Large atomic data in R: package 'ff'

Daniel Adler Jens Oehlschlägel Oleg Nenadic Walter Zucchini

A proof of concept for the **ff** package has won the large data competition at useR!2007 with its C++ core implementing fast memory mapped access to flat files. In the meantime we have complemented memory mapping with other techniques that allow fast and convenient access to large atomic data residing on disk. ff stores index information efficiently in a packed format, but only if packing saves RAM. HIP (hybrid index preprocessing) transparently converts random access into sorted access thereby avoiding unnecessary page swapping and HD head movements. The subscript C-code directly works on the hybrid index and takes care of mixed packed/unpacked/negative indices in ff objects; ff also supports character and logical indices. Several techniques allow performance improvements in special situations. ff arrays support optimized physical layout for quicker access along desired dimensions: while matrices in the R standard have faster access to columns than to rows, ff can create matrices with a row-wise layout and arbitrary 'dimorder' in the general array case. Thus one can for example quickly extract bootstrap samples of matrix rows. In addition to the usual `[` subscript and assignment `[<-` operators, ff supports a `swap` method that assigns new values and returns the corresponding old values in one access operation - saving a separate second one. Beyond assignment of values, the `[<-` and `swap` methods allow adding values (instead of replacing them). This again saves a second access in applications like bagging which need to accumulate votes. ff objects can be created, stored, used and removed, almost like standard R ram objects, but with hybrid copying semantics, which allows virtual `views` on a single ff object. This can be exploitet for dramatic performance improvements, for example when a matrix multiplication involves a matrix and it's (virtual) transpose. The exact behavior of ff can be customized through global and local `options`, finalizers and more.

The supported range of storage types was extended since the first release of ff, now including support for atomic types `raw`, `logical`, `integer` and `double` and ff data structures `vector` and `array`. A C++ template framework has been developed to map a broader range of signed and unsigned types to R storage types and provide handling of overflow checked operations and NAs. Using this we will support the packed types 'boolean' (1 bit), 'quad' (2 bit), 'nibble' (4 bit), 'byte' and 'unsigned byte' (8 bit), 'short', 'unsigned short' (16 bit) and 'single' (32bit float) as well as support for (dense) symmetric matrices with free and fixed diagonals. These extensions should be of some practical use, e.g. for efficient storage of genomic data (AGCT as.quad) or for working with large distance matrices (i.e. symmetric matrices with diagonal fixed at zero).