kdb+ and R

Contents

Note on highlighted text:

Lines highlighted in blue are q commands and could be run in a q session. You need only enter those lines beginning with a `q)` prompt, and can ignore comments which begin with `/` followed by a space. Any results are shown below the prompt.

For example in the following, you would type in `sum 2 3 5 7` and press Enter. The system displays the result of `17`.

```
q)sum 2 3 5 7      / example of sum
17
```

Note: commands entered in the q console must be on a single line. However, commands can be given over several lines in a script.

Lines highlighted in brown are R code. You can enter any lines beginning with `>` and the results are displayed below the prompt. Comments in R are prefixed with `#`

```
# This is a comment
> sum(2,3,5,7)
[1] 17
```

1 Introduction

kdb+ and R[1] are complementary technologies. kdb+ is the world's leading timeseries database and incorporates a programming language called q. R is a programming language and environment for statistical computing and graphics. Both are tools used by data scientists to interrogate and analyze data. Their features sets overlap in that they both:

- are interactive development environments
- incorporate vector languages
- have an built-in set of statistical operations
- can be extended by the user
- are well suited for both structured and ad-hoc analysis

They do however have several differences:

- kdb+ can store and analyze petabytes of data directly from disk whereas R is limited to reading data into memory for analysis
- kdb+ has a larger set of datatypes including extensive temporal times (timestamp, timespan, time, second, minute, date, month) which make temporal arithmetic easy
- R contains advanced graphing capabilities whereas kdb+ does not
- built in routines in kdb+ are generally faster than R
- R has a much larger set of prebuilt statistical routines

When used together, kdb+ and R provide an excellent platform for easily performing advanced statistical analysis and visualization on large volumes of data.

2 kdb+ and R Working Together

Given the complementary characteristics of the two languages, it is important to utilize their respective strengths. All the analysis could be done in kdb+; the q language is sufficiently flexible and powerful to replicate any of the pre-built R functions. Below are some best practice guidelines, although where the line is drawn between kdb+ and R will depend on both the system architecture and the strengths of the data scientists using the system.

[1] `http://www.r-project.org`

1. Do as much of the analysis as possible in kdb+. kdb+ analyzes the data directly from the disk and it is always most efficient to do as much work as possible as close to the data as possible. Whenever possible avoid extracting large raw datasets from kdb+. When extractions are required, use kdb+ to create smaller aggregated datasets
2. Do not re-implement tried and tested R routines in q unless they either
   (a) can be written more efficiently in q and are going to be called often
   (b) require more data than is feasible to ship to the R installation
3. Use R for data visualization

2.1 Interfaces

There are four ways kdb+ and R can be interfaced:

1. R can connect to kdb+ and extract data (section 3)
2. kdb+ can instantiate an R instance in its local memory space and invoke R routines locally (section 4.1)
3. kdb+ can connect to a remote instance of R via TCP/IP and invoke R routines remotely (section 4.2)
4. kdb+ can load the R maths library and invoke the R maths routines locally (section 4.3)

From the point of view of the user, kdb+ and R may be:

- both on the local machine
- one local and one remote
- both remote but local to each other
- both remote and remote from each other

Considering the potential size of the data, it is probably more likely that the kdb+ installation containing the data will be hosted remotely from the user. Points to consider when selecting the integration method are:

- if interactive graphing is required, either interface (1) or (2) must be used
- interface (2) can only be used if the kdb+ and R installations are installed on the same server
- interfaces (2) and (4) require less data transfer between (possibly remote) processes

- interfaces (2) and (3) both require variables to be copied from kdb+ to R for processing, meaning that at some point in time two copies of the variable will exist, increasing total memory requirements

2.2 Performance Testing

Generally speaking, running analysis in kdb+ will be faster than R either by virtue of the implementation or due to the data being local to kdb+. Both languages have similar facilities for testing the performance of code. The below code samples calculate the time taken to do matrix multiplication on identical 100x100 matrices, 1000 times. kdb+ timings are in milliseconds, R timings are in seconds. For this operation kdb+ is more than 5 times faster than R (238ms against 1240ms).

