COOL performance
making collaboration slick & quick.

By Michael Meeks

General Manager

@michael_meeks michael.meeks@collabora.com

“Stand at the crossroads and look; ask for the ancient paths, ask where the good way is, and walk in it, and you will find rest for your souls...” - Jeremiah 6:16
Outline

Basics of how COOL works

LibreOffice core Technology
  - Wiggly lines

COOLWSD / Kit
  - I/O and queueing

Javascript:
  - Websocket
  - String / Image handling & async
  - DOM mutation
  - JQuery / Select2

Profiling, tools & future
How Collabora Online (COOL) works:

**Browser**
- Thin Javascript.
- Overlays for cursor / selection etc.
- Pan / zoom interpolation / shape overlays for fluid movement

**WSD**
- Web Services Daemon – multiplexes all messages to/from the Kit

**Kit**
- A securely contained & isolated LibreOffice
- Streams ‘tiles’ to the client as PNG images
  - has view of whole document: unusually zoomed out.
  - Has multiple views – one per user.

**User**
- cognitive biases & perceptual fun.
LibreOffice core Tech.
Performance Testing & typing ...

- Customer feedback: “we tested it with eight people doing random typing”
- Profiled this use-case; it is/was slow
  - The mis-spelling squiggly-line (cf. wrong language setting?) ...
    - an unfeasible amount of CPU ~90% of rendering time
    - A most beautiful sub-divided, AA b-spline but … ~2 pixels high mostly.
    - Fixed in 6.4.10
- Mashing the keyboard a pathological case: we’re still working on improving.
- Sdf sadf kjh lksdhfk ashdflkjashdlkfh slkdfhkasdh flksjdh f;ksah dflk kweyr iuh ks,dnf;yi o;wae ,n sadlkjfh
JSON generation

Lots of events generate JSON

- Particularly side-bar & dialog – description of widgets:
  - Lots of JSON: DumpAsPropertyTree
- Switch from:
  - `boost::property_tree::ptree` DumpAsPropertyTree()
  + `void DumpAsPropertyTree(tool::JsonWriter& rJsonWriter)`
- Instead of deep duplicating & returning ptree's
- Implement a new JsonWriter
  - Ultimately a stream type interface anyway.
  - Disappears from the profile.
- Thanks to Noel Grandin
Image scaling & rendering
Continual re-scaling of bitmaps

We had a nice image scaling cache:

- Problem: only caches one size per image
- For (random) reasons: not working nicely on Android.
- Now we have a multi-resolution scaled image cache:
  - Hugely faster, particularly for large zoom-out

Online

- Now we scale the cache size based on the number of open views
- Great for multiple users at different zooms
- Thanks to Lubos Lunak
Pointless ~O(n^3) in SwRegionRects

SwRegionRects::Compress()

- Notionally saves effort & space by compressing invalidated rectangles together.
- Particularly problematic with COOL – since the document is always visible in a gigantic pseudo-view.

Now only ~O(n^2) in number of regions

- https://gerrit.libreoffice.org/c/core/+/122121

Thanks to Lubos Lunak

Should accelerate all large writer documents with complex invalidations.
Calc: ScDocument::GetPrintArea

Called surprisingly often

- Switching views, when re-rendering a region etc.

Pixel area dependent on zoom

- Row heights vary in real height based on zoom level
  - But all look the same height.

So – scan from the beginning ...

Cost is all in:

- ScTable::GetRowForHeight(sal_uLong nHeight)

Now massively faster

- Walks both ‘hidden’ and ‘height’ span-trees concurrently – in jumps.
- Instead of iterating row by row.
And much more in core ...

Noel Grandin’s work

• Endless profiling & improvement:

Lots of misc other pieces

• Faster file opening
• Better font caching to accelerate text rendering
• Quicker scrolling
• Quicker spreadsheet filtering
• Faster large chart insertion/setup

Don’t paint to windows

• In LOK mode we used to often calculate & paint to an invisible 1x1 pixel window
• Avoid repeated writer layout calls too.

Detail overload ...
Web Service Daemon / Kit
Shuffling vectors ...

