The `lazy.frame` Package

Bryan W. Lewis
blewis@illposed.net

October 11, 2011

1 Preface

I've been working with some large-ish text files of comma separated values (CSV) recently. The files are each about three gigabytes with about 20 million rows. My computer has plenty of memory for R to load each file.

But, it takes a while.

And I'm impatient.

Now, I don't really need the entire data set in memory. I really just need to filter the data a bit and then sample from the rows. I think that this situation is typical enough–wanting fast access to subsets of large text files–that I wrote this package for it.

The `lazy.frame` package lets me quickly and efficiently work with subsets from a text file without loading the entire file into memory. A "lazy.frame" presents a text file as a kind of simple data frame, but without first loading the file into memory. Lazy frames lazily load data from their backing files only when required, for example by an indexing operation. They are essentially lazy wrappers for the `read.table` function with a few extra convenience functions.

There are several compelling R packages for working directly with file-backed data: The bigmemory package by Emerson and Kane provides a memory mapped matrix object, free from R indexing constraints, and a comprehensive suite of fast analysis functions. The nicely simple but powerful mmap package by Jeff Ryan defines a data frame-like memory mapped object. And the venerable ff package by Adler, Oehlschlägel, et. al. defines a variety of memory mapped data frame-like objects and functions. All of these packages have really interesting features. Most of them are designed to facilitate working with objects larger than the physical RAM available on a computer.

But, recall that my data sets fit into the RAM on my computer (RAM is really cheap)! My main irritation is the bottleneck incurred by parsing the entire data set, which isn't really avoided by the above packages (although some of the packages do include methods to help expedite loading data from text files).

Of course, lazy frames aren't a panacea and have limitations discussed below. For *really* large data sets, or for more sophisticated operations involving all the data, bigmemory is a better option. Lazy frames are really good for quickly extracting subsets from large text files with between roughly a million and a hundred million or so rows.

2 Using lazy.frame package

3 Limitations

4 Examples

I present a few examples that compare indexing operations on lazy frames with indexing operations on data frames read in by read.table. All experiments were conducted on a 2 GHz, four-core AMD Opetron computer with 12 GB of DDR-2 RAM running Ubuntu 9.10 GNU/Linux and R version 2.12.1. The data files resided on a Fusio-io ioXtreme solid state disk rated at 700MB/s data read rate and 80μs read latency. In order to minimize disk caching effects between tests, the command

```
echo 3 > /proc/sys/vm/drop_caches
```

was issued (wiping clean the Linux disk memory cache) just before each test.

4.1 An uncompressed file example

```
read.table results:
load time:
   user  system elapsed
648.380  33.350 682.699
            used   (Mb) gc trigger   (Mb)   max used   (Mb)
Ncells    138089    7.4     667722   35.7     380666   20.4
Vcells 285413776 2177.6  832606162 6352.3 1034548528 7893.0
[1] 17826159        27
subset time:
   user  system elapsed
  27.87    2.41   30.31
[1] 95166    27

read.table with colClasses results:
```

```
load time:
   user  system elapsed
443.290  82.780 526.141
            used   (Mb) gc trigger   (Mb)  max used   (Mb)
Ncells    138519    7.4     350000   18.7    350000   18.7
Vcells 285348278 2177.1  649037152 4951.8 641872298 4897.1
[1] 17826159       27
subset time:
   user  system elapsed
 28.410   2.180  30.593
[1] 95166    27

file.frame results:
load time:
   user  system elapsed
   2.34    2.05    4.39
         used (Mb) gc trigger (Mb) max used (Mb)
Ncells 140517  7.6     350000 18.7   350000 18.7
Vcells 130910  1.0     786432  6.0   531925  4.1
[1] 17826159       27
subset time:
   user  system elapsed
 40.870  11.770  52.709
[1] 95166    27
```