In q:

```
q)a:100 100#`float$til 100000
q)b:100 100#`float$til 100000
q)\t do[1000;a mmu b]
238
```

and in R:

```
> a<-matrix(0:99999,ncol=100,nrow=100)
> b<-matrix(0:99999,ncol=100,nrow=100)
> system.time(replicate(1000,a%*%b))
   user  system elapsed
  0.252   0.554   1.240
```

3 Calling kdb+ from R

3.1 Q Server for R

This is the most likely use case- users who are comfortable in R connecting to a kdb+ database to extract partially analyzed data into R for further local manipulation, analysis and display. A prebuilt interface from R to kdb+ is available at

`http://code.kx.com/wiki/Cookbook/IntegratingWithR/QServer`

and is currently available for linux (64bit), Windows (32bit and 64bit) and OSX operating systems. The interface allows R to connect to a kdb+ database and send a request to it, which may or may not return a result. There are three methods available:

- `open_connection(hostname,port,username:password)`: open a connection to a kdb+ database. Multiple connections can be open at once

- `close_connection(connectionhandle)`: close a connection
- `execute(connectionhandle,request)`: execute a request on the specified connection handle

To open and initialize a connection:

```
> source("c:/r/qServer.R")
> h<-open_connection("127.0.0.1",5001,"test:pass")
```

To request data and plot it:

```
> execute(h,"select count i by date from trades")
        date       x
1 2014-01-09 5125833
2 2014-01-10 5902700
3 2014-01-13 4419596
4 2014-01-14 4106744
5 2014-01-15 6156630
> res<-execute(h,"select tradecount:count i, sum size, last price, vwap:
    size wavg price by 0D00:05 xbar time from trades where date
    =2014.01.14,sym=`GOOG")
> head(res)
                  time tradecount     size price     vwap
1   2014-01-14 08:00:00       4358 11517850 13.29 13.15321
2   2014-01-14 08:05:00       4546 11880679 13.93 13.44213
3   2014-01-14 08:10:00       4388 11639714 14.31 13.88684
4   2014-01-14 08:15:00       4495 11847832 14.34 14.44223
5   2014-01-14 08:20:00       4406 11568602 14.56 14.27125
6   2014-01-14 08:25:00       4528 11771566 14.83 14.62999
> plot(res$time,res$price,type="l",xlab="time",ylab="price")
```

which produces the plot in Figure 1:

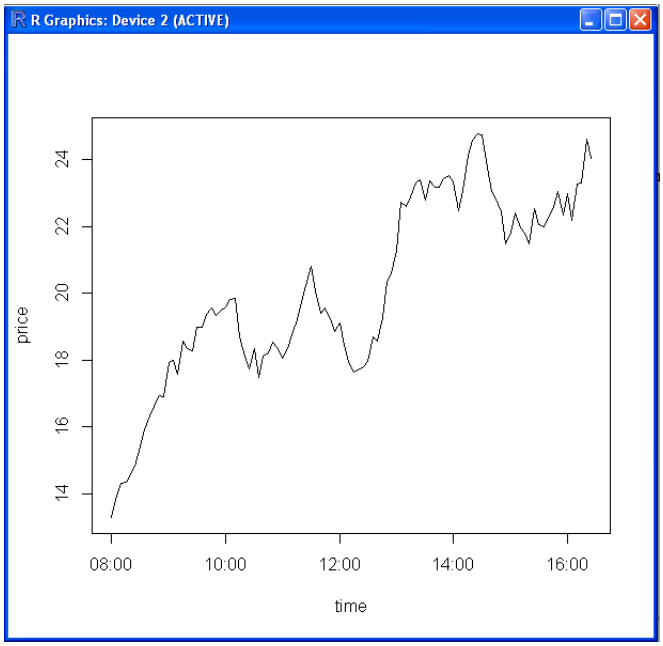

Figure 1: Last traded price plot drawn from R

More comprehensive graphing is available in additional R packages which can be freely downloaded. For example, using the `xts` package:

```
> library(xts)

# extract the HLOC buckets in 5 minute intervals
> res<-execute(h,"select high:max price,low:min price, open:first price,
    close:last price by 0D00:05 xbar time from trades where date
    =2014.01.09,sym=`GOOG")

# create an xts object from the returned data frame
# order on the time column
> resxts<-xts(res[,-1], order.by=res[,'time'])

# create a vector of colours for the graph
# based on the relative open and close prices
> candlecolors <- ifelse(resxts[,'close'] > resxts[,'open'], 'GREEN', 'RED
    ')

# display the candle graph with approrpiate labels
> plot.xts(resxts,type='candles',width=100,candle.col=candlecolors,bar.col
    ='BLACK',xlab="time",ylab="price",main="GOOG HLOC")
```

produces the plot in Figure 2:

Figure 2: Candlestick plot using xts package

Another popular package is the `quantmod` package which contains the chartSeries function.

```
> library(quantmod)

# extract the last closing price in 30 second buckets
> res<-execute(h,"select close:last price by 0D00:00:30 xbar time from
    trades where date=2014.01.09,sym=`GOOG")

# create the closing price xts object
> closes<-xts(res[,-1], order.by=res[,'time'])

# chart it.  Add Bollinger Bands to the main graph
# add additional Relative Strength Indicator and Rate Of Change graphs
> chartSeries(closes,theme=chartTheme("white"),TA=c(addBBands(),addTA(RSI(
    closes)),addTA(ROC(closes))))
```

This produces the plot shown in Figure 3:

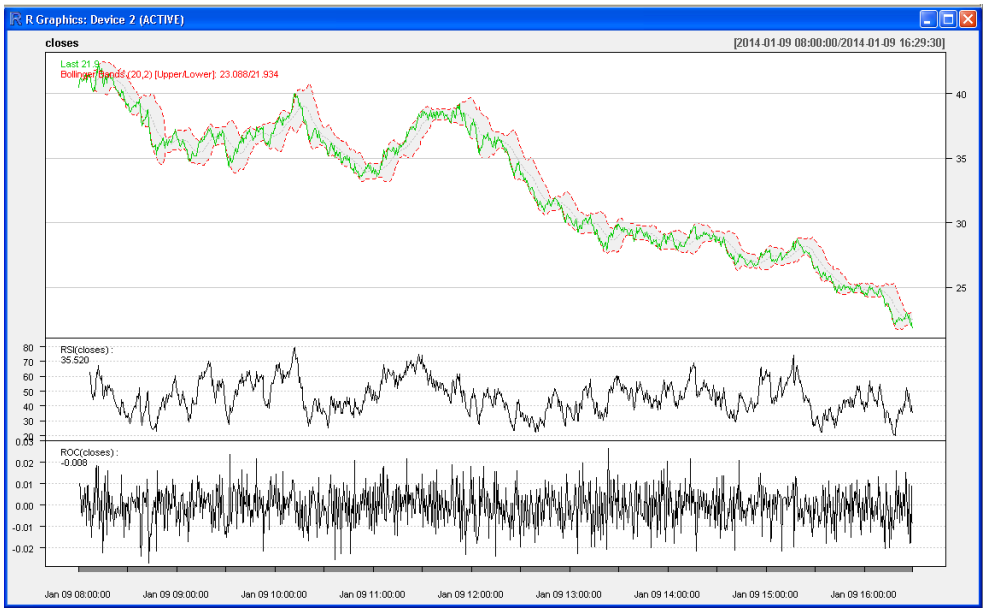

Figure 3: Chart example from quantmod package

Close the connection when done:

```
> close_connection(h)
[1] 0
```

3.2 ODBC

Although it is not the preferred method, the kdb+ ODBC driver can be used to connect to kdb+ from R if R is running on Windows. Details of how to install the ODBC driver can be found at

