Anatomy of a header sort

Stewart A. Levin & Yinbin Ma

ABSTRACT

Routine trace sorting at SEP is basically a grid sort. When faced with spiral shooting around sea-bottom nodes, defining an offset grid was nonintuitive. Leveraging the venerable Unix disk-based `sort` program, we were able to complement our `Sort3d` with a gridless header `SortByHdrs` program capable of handling a couple billion traces if needed. Here we highlight some important subtleties in this approach.

INTRODUCTION

For some nodal data QC we applied hyperbolic moveout (LMO-like, not NMO-like) in order to assess orientation, drift and water velocity corrections as a function of offset. In this case a spiral shooting geometry made applying our usual grid-based `Sort3d` problematic. As years ago Dr. Levin had written a sort program for the massive first-generation SEAM dataset (`www.seg.org/resources/research/seam`), he confidently proposed, and we developed, a derivative module that simply sorted on trace header values so that we didn't need an offset-defined grid. Like the SEAM sort, the engine for this new sort is also the venerable Unix `sort` program.

SUBTLETIES

Unix `sort` operates on text files, not binary files. Therefore, to use it to sort SEP binary files, we create a textual index file with printed values for header keys and a sequential trace number. After sorting on the key values, we read back the reordered trace numbers to select and process the actual headers and traces.

Unix `sort` is capable of handling more than $2^{31}-1$ records, but since current seplib query and disk block access routines use four byte integers, that is the maximum number of traces we can sort.

The only (well) supported SEP trace header key formats are 32 bit integers and floats, the printed values of which are understood by GNU `sort`'s numeric comparison routines. To avoid tangling with floating point format precision selection, we use the observation that nonnegative IEEE floating point bit patterns increase monotonically when viewed as integers and hence sort order is preserved when that bit pattern is printed as an integer. For negative floating point numbers, we use the negative of the bit pattern of its absolute value. One issue with the numeric sort in general is that it is locale dependent. We handle this by overriding the default locale with LC_ALL=C for the sort process.

With network mounted filesystems it is notoriously difficult to ensure that a file written by one process can be immediately and fully seen by another process. This is not specific to the processes being on different computers, though that is where it is most often seen (or not seen). To bypass this potential problem, we communicate all data between `SortByHdrs` and `sort` via pipes instead of shared disk files. The SEP *datapath* query is used to set a suitably large directory for `sort`'s scratch files.

There are times when someone might want to interrupt a sort, but does not want to corrupt what has already been sorted due to partially completed I/O. We use a signal handler to set a graceful termination flag in the event of an interrupt.

The more recent versions starting with 8.6 (2010-10-15) of GNU `sort` automatically use multiprocessor parallelism for speed. Checking around at SEP, revealed that all the machines we've used have versions prior to that. A good reason to do some long delayed upgrading.

Finally, we explicitly wait on the sort process to terminate. This avoids possible "zombie" processes and undeleted temporary disk files. In the `SortByHdrs` code, this wait is directly for the `sort` process. When a more traditional *system*() or *popen*() mechanism is used to start a process, it is the shell that spawns the command and not the command itself that is the target of the wait.