ALTREP: Alternate Representations of Basic R Objects

Luke Tierney

Department of Statistics & Actuarial Science
University of Iowa

October 27, 2018

Introduction

- R is widely used in the field of statistics and beyond, especially in university environments.
- R was originally developed by Robert Gentleman and Ross Ihaka in the early 1990's for a Macintosh computer lab at U. of Auckland, NZ.
- Since 1997 R has been developed and maintained by the *R-core* group, with 20 member located in 9 different countries.
- The S language, on which R is based, was originally developed at Bell Labs to support flexible data analysis.
- As S evolved, it was developed into a full language that also supports development of software for new methodology.
- R has become the primary framework for developing and making available new statistical methodology.
- Many (now over 13,000) extension packages are available through CRAN; more from Bioconductor and other repositories.

Introduction

- Since joining R-core in 1998 I have worked mostly on computational infrastructure, such as
  - memory management
  - name space management
  - error handling framework
  - compilation
  - parallel computing support
- Much of this is enabling technology not used directly by typical users or only by package authors.
- The topic of this talk is of a similar nature.

Introduction

- This is joint work with Gabe Becker and Tomas Kalibera.
- The C level R implementation works with a fixed set of data types, e.g. INTSXP, REALSXP, ENVSXP.
- These have a particular memory layout, but are accessed only through a function/macro abstraction.
- For vector data the accessors are
  - LENGTH for the number of elements;
  - DATAPTR (usually via INTEGER, REAL, etc.) for a pointer to a contiguous region in memory.
- The memory is typically allocated by malloc

Introduction

- ALTREP allows for alternate representations of these data types.
- Some examples of things we want to enable:
  - allow vector data to be
    - in a memory-mapped file;
    - distributed, e.g. within Apache Spark or Hadoop;
    - shared with other applications, e.g. with Apache Arrow;
  - allow compact representation of arithmetic sequences;
  - allow adding meta-data to objects;
  - allow computations/allocations to be deferred;
  - support alternative representations of environments.
- To existing C code ALTREP objects look like ordinary R objects.
- Updated C code may be able to take advantage of special features.
- Current state is available in the ALTREP SVN branch.
- More details are available in ALTREP.html at the branch root.
- Initial ALTREP support is available as of R 3.5.0.

Example: Compact Integer Sequences

- Vectors created by n1:n2, seq_along or seq_len can be represented compactly.
- In 3.4.x with JIT disabled:

  ```
  system.time(for (i in 1:1e9) break)
  ##   user  system elapsed
  ##  0.258   1.141   1.400
  ```

- In R 3.5.0 with ALTREP:

  ```
  system.time(for (i in 1:1e9) break)
  ##   user  system elapsed
  ##  0   0.004   0.000   0.003
  ```

Example: Compact Integer Sequences

- In R 3.4.x creating a larger sequence may fail:

  ```
  x <- 1:1e10
  ## Error: cannot allocate vector of size 74.5 Gb
  ```

- In R 3.5.0 with ALTREP this succeeds:

  ```
  x <- 1:1e10
  length(x)
  ## [1] 1e+10
  ```

- Some operations may still fail:

  ```
  y <- x + 1L
  ## Error: cannot allocate vector of size 74.5 Gb
  ```

Example: Deferred String Conversions

- Converting integers or reals to strings is expensive.
- In lm and glm default row labels on design matrices are created but rarely used.
- In R 3.5.0 the internal coerce function returns a *deferred string conversion* ALTREP object.
- This class has a subset method that returns another deferred conversion object.

Example: Deferred String Conversions

- For lm with $n = 10^7$ and $p = 2$:

```
x <- rnorm(1e7)
y <- x + rnorm(1e7)
system.time(lm(y ~ x))
##   user  system elapsed
## 19.804   0.860  20.703   R 3.4.2 patched
##  1.960   1.184   3.147   R 3.5.0 with ALTREP
```

- For glm:

```
system.time(glm(y ~ x))
##   user  system elapsed
## 20.880   1.624  22.517    R 3.4.2 patched
##  6.144   5.508  11.657    R 3.5.0 with ALTREP
```

- Deferred evaluation could be useful in many other settings as well.
- Linear or generalize linear model result objects are one example.