`http://code.kx.com/wiki/Cookbook/ODBC/qserver`

The RODBC package should be installed in R. An example is given below.

```
# install RODBC
> install.packages("RODBC")

# load it
> library(RODBC)

# create a connection to a predefined DSN
> ch <- odbcConnect("localhost:5000")

# run a query
# s.k should be installed on the kdb+ server to enable standard SQL
# However, all statements can be prefixed with q) to run standard q.
> res <- sqlQuery(ch, paste('q)select count i by date from trade'))
```

4 Calling R from kdb+

There are three interfaces which allow you to invoke R from kdb+. 32bit and 64bit versions are available at

`http://code.kx.com/wiki/Cookbook/IntegratingWithR`

If the appropriate version is not available for your target system, it can be built from source by following the instructions outlined in the associated README.

4.1 R in the kdb+ Memory Space

A shared library can be loaded which brings R into the kdb+ memory space, meaning all the R statistical routines and graphing capabilities can be invoked directly from kdb+. Using this method means data is not passed between remote processes. The library has five methods:

- `Ropen`: open R
- `Rclose`: close R
- `Rcmd`: run an R command, do not return a result
- `Rget`: run an R command, return the result to kdb+
- `Rset`: set a variable in the R memory space

An example is outlined below, using kdb+ to subselect some data and then passing it to R for graphical display.

```
q)select count i by date from trades
date      | x
----------| ----
2014.01.13| 1025
2014.01.14| 927
2014.01.15| 1139
2014.01.16| 799
2014.01.17| 968

/- extract mid prices in 5 minute bars
q)mids:select mid:last .5*bid+ask by 0D00:05 xbar time from quotes where
    date=2014.01.17,sym=`IBM
q)mids
time                          | mid
------------------------------| ------
2014.01.17D08:00:00.000000000| 45.24
2014.01.17D08:05:00.000000000| 45.24
2014.01.17D08:10:00.000000000| 45.255
2014.01.17D08:15:00.000000000| 45.355
2014.01.17D08:20:00.000000000| 45.435
..

/- Load in R
q)\l rinit.q

/- Pass the table into the R memory space
```

```
q)Rset["mids";mids]
0i

/- Graph it
q)Rcmd["plot(mids$time,mids$mid,type=\"l\",xlab=\"time\",ylab=\"price\")"]
0i
```

This will produce a plot as shown in Figure 4:

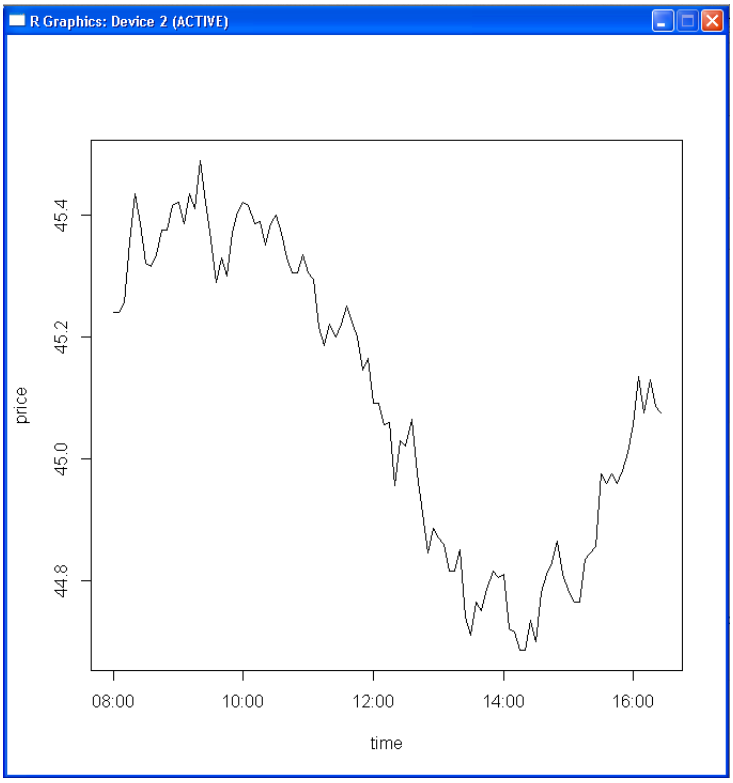

Figure 4: Quote mid price plot drawn from kdb+

To close the graphics window, use dev.off() rather than the close button on the window.

```
q)Rcmd["dev.off()"]
0i
```

