Accessing Redis Data Caches via Rcpp

Dr Dirk Eddelbuettel
dirk@eddelbuettel.com
@eddelbuettel

R/Finance 2014
Lightning Talk
17 May 2014

Longer version available here
Outline

1. Redis
2. Speed
Why the hype?

- **Simple**: Does one thing, and does it well
- **Fast**: Run `redis-benchmark` to see just how fast
- **Widely used**: Twitter, GitHub, Craigslist, StackOverflow, …
- **Multi-language**: Bindings from anything you may use
- **Active**: Well maintained and documented
More generally

We can

- Read
- Write

from just about any programming language or shell.

(So far) all we require is string processing.
Redis supports many relevant data types:

- Strings
- Hashes
- Lists
- Sets
- Sorted Sets

as well as transactions, key management, pub/sub, embedded scripting, connection management and more.
Wonderful package by Bryan Lewis that covers (all of ?) Redis

Awesome for things like
\[
\text{redisSet("myModel", lm(someFormula, someData))}
\]

(Mostly) efficient enough.

Uses string format exclusively.

Automagically deploys R serialization.

Also used as backend for \texttt{doRedis}
Redis Speed

Simple helper functions

redisConnect("someServer.some.net")

rput <- function(X) {
  xstr <- deparse(substitute(X))
  redisSet(xstr, X)
}

rget <- function(key) {
  val <- redisGet(key)  # default instance
  redisDelete(key)
  invisible(val)
}
require(rredis)
redisConnect()

memoize <- function(expr, key=NULL, expire_time=Inf,
                        verbose=FALSE, envir=parent.frame()) {
    if (is.null(key)) {
        key <- paste(substitute(expr), collapse=

    }
    if (redisExists(key)) {
        ret <- redisGet(key)
    } else {
        ret <- eval(substitute(expr), envir=envir)
        redisSet(key, ret)
    }
    if (expire_time < Inf) {
        redisExpireAt(proj_doc_key,
                      as.integer(as.POSIXct(Sys.time())+expire_time))
    }
    ret
}
Outline

1. Redis
2. Speed

Dirk Eddelbuettel
Redis via Rcpp
Time series

Our basic premise and idea is to deploy disconnected writers (middleware clients in C, C++, Python, ...) and consumers (R) – by placing Redis in the middle.

But for “longer” time series the combined cost of deserialization and parsing is too high in R.
Example

```r
set.seed(123); N <- 2500
x <- xts(100*cumprod(1+rnorm(N)*0.005 +
    (runif(N)>0.95)*rnorm(N)*0.025),
    order.by=Sys.time()+cumsum(exp(3*runif(N))))
plot(x, main="Simulated Series", type='l')
```

Simulated Series
Writing and Reading

With `rredis` we set and get the time series as follows:

```r
setAsAscii <- function(dat) {
  N <- nrow(dat)
  ## insertion is row by row
  for (i in 1:N) {
    redisZAdd("ex:ascii:series",
                dat[i, 1], dat[i,])
  }
}

## retrieval is by list
getFromAscii <- function() {
  xx <- do.call(rbind,
                redisZRange("ex:ascii:series", 0, -1))
  xt <- xts(xx[, -1],
            order.by = as.POSIXct(xx[, 1], origin = "1970-01-01"))
}
```
A (fairly new) CRAN package we released recently.

It does just one thing: give us serialization and deserialization from the R API at the C(++) level.

It is used by RcppRedis, and provides it with C-level (de-)serialization without having to call “up” to R.
A (fairly new) (and highly incomplete) CRAN package (as of this week).

It covers just a couple of commands, but those run rather fast.
Writing and Reading

```r
setAsBinary <- function(dat) {
  redis$zadd("ex:bin:series", as.matrix(dat))
}

getFromBinary <- function() {
  zz <- redis$zrange("ex:bin:series", 0, -1)
  zt <- xts(zz[, -1],
            order.by=as.POSIXct(zz[, 1], origin="1970-01-01"))
}
```
// redis "zadd" -- insert score + matrix row (no R serial.)
// by convention, 1st elem of row vector is the score value
double zadd(std::string key, Rcpp::NumericMatrix x) {
    double res = 0;
    for (int i=0; i<x.nrow(); i++) {
        Rcpp::NumericVector y = x.row(i);
        // uses binary protocol, see hiredis doc at github
        redisReply *reply =
            static_cast<redisReply*>(redisCommand(prc_,
                "ZADD %s %f %b",
                key.c_str(),
                y[0],
                y.begin(),
                y.size() * szdb));

        checkReplyType(reply, replyInteger_t);
        res += static_cast<double>(reply->integer);
        freeReplyObject(reply);
    }
    return(res);
}
## Writing

<table>
<thead>
<tr>
<th>Test</th>
<th>Replications</th>
<th>Elapsed</th>
<th>Relative</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>setAsBinary</code></td>
<td>1</td>
<td>0.127</td>
<td>1.000</td>
</tr>
<tr>
<td><code>setAsAscii</code></td>
<td>1</td>
<td>100.001</td>
<td>787.409</td>
</tr>
</tbody>
</table>

## Reading

<table>
<thead>
<tr>
<th>Test</th>
<th>Replications</th>
<th>Elapsed</th>
<th>Relative</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getFromBinary</code></td>
<td>10</td>
<td>0.031</td>
<td>1.000</td>
</tr>
<tr>
<td><code>getFromAscii</code></td>
<td>10</td>
<td>4.792</td>
<td>154.581</td>
</tr>
</tbody>
</table>
Right now the **RcppRedis** package straddles three worlds:

- Strings to communicate with Python, C++, cmdline, ...
- Raw R strings and (de-)serialization to talk to **rredis**
- Binary data (as vectors) for efficient time series storage.

We don’t plan to provide the cross-product of encodings and commands, but rather pick and choose.

We now have **Shiny** apps that slice and dice (near) real-time series related to trading. And I am not going to say more.
This short talk tried to convince you that

- Redis is cooler than sliced bread.
- `rredis` is a wonderful package you should use.
- Redis also allows binary connection.
- (Lots of) string-to-numeric conversions are slow.
- `Rcpp` is ready, willing and able to help.
- `RcppRedis` helps overcome a few bottlenecks.

`RcppRedis` is open for collaboration. See what it does, see what it misses, and consider contributing to it.