Example: Wrapper Objects and Attributes

- Currently changing an attribute on a shared vector requires a copy of the vector data.
- *Wrapper objects* can hold the new attribute value and a reference to the original object to access its data.
- The unclass function is sometimes used to drop a class attribute.
- In current R this forces a copy of the data, which can be expensive:

```
x <- structure(numeric(1e9), class = "foo")
system.time(base::unclass(x))
##   user  system elapsed
##  1.315   2.709   4.032
```

- Using a wrapper avoids the copy:

```
system.time(unclass(x))
##   user  system elapsed
##  0.010   0.003   0.012
```

- Automatic use of wrappers when changing attributes will most likely be added to R-devel soon.

Example: Wrapper Objects and Meta-Data

- Wrapper objects can also be used to attach meta-data, such as
  - is the vector sorted;
  - are there no NA values.
- The sort function returns a wrapper that records that the vector is sorted.
- Sorting a large vector takes some time:

```
x <- rnorm(1e8)
system.time(y <- sort(x))
##    user  system elapsed
##   8.300   0.576   8.924
```

- The result y is known to be sorted:

```
system.time(sort(y))
##    user  system elapsed
##       0       0       0
```

Example: Wrapper Objects and Meta-Data

- The sorting process will discover whether there are any NA values.
- When there are no NA values this is recorded by sort function in the returned wrapper.
- This information is checked by anyNA and used for a quick return:

```
system.time(anyNA(x))
##    user  system elapsed
##   0.062   0.000   0.061
system.time(anyNA(y))
##    user  system elapsed
##       0       0       0
```

Example: Wrapper Objects and Meta-Data

- Compact integer sequences also carry meta-data:

```
indx <- seq_along(x)
system.time(anyNA(indx))
##   user  system elapsed
##      0       0       0
system.time(sort(indx))
##   user  system elapsed
##      0       0       0
```

- ALTREP objects can also provide methods for some basic summaries:

```
system.time(sum(x))
##   user  system elapsed
##  0.176   0.000   0.176
system.time(sum(as.double(indx)))
##   user  system elapsed
##      0       0       0
```

- These summaries could be computed by special formulas or memoized.

Example: Memory Mapped Vectors

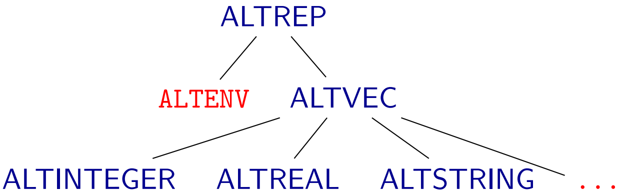
- R 3.5.0 includes experimental sample classes for memory mapped integer and real vectors.
- The file can be opened for reading and writing or in read-only mode.
- When used by ALTREP-aware code these will not result in allocating memory for holding all the data.
- Using non-aware functions may result in attempts to allocate large objects.
- The class provides an option for signaling an error when the raw data pointer is requested.
- A variant is also available as a small experimental package `simplemmap`.
- It should be possible to allow for files with 8, 16, or 64 bit integers, at the expense of translation overhead.

Mutable Vectors

- R uses pass by value semantics:
  - Conceptually, a function receives private copies of its arguments.
- This eliminates bugs that would otherwise occur, but at a cost.
- R can often avoid copying, but sometimes it cannot.
- It can be useful to have objects that are considered mutable, or passed by value, especially for internal data structures.
- A number of packages cheat on this at the C level.
- ALTREP may allow for providing mutable vectors in a more disciplined and safe way.
- Experiments on this are currently in progress.