Buffering outgoing socket data: std::vector<char>

- Transmit from the beginning and then erase(begin(), begin() + sentBytes)
- Unfortunately: SSL: 16k max writable chunks
- 20Mb images / document downloads common
- Shuffling ~10Mb average - 1200x times down a vector – not fast.

Buffer class

- Wrap a std::vector<char>
- Don’t erase – have an offset: send 1Mb at a time before shuffling
  - bingo – 64x faster.
STL / Android amazement

STL on Android is abysmal

- Thankfully we no longer have to binary-patch it at run-time; but ...

vector::~vector<char>

- Very high on the profile – doing some ‘0’ assignment in a loop while destroying ?
- allocation – understandably slow – but freeing [!] ...
- More time spent allocating, wiping & freeing std::vector<char>
- Than rendering document content: huh !
- calloc buffer to render into instead.
And here it is:

Android std::vector <char> folly:

Scaling bitmaps, rendering tiles etc.
Merge key-events

Under heavy-load

- Can’t process key-events in the time they come in:

Input event compression:

- Kill unnecessary keyup events, then:

  child-foo textinput id=0 text=f
  child-foo textinput id=0 text=o
  child-foo textinput id=0 text=o → Turn it into:
  child-foo textinput id=0 text=foo

- So we can catch-up ... (also for removetextcontext (backspace/delete) events)
- Thanks to Tor Lillqvist.
Asynchronous save ...

Previously

- Paused all document editing during save + up-load

Up-load

- Thought to be fast: data-center ↔ data-center internal network link & storage.
- But ... some backends: several seconds
- So re-worked to continue editing while we up-load.
- Thanks to Ashod Nakasian

Solves autosave ‘stalls’ while typing

Even so some things sync still:

- Rename for example
- So be pretty there:

Saving document, please wait...
Javascript
End to end profiling

Catching badness across the board

- Found that we had been optimizing the wrong piece.
- So implemented a new end-to-end profiler.

Core: ProfileZone

- Passing data back from Kit → WSD

JS: TraceEvent logging

- Passing data back from browser → WSD

WSD:

- ProfileZone code too.

To enable:

- Tripple-click in Help→About
- [x] Performance Tracing
- Needs: trace_event[@enable] config option in loolwsd.xml.

Visualize:

- Load trace in chrome://tracing

Thanks to Tor Lillqvist
Profiling: Javascript – the surprise

We thought JS in the browser is fast

- We obsessed about network latency & server-side performance.
  - We were mostly wrong.
  - (though lots of sillies on the server-side too ...)

Please be careful with your JS

- DOM mutation, Canvas re-rendering, ‘elegant’ code using unusual libraries.
Watch each tile render: (spreadsheet with red background)

Websocket messages processed one by one at idle ...
do a re-render → we see an animation of each tile rendering

<table>
<thead>
<tr>
<th>LONG TASKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Spreadsheet Image]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Timings</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Timeline Image]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Experience</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Experience Table]</td>
</tr>
</tbody>
</table>

Same problem with async image load from .src=<base64 URL>
Event emission:

_emitSlurpedEvents: function() {
    this._map._docLayer.pauseDrawing();

    try {
        for (var i = 0; i < queueLen; ++i) {
            var evt = this._slurpQueue[i];

            if (evt.isComplete()) {
                try {
                    // it is - are you ?
                    this._onMessage(evt);
                }
            }
        }
    }
}
Websocket → base64 imgURL

lots of gc pressure & hence time
lots of string copying, slow ...
Before code:

```
// read the tile data
var strBytes = '';
for (var i = 0; i < data.length; i++) {
    strBytes += String.fromCharCode(data[i]);
}
img = 'data:image/png;base64,' + window.btoa(strBytes);
```

After code:

```
// convert to string of bytes without blowing the stack if data is large.
_strFromUint8: function(data) {
    var i, chunk = 4096;
    var strBytes = '';
    for (i = 0; i < data.length; i += chunk) {
        strBytes += String.fromCharCode.apply(null, data.slice(i, i + chunk));
        strBytes += String.fromCharCode.apply(null, data.slice(i));
    }
    return strBytes;
},
...
img = 'data:image/png;base64,' + window.btoa(this._strFromUint8(data));
Invisibly repeating the same work.

Now we: delay all the cursor related `onScrollTo` work / etc. until we have processed our whole incoming queue
We were continually re-creating & destroying table handles for multiple redundant table selected messages:
15x faster do it just once.

