Collabora Productivity

Approaching the 1M columns / rows limit in Calc Online

By Marco Cecchetti

Collabora Productivity

marco.cecchetti@collabora.co.uk

Overview

Before: only 1000 rows was handled

- Fetching the whole row header
- Cursor position computation dependent on the row index: the higher row index → the longer to get cursor position

Now: we can handle 1 million rows

- Fetching visible header portion only
- Update row/col header independently
- Position caching for both cursor and top/last visible row header

Problems for handling 1M columns /rows

Header fetching

When we were dealing with at most 1000 rows

- Fetching of the whole row header occurred:
  - on document loading
  - on row operations (insertion, deletion, resizing)
  - No need on scrolling
- That was not really expensive for only 1K rows
- It had the advantage to require header updating not very often.

Performance in computing cursor position

Dependent on the row index

- Calc stores the height property for each row
- To get the distance of the N-th row wrt the top of the document it is needed to sum up all the row heights.
- The greater is N:
  - the longer takes to compute the cursor position
  - and the position of the first row visible by the user

The implemented solution

Header fetching

Fetched only the portion of visible header

- On the LOK core side, we compute the row positions for the requested range only
- On the LOK client side, we need to request a header update, each time some scroll occurs.
- One could think that this is expensive
  - Anyway messages btw core and LOK clients are asynchronous
  - Suppression of multiple header data messages when they are queued together: only the last one is sent

Position caching goals

Computing row distance from the document top independently by the row index

- Allow to compute cursor position faster whatever the current document part the user is working on
- A better UX for document navigation through arrows keys or PgUp/PgDn
- Allow to compute the first and the last visible row position faster even when the visible row range is near to the bottom of the document for a 1 million rows spreadsheet.
- Header updating occurs faster avoiding serious delay between the LOK client request and the LOK core reply.

Position caching implementation

A very simple data structure keeping tracks of some special positions.

- Per each tab view we have 2 instances of this data structure.
- One for rows, storing:
  - The latest cell cursor row index and its x coordinate in pixel
  - The latest top and bottom visible row index and their x coordinate in pixel
- One for columns, storing:
  - The latest cell cursor column index and its y coordinate in pixel
  - The latest leftmost and rightmost visible column index and their y coordinate in pixel

ScPositionHelper

```
class ScPositionHelper
{
public:
    typedef SCCOLROW index_type;
    typedef std::pair<index_type, long> value_type;
    static_assert(std::numeric_limits<index_type>::is_signed, "ScPositionCache: index type is not signed");

private:
    static const index_type null = std::numeric_limits<index_type>::min();

    class Comp
    {
    public:
        bool operator() (const value_type& rValue1, const value_type& rValue2) const;
    };

    std::set<value_type, Comp> mData;

public:
    ScPositionHelper();

    void insert(index_type nIndex, long nPos);
    void removeByIndex(index_type nIndex);
    void removeByPosition(long nPos);
    void invalidateByIndex(index_type nIndex);
    void invalidateByPosition(long nPos);
    const value_type& getNearestByIndex(index_type nIndex) const;
    const value_type& getNearestByPosition(long nPos) const;
    long getPosition(index_type nIndex) const;
    index_type getIndex(long nPos) const;
};
```

Inserting/removing a position into the cache

- A new (row index, x coordinate) pair is inserted into the row positions cache:
  - Soon after current cell cursor position has been computed due to a request from a LOK client
  - Soon after a new start/end row position for the visible part of the row header has been computed due to an update request from a LOK client
- In both cases, before inserting a new row position, the old one is removed.

Retrieving a cached position by index

- It is possible to look for the nearest cached row position to a given row index:

value_type& getNearestByIndex(index_type nIndex)

- That occurs when you want to compute the x coordinate for a new cell cursor position:
  - You know the new row index of the cell cursor
  - You get back the row index and x coordinate in pixel of the cached row position whose index is the nearest to the passed one

Retrieving a cached position by x coordinate

- It is possible to look for the nearest cached row position to a given row x coordinate:

value_type& getNearestByPosition(long nPos)

- That occurs when you want to compute the index and coordinate of the first row intersecting the visible area of the document for a given view
  - You know a x coordinate which is the the top of the visible area wrt the top of the document
  - You get back the row index and x coordinate in pixel of the cached row position whose x coordinate is the nearest to the passed one

Make the solution safe

Invalidating cached position

How it is achieved

- By index:

void invalidateByIndex(index_type nIndex)

  - Any cached row position whose index is greater than nIndex is deleted from the cache

- By position:

void invalidateByPosition(long nPos)

  - Any cached row position whose x coordinate is greater than nPos is deleted from the cache

Invalidating cached position

When is needed:

- On row operations (insertion, deletion, resizing, ...)
- When the value of the PPTY parameter is updated (scaling)
- An undo/redo action related to a row operation occurs
- The same is needed for cached column positions

The solution takes care of both row/column operations performed in another view and the current tab each view is displaying.

Collabora Productivity

Thank you for listening!

By Marco Cecchetti

marco.cecchetti@collabora.co.uk