Streaming Data And Concurrency In R

Rory Winston

rory@theresearchkitchen.com

About Me

- Independent Software Consultant
- M.Sc. Applied Computing, 2000
- M.Sc. Finance, 2008
- Apache Committer
- Interested in practical applications of functional languages and machine learning
- *Really* interested in seeing R usage grow in finance

Parallelization vs. Concurrency in R

- Multithreading vs. parallelization
- i.e. `fork()` vs. `pthread_create()`
- R interpreter is single threaded
- Some historical context for this (e.g. non-threadsafe BLAS implementations)
- Multithreading can be complex and problematic
- Instead a focus on parallelization:
  - Distributed computation: `gridR`, `nws`, `snow`
  - Multicore/multi-cpu scaling: `Rmpi`, `Romp`, `pnmath`
  - Interfaces to PBLAS/Hadoop/OpenMP/MPI/Globus/etc.
- Parallelization suits large CPU-bound processing applications
- So do we really need it at all then?

Multithreading Is A Valuable Tool

- I say, "yes"
- For general real-time (streaming to be more precise) data analysis
- (Growing interest in using R for streaming data, not just offline analyis)
- GUI toolkit integration
- Fine-grained control over independent task execution
- Fine-grained control over CPU-bound and I/O-bound task management
- *"I believe that explicit concurrency management tools (i.e. a threads toolkit) are what we really need in R at this point."* - Luke Tierney, 2001

Will There Be A Multithreaded R?

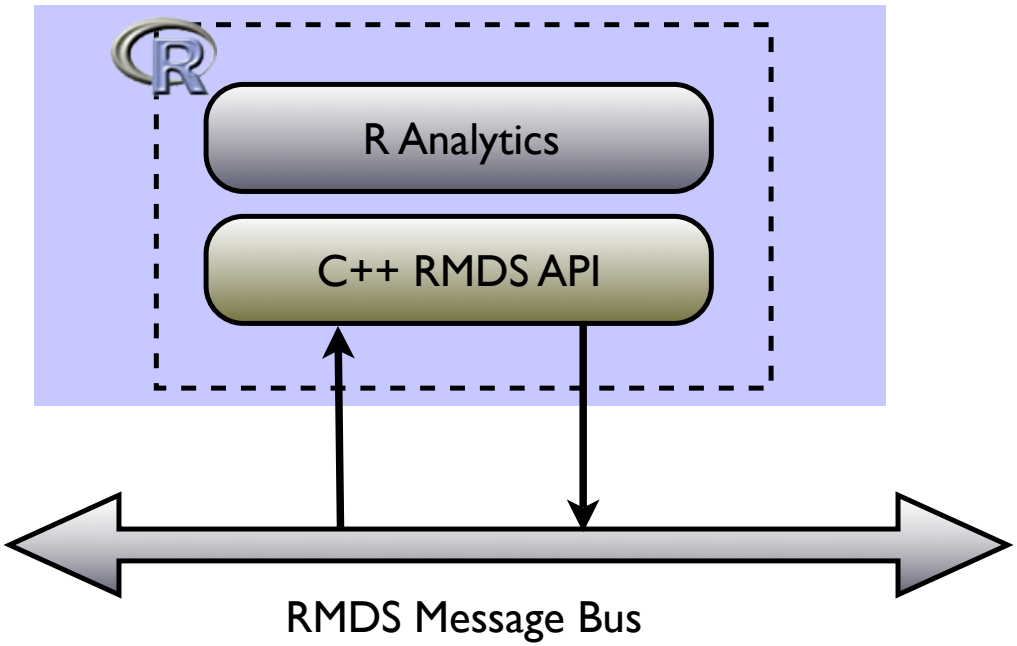
- Short answer is: Most likely not
- At least not in its current incarnation
- Internal workings of the interpreter not particularly amenable to concurrency:
  - Functions can manipulate caller state («- vs. <-)
  - Lazy evaluation machinery (promises)
  - Dynamic State, garbage collection, etc.
  - Scoping: global environments
  - Management of resources: streams, I/O, connections, sinks
- Implications for current code
- Possibly in the next language evolution (cf. Ihaka?)

Motivating Example

- Based on work I did last year and presented at UseR! 2008
- Wrote a real-time and historical market data service from Reuters/R
- The real-time interface used the Reuters C++ API
- R extension that spawned listening thread and handled market updates
- New version also does publishing as well as subscribing

Motivating Example

- The (real-world) example involves building a new high-frequency trading system
- Step 1 is handling market prices (in this case interbank currency prices)
- Need to ensure that the new system's prices are:
  - Correct;
  - Fast


R Analytics
C++ RMDS API
RMDS Message Bus

Issues With This Approach

- As R interpreter is single threaded, cannot spawn thread for callbacks
- Thus, interpreter thread is locked for the duration of subscription
- Not a great user experience
- Need to find alternative mechanism