```javascript
_updateTableMarkers: function() {
  if (this._currentTableData === undefined)
    return; // not writer, no table select
  if (this._currentTableMarkerJson === this._lastTableMarkerJson)
    return; // identical table setup
  this._lastTableMarkerJson = this._currentTableMarkerJson;

  avoid destroying & re-creating identical table handles
```

‘messagesdone’ to do it right easily:

New ‘messagesdone’ event

• fired when we have emitted all complete slurped messages
• If you’re updating view-state, re-render once at the end ...
JQuery plugin thrash:

Select2 → argh!

- That 31337 new JQuery plugin
- **800ms** on startup of thrash
- Saw this with jsdom → noticed it ... ~5s+ of CPU time

Thanks to Mert for fixing it

- Using native JS now
Calc: client side rendering ...

Spreadsheets

- Header / row column sizing
  - Replicate the rounding nightmare in the client to avoid sending it later

Render grid-lines on the client

- Instant <ctrl>-<down-arrow>
- Possible to do some cursor movement locally too in future.
- Potentially render ‘cell’ tiles.
Ongoing Work ...
Deltas ...

private/mmeeks/deltas
- Monotonic tile-id updates
- Diff tiles to previous versions & send a binary patch → Time compression.
  - Deflate too
  - Inflate in JS
  - big B/W reduction.

CanvasSections:
- dirtying – to avoid re-paint

Better JS usage
- Async loading of images:
  - Horribly slow
  - Can’t be controlled / sequenced by JS
- Seems better to unpack pixels & send to Canvas manually
  - (amazingly)

Cursor / tile delta synchronization

Work ongoing – not yet merged.
Other in-progress wins

Reduce protocol thrash

- Avoid redundant notifications:
  - eg. per key-stroke:

  statechanged: .uno:LanguageStatus=English (USA);en-US
  statechanged: .uno:InsertPageHeader={}  
  statechanged: .uno:InsertPageFooter={}  
  statechanged: .uno:Undo=enabled
  statechanged: .uno:Orientation=IsPortrait
  statechanged: .uno:TrackedChangeIndex=
  tabstoplistupdate: { "tabstops": ""}  

Each change:

- Forces a spin of the browser main-loop to read from the websocket.
- On a ‘busy’ browser – adds lots of latency.

Others happen too fast:

- statechanged: .uno:StateWordCount=3 words, 13 characters
Testing tools ... 

**Perf-test**

- `cd browser ; make perf-test`
- Built on sample customer writer odt
- Plenty of complex tables, layout, text
- Runs Javascript as-is
  - jsdom, jscanvas
- Six concurrent users
  - Jump to a bookmark
  - Do random typing

**Coolstress**

- `./coolstress wss://localhost:9980 test/data/hello-world.odt test/traces/writer-hello-shape.txt`
- Loads a document, and replays a trace
  - cf. test/traces for sample editing sessions
- Approximates responses of JS client
- Very scalable – easy to run 300 simulated clients at once & measure latency / metrics.
Conclusions: much faster

Much improved performance work for Collabora Online

- Lots of this in LibreOffice 7.2, more coming in 7.3
- Much of it shipping in COOL 6.4.11, more just arrived in COOL 21.11

More work to do here

- more stress testing & profiling is underway
- We’re not even half way done yet.
Collabora’s mission:
Make Open Source ROCK
Oh, that my words were recorded, that they were written on a scroll, that they were inscribed with an iron tool on lead, or engraved in rock for ever! I know that my Redeemer lives, and that in the end he will stand upon the earth. And though this body has been destroyed yet in my flesh I will see God, I myself will see him, with my own eyes - I and not another. How my heart yearns within me. - Job 19: 23-27