SWMR Performance Data

This document presents some initial performance data on the Single-Writer/Multiple-Readers feature.

Raw data and test parameters can be found in the accompanying spreadsheet. In the interest of brevity, they are not repeated in this preliminary document.

Methodology (Brief)

A small program was written that creates a new file containing a chunked dataset of 32-bit integers and then appends data to the dataset under several SWMR conditions:

- No SWMR flag set
- SWMR flag set, but no explicit flushing performed
- SWMR flag set, flushed frequently (every 100 elements)
- SWMR flag set, flushed rarely (every 10,000 elements)

Both 1D and 2D datasets were tested so that the performance of both chunk index structures (extensible array and version 2 B-tree, respectively) could be investigated. In the 1D case, a variety of chunk sizes were used. In the 2D case, only one chunk size was used, but appending in both directions and along the diagonal was investigated to see if there were differences in the performance.

The reported results are an average of several runs on a lightly loaded 32-bit linux machine using a local ext3 disk.

Results

1D Tests

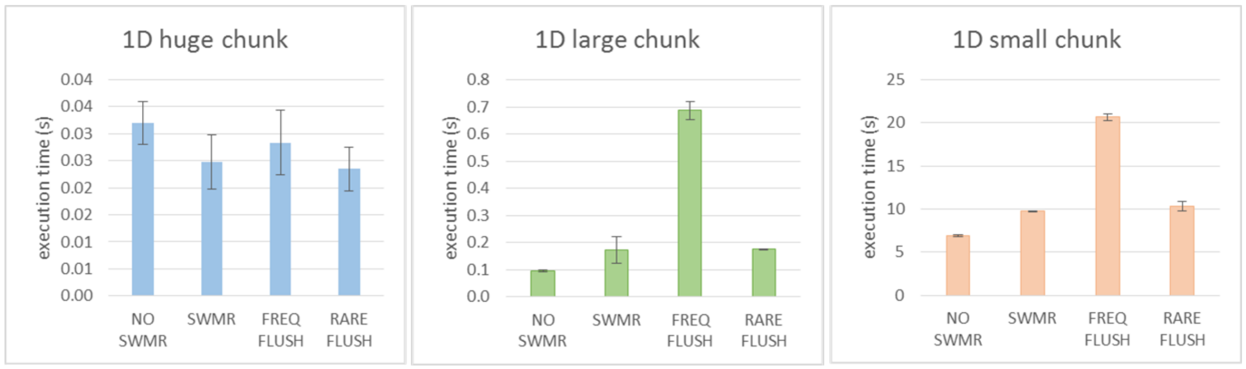

Figure 1 - 1D SWMR test results. The error bars show the standard error of the mean.

Table 1 - 1D execution times with respect to the NO SWMR test run. The yellow highlights show the odd results for huge data chunks. The red highlight shows the very poor performance of large chunks that are frequently flushed.

DATA	SWMR	FREQ FLUSH	RARE FLUSH
huge chunk (100 k elements)	-22%	-11%	-27%
large chunk (1000 elements)	+79%	+615%	+81%
small chunk (10 elements)	+40%	+197%	+48%

It's hard to know what to make of these data and we'll clearly have to investigate this further. There does appear to be a performance hit for using SWMR and this hit appears to grow with flush frequency, which is not surprising. Understanding the other data will take some time. It's unclear, for example, why setting the SWMR flag would provide better performance when the chunks are very large than having it off. One factor that likely has an effect is the nature of the extensible array. With a small number of chunks, only an index block will be necessary since, as an optimization, it can store a certain number of elements. This would reduce the flush dependencies to just the dependency between the header and the index block, so the cache performance penalty would be very low. It's especially unclear why the large chunk case is so pathological when frequently flushed. That use case should be a high priority when it comes time to optimize the feature.

2D Tests

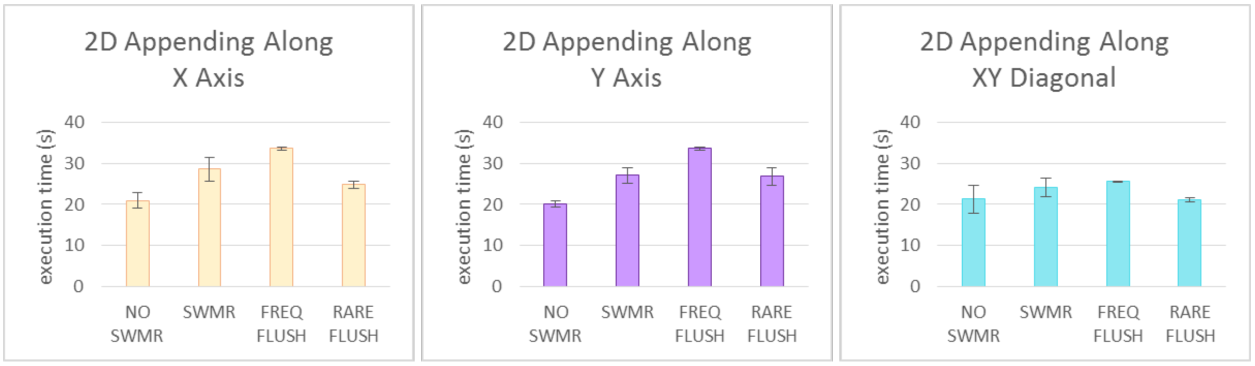

Figure 2 - 2D SWMR test results. The error bars show the standard error of the mean.

Table 2 - 2D execution times with respect to the NO SWMR test run.

DATA	SWMR	FREQ FLUSH	RARE FLUSH
Append in X direction	+37%	+61%	+19%
Append in Y direction	+35%	+67%	+33%
Append in XY direction	+13%	+20%	-1%

These data are much more consistent than the 1D data, probably due to the more uniform indexing scheme. Appending in all directions was measured to see if there were any glaring inefficiencies. As the data show, there is a measurable penalty for the SWMR feature and flushing frequently adds an additional penalty. The improved performance of the RARE FLUSH data with respect to the SWMR (no flush) data is more difficult to explain but may be due to an emptier cache making the SWMR traversals and checks less time consuming. This will require further investigation to explain.

Conclusions

The results presented here and in the accompanying spreadsheet are intended to give a first look at the performance impact of the SWMR feature on write operations. Given the relatively mature state of the metadata cache changes, flush dependency behavior, and chunk index structures, these data should provide an approximate upper bound on the SWMR write performance penalty. It should be noted, though, that the SWMR feature is still under development and has not been optimized at this time so these numbers are likely to come down in the future.