

# Benchmarking multiple out-of-memory strategies

Vincent J. Carey, *stvjc at channing.harvard.edu*

February 16, 2017

## Contents

---

1	Introduction	1
2	The harness	1
3	An example method (OOM benchmarker for HDF5 matrix)	2

## 1 Introduction

---

In many large-data situations, it is impractical to load and retain data in R's working memory space. We have had a look at HDF5, SQLite and tabix-indexed text as possible solutions to problems arising with memory constraints. We'll call these "out-of-memory" (OOM) approaches

How can we obtain data on which approach will be most effective for a given task? Comparative benchmarking is a very useful skill and we give a very rudimentary account of this here.

## 2 The harness

---

It is common to speak of a program that drives other programs as a "harness" (see [wikipedia](#) for related discussion). We have such a program in ph525x:

```
library(benchOOM)
benchOOM
## function (NR = 5000, NC = 100, times = 5, inseed = 1234, methods = list(.h5RoundTrip,
##   .ffRoundTrip, .slRoundTrip, .dtRoundTrip, .bmRoundTrip))
## {
##   nel = NR * NC
##   set.seed(inseed)
##   x = array(rnorm(nel), dim = c(NR, NC))
##   cbind(NR = NR, NC = NC, times = times, do.call(rbind, lapply(methods,
##     function(z) getStats(times, x, rtfun = z))), units = "microsec")
## }
## <environment: namespace:benchOOM>
```

This program is going to help us assess performance of various OOM approaches. We consider a very limited problem, that of managing data that could reside in an R matrix. The main parameters are

- NR and NC: row and column dimensions
- times: number of benchmark replications for averaging
- inseed: a seed for random number generation to ensure reproducibility
- methods: a list of methods

The `methods` parameter is most complex. Each element of the list is assumed to be a function with the matrix to be managed via OOM as the first argument, some additional parameters, and a parameter `intimes` that gives the number of benchmark replicates.

Our objective is to produce a table that looks like

```
> b1
  NR  NC times      meth      wr      ingFull      ing1K
1 5000 100    5      hdf5 10.71714   9.4100810 14.2984402
2 5000 100    5        ff 25.34365  63.0977338  4.4320688
3 5000 100    5     sqlite 174.89003 105.1254638 28.4717496
4 5000 100    5 data.table 49.35190   7.9871552 13.9007588
5 5000 100    5 bigmemory 23.39697   0.9660878  0.9950034
```

where each method listed in `meth` is asked to perform the same task a fixed number of times for averaging. The construction of the table occurs by binding together metadata about the task and method to the result of `getStats`. We'll leave the details of `getStats` to independent investigation.

### 3 An example method (OOM benchmarker for HDF5 matrix)

Let's look at the method for HDF5:

```
benchOOM:::h5RoundTrip
## function (x, chunkIn = c(1000, 10), inLevel = 0, intimes = 1)
## {
##   if (file.exists("ex_hdf5file.h5"))
##     file.remove("ex_hdf5file.h5")
##   requireNamespace("rhdf5")
##   h5createFile("ex_hdf5file.h5")
##   h5createDataset("ex_hdf5file.h5", "x", c(nrow(x), ncol(x)),
##     storage.mode = "double", chunk = chunkIn, level = inLevel)
##   mw = microbenchmark(h5write(x, "ex_hdf5file.h5", name = "x"),
##     times = intimes)
##   mr = microbenchmark(h5read("ex_hdf5file.h5", name = "x"),
##     times = intimes)
##   msel = microbenchmark(ysel <- h5read("ex_hdf5file.h5", name = "x",
##     index = list(4001:5000, 1:100)), times = intimes)
##   stopifnot(all.equal(ysel, x[4001:5000, ]))
##   list(mwrite = mw, ingFull = mr, ing1K = msel, times = intimes,
##     method = "hdf5")
## }
## <environment: namespace:benchOOM>
```

The program has three main phases

- HDF5-related setup, cleaning out any previous archives and establishing the basic target file
- Benchmarking of data export via `h5write`
- Benchmarking of ingestion via `h5read` with various restrictions

The results of `microbenchmark` are assembled in a list.