Using the qs package

qs – quick serialization of R objects

This package provides an interface for quickly writing (serializing) and reading (de-serializing) objects to and from disk. The goal of this package is to provide a lightning-fast and complete replacement for the saveRDS and readRDS functions in R.

Inspired by the fst package, qs uses a similar block-compression approach using the zstd library and direct "in memory" compression, which allows for lightning quick serialization. It differs in that it uses a more general approach for attributes and object references for common data types (numeric data, strings, lists, etc.), meaning any S3 object built on common data types, e.g., tibbles, time-stamps, bit64, etc. can be serialized. For less common data types (formulas, environments, functions, etc.), qs relies on built in R serialization functions via the RApiSerialize package followed by block-compression.

For character vectors, `qs` also uses the alt-rep system to quickly read in string data.

Features

The table below compares the features of different serialization approaches in R.

	qs	fst	saveRDS
Not Slow	Yes	Yes	No
Numeric Vectors	Yes	Yes	Yes
Integer Vectors	Yes	Yes	Yes
Logical Vectors	Yes	Yes	Yes
Character Vectors	Yes	Yes	Yes
Character Encoding	Yes	(vector-wide only)	Yes
Complex Vectors	Yes	No	Yes
Data.Frames	Yes	Yes	Yes
On disk row access	No	Yes	No
Attributes	Yes	Some	Yes
Lists / Nested Lists	Yes	No	Yes
Multi-threaded	No (Not Yet)	Yes	No

Summary Benchmarks

The table below lists serialization speed for several different data types (listed in MB/s).

	qwrite	qread	saveRDS	readRDS	write_fst threads:1	read_fst threads:1	write_fst threads:4	read_fst threads:4
Integer Vector	1015.2	889.8	27.1	135.5	686.6	442.4	699.1	567.9
Numeric Vector	861.2	954.0	24.3	131.9	744.0	638.7	754.4	848.0
Character Vector	1312.9	715.8*	49.1	43.9	1440.9	59.5	1536.3	59.3
List	197.2	311.5	7.7	123.5	N/A	N/A	N/A	N/A
Environment	56.0	117.5	7.7	89.6	N/A	N/A	N/A	N/A

Objects used for benchmarking

- Integer Vector: `sample(1e8)`
- Numeric Vector: `runif(1e8)`
- Character Vector: `qs::randomStrings(1e7)`
- List: `map(1:1e5,sample(100))`
- Environment:`x<-map(1:1e5,sample(100)); names(x)<-1:1e5; as.environment(x)`

Installation:

1. `devtools::install_github("traversc/qs")`

Example:

See `tests/correctness_testing.r` for more examples. Below is an example serializing a large `data.frame` to disk.

```
library(qs)
x1 <- data.frame(int = sample(5e6, replace=T),
                 num = rnorm(5e6),
                  char = randomStrings(5e6), stringsAsFactors = F)
qsave(x1, "/tmp/mydata.qs")

x2 <- qread("/tmp/mydata.qs")
identical(x1, x2) # returns true
```

```
## [1] TRUE
```

Additional Benchmarks

Data.Frame benchmark

Benchmarks for serializing and de-serializing large data.frames (5 million rows) composed of a numeric column (`rnorm`), an integer column (`sample(5e6)`), and a character vector column (random alphanumeric strings of length 50). See `vignettes/dataframe_bench.png` for a comparison using different compression parameters.

This benchmark also includes materialization of alt-rep data, for an apples-to-apples comparison.