Abstract Classes

- The framework is designed around a set of *abstract classes*:



ALTREP

ALTENV ALTVEC

ALTINTEGER ALTREAL ALTSTRING . . .

- The most specific classes correspond to R data types.
- Concrete classes specialize one of these.
- Each abstract class level defines a set of methods.
- Each concrete class has a table of method implementations.

Methods

General Methods

- ALTREP object methods:
  - Duplicate: SEXP Duplicate(SEXP x, Rboolean deep)
  - Coerce: SEXP Coerce(SEXP x, int type)
  - Length: R_xlen_t Length(SEXP x)
  - Inspect
- The standard operations defer to these methods for ALTREP objects.
- Duplicate and Coerce methods can return NULL to fall back to the default behavior.

Methods
Vector Methods

- ALTVEC methods:
  - Dataptr: SEXP Dataptr(SEXP x, Rboolean writeable)
  - Dataptr_or_null
  - Extract_subset
  - `Extract_subarray`
- Dataptr may need to allocate memory;
  - for now GC is suspended when calling the method.
- Dataptr_or_null will not allocate.
- Dataptr_or_null and Extract_subset can be used to avoid fully allocating an object.
- Obtaining a read-only data pointer is also sometimes useful.
- Adding `Extract_subarray` will help for interfacing to structured storage systems.

Methods
Specific Vector Methods

- Specific vector methods (patterned after JNI):
  - Elt
  - Get_region
  - No_NA
  - Is_sorted
  - and several others.
- Some numeric vector methods:
  - Min
  - Max
  - Sum
- A single method for extracting properties specified by a bitmask might be useful.

Changes to Existing Functions

- Existing functions will work without modification.
- But using a writable data pointer via REAL or INTEGER
  - may cause allocation or reading of full data;
  - may require flushing meta-data information.
- Some functions modified to avoid using the data pointer include
  - mean
  - min
  - max
  - sum
  - prod.
- These use Get_region to process data in chunks.
- Many more functions could be modified along these lines.

Changes to Existing Functions

- Subsetting has also been modified to avoid using the data pointer.
- This means, for example, that head and sample avoid allocation:

```
x <- 1:1e12
length(x)
## [1] 1e+12
head(x)
## [1] 1 2 3 4 5 6
> sample(x, 10)
## [1] 736617330192 392069636550 568241239321 224393184527
## [5] 851984238988 174365872796 366347672451  84457266227
## [9]  72327203393 761965661188
```

- Other operations attempt to allocate and fail:

```
x + 1
## Error: cannot allocate vector of size 7450.6 Gb
log(x)
## Error: cannot allocate vector of size 7450.6 Gb
```

Serialization and Package Support

- Classes can provide custom serialization by defining methods for
  - Serialized_state
  - Unserialize
- Packages can register ALTREP classes.
- Serialization records the package and class name.
- Unserializing loads the package namespace and looks up the registered class.
- A sample package implementing a memory mapped vector object is available on GitHub.

Serialization and Package Support

- Custom serialization requires a bump in the serialization version:
  - Older R versions cannot handle custom serializations; bumping the format version gives a clearer error message.
  - Some packages that make assumptions about the serialization format may need updates (e.g. `digest`).
  - This provides an opportunity for some other changes (e.g. recording native encoding information).
- The default serialization has been bumped in R-devel.
- Bumping the serialization version created unexpected problems because source packages contain serialized meta data for documentation and vignettes.