Alternative Approach

- If we cannot run subscriber threads in-process, need to decouple
- Standard approach: add an extra layer and use some form of IPC
- For instance, we could:
  - Subscribe in a dedicated R process (A)
  - Push incoming data onto a socket
  - R process (B) reads from a listening socket
- Sockets could also be another IPC primitive, e.g. pipes, shared mem
- We will use the `bigmemoRy` package to leverage the latter

The bigmemoRy package

- From the description: `"Use C++ to create, store, access, and manipulate massive matrices"`
- Allows creation of large ($\geq$ RAM) matrices
- These matrices can be mapped to files/shared memory
- It is the shared memory functionality that we will use

```
big.matrix(nrow, ncol, type = "integer", ....)
shared.big.matrix(nrow, ncol, type = "integer", ...)
filebacked.big.matrix(nrow, ncol, type = "integer", ...)
read.big.matrix(file, sep=, ...)
```

Sample Usage

```
> library(bigmemory)
> X <- shared.big.matrix(type="double", ncol=1000, nrow=1000)
> X
An object of class “big.matrix”
Slot "address":
<pointer: 0x7378a0>
```

Create Shared Memory Descriptor

```
> desc <- describe(X)
> desc
$sharedType
[1] "SharedMemory"

$sharedName
[1] "53f14925-dca1-42a8-a547-e1bccae999ce"

$nrow
[1] 1000

$ncol
[1] 1000

$rowNames
NULL

$colNames
NULL

$type
[1] "double"
```

Export the Descriptor

In R session 1:

```
> dput(desc, file="/tmp/matrix.desc")
```

In R session 2:

```
> library(bigmemory)
> desc <- dget("/tmp/matrix.desc")
> X <- attach.big.matrix(desc)
```

Now R sessions A and B share the same `big.matrix` instance

Share Data Between Sessions

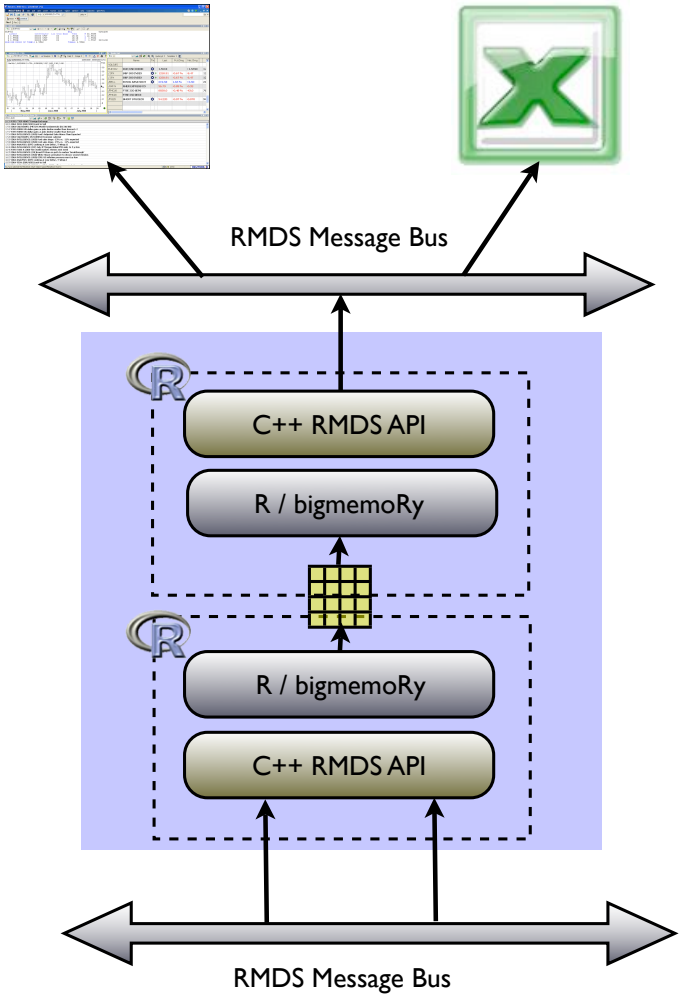
R session 1:

```
> X[1,1] <- 1.2345
```

R session 2:

```
> X[1,1]
[1] 1.2345
```

Thus, streaming data can be continuously fed into session A
And concurrently processed in session B


RMDS Message Bus
C++ RMDS API
R / bigmemoRy
R / bigmemoRy
C++ RMDS API
RMDS Message Bus

Summary

- Lack of threads not necessarily a barrier to concurrent analysis
- Packages like `bigmemoRy`, `nws`, etc. facilitate decoupling via IPC
- Could potentially take this further (using e.g. `nws`)

References

- bigmemoRy:
  http://cran.r-project.org/web/packages/bigmemory/
- Luke Tierney's original threading paper:
  http://www.cs.uiowa.edu/~luke/R/thrgui/
- HPC and R Survey:
  http://epub.ub.uni-muenchen.de/8991/
- Inside The Python GIL:
  www.dabeaz.com/python/GIL.pdf