Serialization speed with default parameters:

Method	write time (s)	read time (s)
qs	0.49391294	8.8818166
fst (1 thread)	0.37411811	8.9309314
fst (4 thread)	0.3676273	8.8565951
saveRDS	14.377122	12.467517

Serialization speed with different parameters

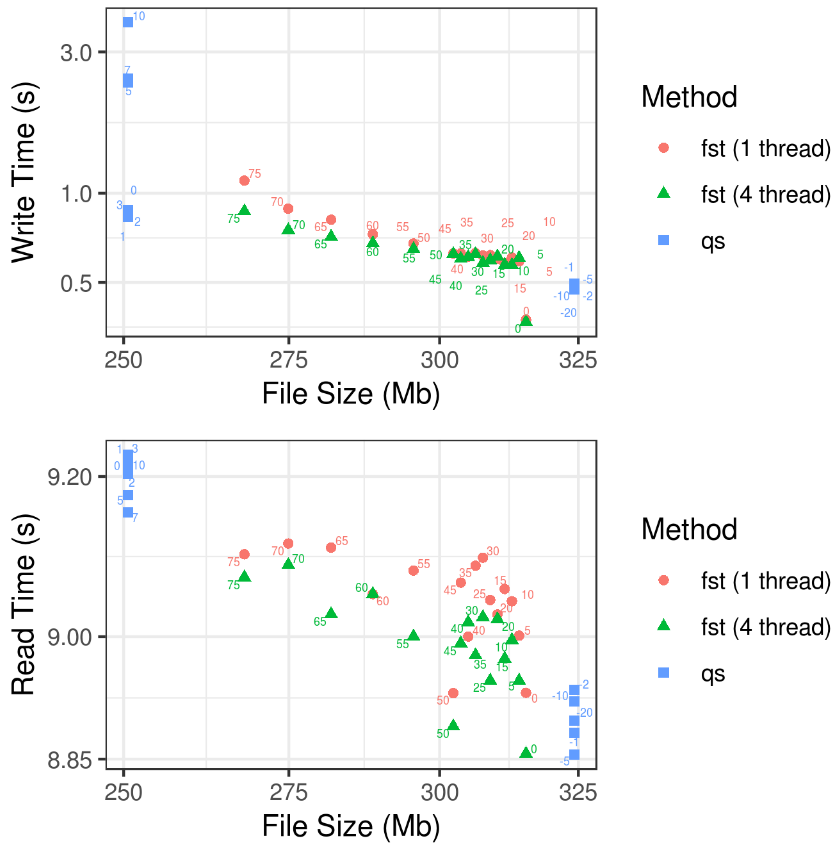

Figure 1: dataframe_bench

Nested List benchmark

Benchmarks for serialization of random nested lists with random attributes (approximately 50 Mb). See the nested list example in the tests/correctness_testing.r.

Serialization speed with default parameters

Method	write time (s)	read time (s)
qs	0.17840716	0.19489372
saveRDS	3.484225	0.58762548

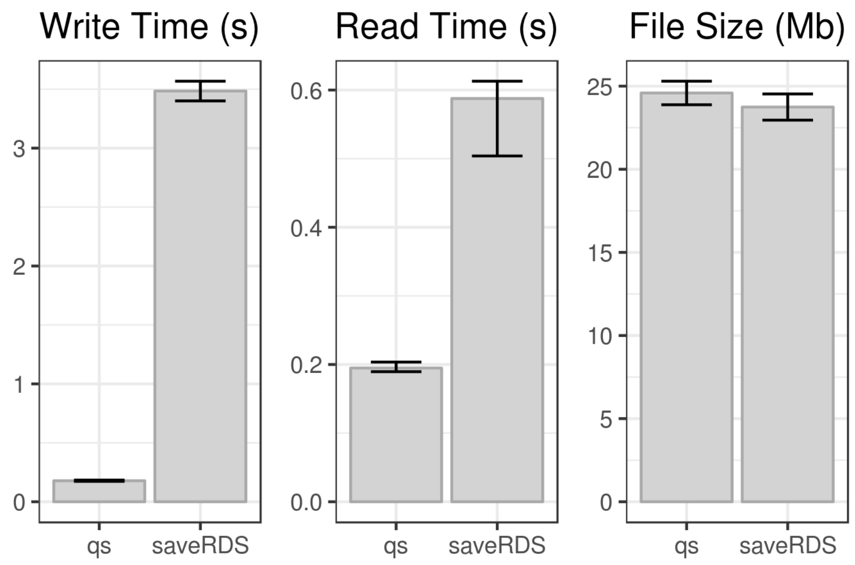

Figure 2: nested_list_bench