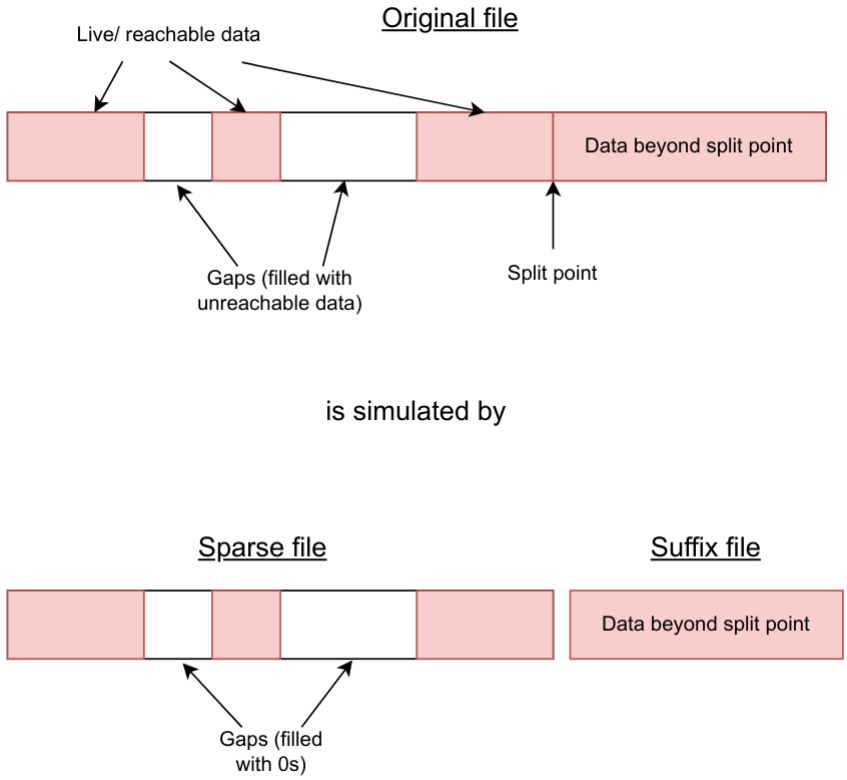
A "chunked file" implementation, and application to reduce on-disk overhead of Irmin layers/GC

Possible problem: The current design of Irmin layers/GC keeps disk space usage for the live data below 30GB, but each running GC requires on-disk extra space of 24GB (which is reclaimed by the filesystem when the GC finishes). If this 24GB is considered too much, it is possible with some extra engineering to reduce the overall on-disk extra space to 5 or 6GB.

[Aside: Numbers above and in what follows have been chosen to be illustrative of the approximate sizes.]

Background: The current design of Irmin layers/GC involves splitting an (append-only) "pack store" file into two parts: a "sparse file", consisting of the live parts of the file before the split point, and the "suffix file", consisting of everything after the split point.



When performing a GC, we construct a completely new sparse+suffix, and then switch the running implementation from the old versions (of the sparse+suffix) to the new versions. If the split point is relatively recent (towards the end of the current suffix file), then the on-disk overhead is approximately:

1. 3GB for the new sparse file (the live data before the split point)
2. 1GB for the temporary working files used to construct the new sparse file
3. (Very small) overhead for the new suffix file

However, if the split point is earlier in the file, then the third component -- the overhead for the new suffix file -- can be large. For example, in the Tezos usecase, the split point is "6 cycles ago", which corresponds to a split point 20GB from the end of the pack store file (which is 20GB from the end of the current suffix). In this case, the overhead is 24GB $(3+1+20)$ in total which is roughly the same order of disk space that is used to store the data itself. In practice, node owners will have a roughly constant disk usage of between 20GB and 30GB for the data of the last 6 or 7 cycles, and periodically a running GC will increase this by 24GB, which will then be reclaimed by the filesystem when GC finishes.

Thus, running GC effectively doubles the disk storage required for keeping the data of the last 6 or 7 cycles.

It is not clear if this is a problem or not. Even if it is not a problem in practice, it is worth noting that the overhead can be made smaller in a relatively simple way.

Suffix file: A "suffix file" represents the data in a file from a given "split point" onwards. If the split point is at position n in the file, then a suffix file contains all the bytes in the file from position n onwards.

The implementation of a suffix file just uses a normal file: attempting to read from position n actually reads from position 0; it is impossible to read before position n; reading at position $n + m$ corresponds to reading from position m in the underlying implementation file; appending to the file is achieved by simply appending to the underlying implementation file.

When we perform GC, we construct a new suffix file, using a new split point somewhere after the split point of the current suffix file; we copy all the data in the current suffix file, from the new split point to the end, in to the new suffix file. This is a simple file copy. During this operation we have two copies of the data from the new split point onwards (one in the current suffix file, one in the new suffix file). This is the overhead when creating the new suffix file. When we switch the running Tezos implementation to use the new sparse+suffix, we can delete the old sparse+suffix to reclaim the space.

```
Current suffix :  [abcdefg.....hijklmnop]
New split point:               ^
New suffix     :              [hijklmnop]

NOTE: the new suffix is constructed by copying the tail of the current suffix
```

Constructing the new suffix would be easy if we were allowed to just use the current suffix, but drop all the data before the new split point. Existing filesystems implement "truncate", which drops all the data in a file AFTER a given point (and reduces the size appropriately). However, what we want to do here is drop all the data in the file BEFORE a given point. Unfortunately, even though this could be easily supported by most modern filesystems, it is not implemented by any that I know of.

Chunked file: A chunked file is a way of simulating a file, which supports the operation "truncate before point". The idea is simple: given a particular "chunk size" (say, 10M), the file "xyz" is stored as a finite sequence of files "xyz.0", "xyz.1", "xyz.2", ... "xyz.n" each of size 10M (except possibly the last), where the index number indicates the index chunk. Thus, "xyz.21" holds the data for the 21st chunk of the file, from offset 21 * 10M upto 22 * 10M. To implement the operation "truncate before point", we simply delete all chunks before the chunk corresponding to the given point. For example, given the point 21 * 10M, we would delete "xyz.0" upto "xyz.20".

NOTE truncate-before deletes chunk files on the underlying filesystem, thus regaining on-disk space.

NOTE We can also remove chunks "in the middle of the file" as well, but we do not consider this further.

Note that the truncate-before operation is not precise: we can only drop complete chunks, so if we try to truncate-before in the middle of a chunk, we will have to keep all the bytes in that chunk, and only drop the preceding chunks. Thus, there is additional overhead related to the chunk size. However, we can't just reduce the chunk size freely: beyond a certain point the performance of our chunked file will begin to suffer, compared to a normal file.

NOTE truncate-before is expected to be fast, since it is simply unlinking some chunk files. Of course, if there are many chunk files this may still take some time. However, the operation is certainly quicker than the current approach which involves copying the tail of the current suffix file to form the new suffix file.

NOTE The implementation must maintain not only the individual "xyz.n" chunks, but also some record of the relevant "truncate-before" point (in case it occurs in the middle of a chunk).

Application to Irmin layers/GC: In order to reduce the overhead involved when creating a new suffix file, we can implement a suffix file using a chunked file. Then we can create the "new" suffix file simply by using truncate-before with the current suffix file. This effectively eliminates any on-disk overhead caused by copying the suffix file.