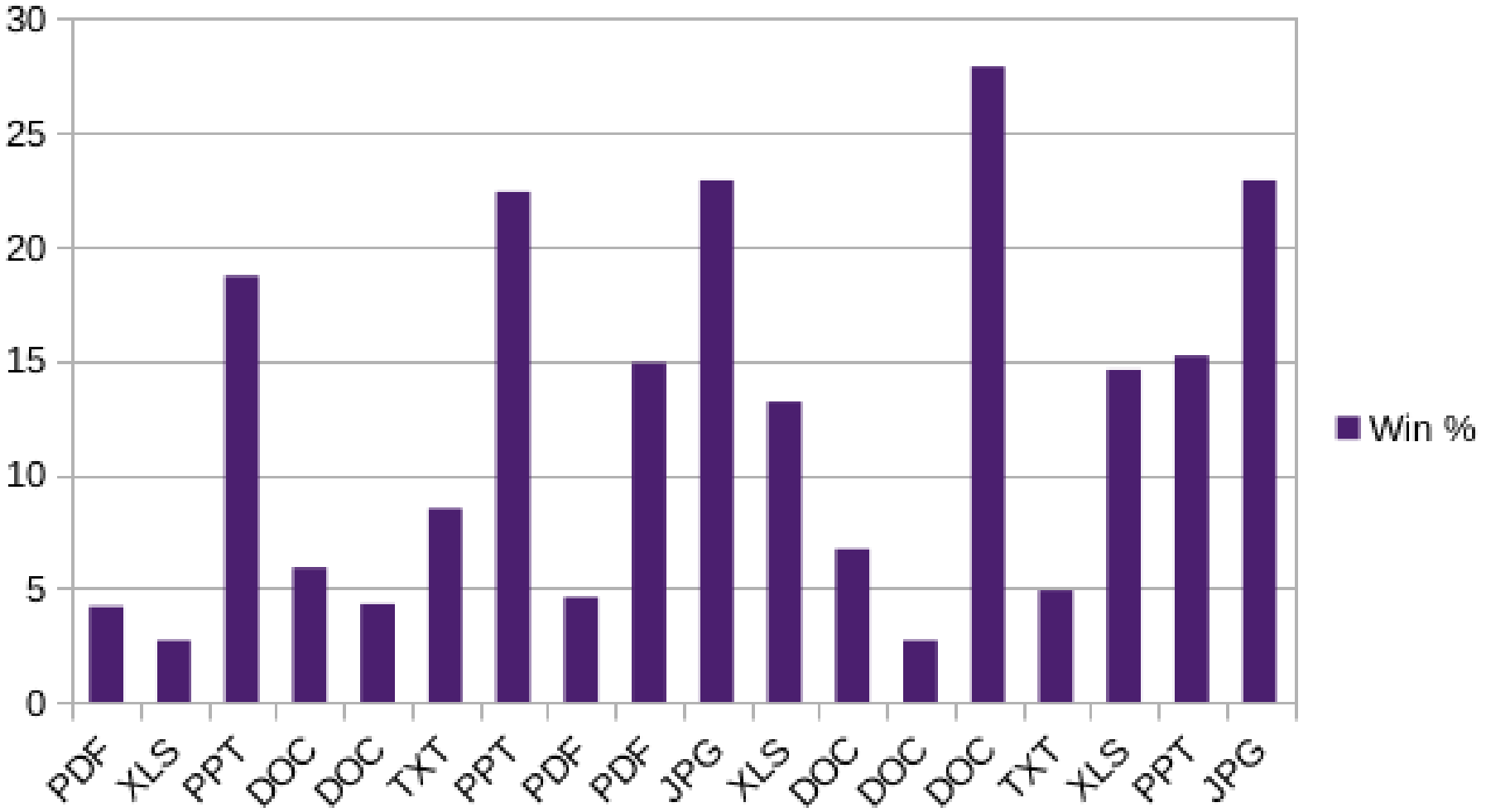
- Headless stored 24-bit RGB images in 32-bits
  - 33% memory overhead per image
  - By far largest memory hog for image-rich documents
- Introduce new SVP_24BIT_FORMAT for 24-bit RGB images
- Support new CAIRO_FORMAT_RGB24_888 format in Cairo
  - Avoids converting 24-bit RGB to Cairo-native 32-bit ARGB

Results

- Significant memory reduction for image-rich documents

Results

Overall win: 12.4%



<Your Question Here/>

By Ashod Nakashian

Ashod.nakashian@collabora.com

Collabora Productivity

Thank you!

By Ashod Nakashian

Ashod.nakashian@collabora.com