Serialization and Package Support

Skeleton of mmap Integer Implementation

```
/* MMAP Classes Objects */
static R_altrep_class_t mmap_integer_class;

/* ALTREP Methods */
static SEXP mmap_Serialized_state(SEXP x) { ... }
static SEXP mmap_Unserialize(SEXP class, SEXP state) { ... }

/* ALTVEC Methods */
static R_xlen_t mmap_Length(SEXP x) { ... }
static void *mmap_Dataptr(SEXP x, Rboolean writeable) { ... }
static void *mmap_Dataptr_or_null(SEXP x, Rboolean writeable) { ... }

/* ALTINTEGER Methods */
static int mmap_integer_Elt(SEXP x, R_xlen_t i) { ... }
static R_xlen_t mmap_integer_Get_region(SEXP sx, ...) { ... }

/* Constructor */
SEXP do_mmap_file(SEXP args) { ... }
```

Serialization and Package Support

Skeleton of mmap Integer Implementation

```c
void R_init_simplemmap(DllInfo *dll)
{
    /* create and initialize class objects */
    R_altrep_class_t cls =
        R_make_altinteger_class("mmap_integer", "simplemmap", dll);
    mmap_integer_class = cls;

    /* override methods */
    R_set_altrep_Unserialize_method(cls, mmap_Unserialize);
    ...
    R_set_altinteger_Get_region_method(cls, mmap_integer_Get_region);

    /* register public routines */
    static const R_ExternalMethodDef ExtEntries[] = {
        {"mmap_file", (DL_FUNC) &do_mmap_file, -1},
        {NULL, NULL, 0}
    };
    R_registerRoutines(dll, NULL, NULL, NULL, ExtEntries);
}
```

Some Implementation Details

- ALTREP objects are allocated as CONS cells with an altrep header bit set.
- Standard operations like LENGTH look at this bit to decide whether to dispatch.
- To allow efficient scalar identification there is also a scalar bit,
- With the ALTREP changes, operations like DATAPTR, STRING_ELT, and SET_STRING_ELT now might cause allocation.
- Eventually code should be rewritten to allow for this.
- For now, GC is suspended in these allocations.

Some Issues and Notes

- Performance can suffer due to:
  - overhead of checking altrep bit for standard objects;
  - dispatching overhead for ALTREP objects.
- Accessing the DATAPTR and possibly allocating may sometimes be much faster.
- Switching to an ALTREP may only pay off if objects are large.
- Deferred evaluations/allocations are very useful, but:
  - allocation failures can be delayed and come at unexpected times;
  - operations may produce unexpected large allocations, e.g. log(1:1e10);
  - some situations can lead to repeated evaluations.
- Memory mapping issues:
  - unserialization failure when the file is not available;
  - some settings might need a conversion layer (e.g. a file of 8-bit integers).
- Deferred edits might be useful for improving complex assignment performance.

Current Status

- The initial ALTREP infrastructure is incorporated in R 3.5.0, including
  - compact integer sequences;
  - deferred string conversions;
  - meta-data wrappers.
- The infrastructure is still experimental and may still change, but mostly through addition of methods.
- Package authors who might benefit from defining ALTREP classes are encouraged to give it a try.
- We may be setting up a GitHub organization for sharing experiments with new ALTREP classes.
- Experience with the package support framework will help to see if further changes are needed.

Next Steps

- Some additional data representations:
  - mutable vectors;
  - memory mapping with translation for byte count or byte order;
  - virtual subarrays;
  - virtual versions of rep results;
  - run-length encoding;
  - sparse vectors/arrays.
- More uses of deferred computation:
  - regression results;
  - reduction operations like log-likelihood computations;
  - ifelse alternatives;
  - edits in complex assignment.
- More use of meta data.
- Wrappers to avoid duplicating when changing attributes.
- Experiment with alternate environment representations.

Conclusions

- The ALTREP changes are evolutionary:
  - Existing code should continue to work.
  - Performance overhead should be minimal.
- The framework should help to
  - allow experimentation with some new ideas;
  - regularize some things currently being done.

Other Things

- Reference counting:
  - more maintainable;
  - allow less duplicating;
  - may help improving complex assignment performance.
- Compilation:
  - reduce remaining interpreted/compiled differences;
  - more optimization opportunities.
  - de-optimize when guard conditions fail.