Alternatively, the table can be written to a file with

```
q)Rcmd["pdf(\"test.pdf\")"]
0i
q)Rcmd["plot(mids$time,mids$mid,type=\"l\",xlab=\"time\",ylab=\"price\")"]
0i
q)Rcmd["dev.off()"]
0i
```

If the kdb+ and R installations are running remotely from the user on a linux machine, the graphics can be seen locally using X11 forwarding over ssh.

4.2 Remote R: Rserve

Rserve[2] allows applications to connect remotely to an R instance over TCP/IP. The methods are the same as those outlined in section 4.1, the difference being that all data is passed over TCP/IP rather than existing in the same memory space.

Every connection to Rserve has a separate workspace and working directory which means user defined variables and functions with name clashes will not overwrite each other. This is different to the previous method where if two users are using the same kdb+ process they can overwrite each others variables in both the kdb+ and R workspaces.

4.3 R Math Library

R contains a maths library which can be compiled standalone. The functions can then be exposed to kdb+ by wrapping them in C code which handles the mapping between R datatypes and kdb+ datatypes (K objects). A partial implementation of this is available at

`http://code.kx.com/wiki/Cookbook/IntegratingWithR`

```
/ load the rmaths library
q)\l rmath.q

/ invoke r math routines
/ Random gamma variates with specified shape and scale
q)rgamma
{[n;shape;scale] rg[`int$n; `float$shape; `float$scale]}
q)rgamma[5;3;.6]
4.495459 1.602277 1.310856 1.036715 2.323034
```

5 Example: Correlating Stock Price Returns

R has an in-built operation to produce a correlation matrix of aligned datasets. kdb+ does not, but one can easily be built as shown in section 5.4. In this example we will investigate different approaches for calculating the correlation of time-bucketed returns for a set of financial instruments. Possible approaches are:

[2] `http://www.rforge.net/Rserve`

1. Extract raw data from kdb+ into R for each instrument and calculate the correlation
2. Extract bucketed data from kdb+ into R, align data from different instruments, and correlate
3. Extract bucketed data with prices for different instruments aligned from kdb+ into R, and correlate
4. Calculate the correlations in kdb+

We will use a randomly created set of equities data, with prices ticking between 8am and 5pm.

```
q)select count i by date from trades where date.month=2014.01m
date      | x
----------| -------
2014.01.09| 5125833
2014.01.10| 5902700
2014.01.13| 4419596
2014.01.14| 4106744
2014.01.15| 6156630

q)select count i by date from trades where date.month=2014.01m,sym=`GOOG
date      | x
----------| ------
2014.01.09| 569187
2014.01.10| 655535
2014.01.13| 490966
2014.01.14| 456447
2014.01.15| 685665
```

We will use the R interface defined in section 3. The R and kdb+ installations are on the same host, so data extract timings do not include network transfer time but do include the standard serialization and de-serialization of data. We will load the interface and connect to kdb+ with:

```
> source("c:/r/qServer.R")
> h <- open_connection("127.0.0.1",9998,NULL)
```

5.1 Extract Raw Data into R

Retrieving raw data into R and doing all the processing on the R side is the least efficient way to approach this task in relation to processing time, network utilization, and memory usage. To illustrate, we can time how long it takes to extract one day's worth of data for one symbol from kdb+ to R.

```
> system.time(res <- execute(h, "select time,sym,price from trades where
    date=2014.01.09,sym=`GOOG"))
   user  system elapsed
   0.27    0.03    0.59
```

Given we wish to process over multiple dates and instruments, we will discount this method.

5.2 Extract Aggregated Data into R

The second approach is to extract aggregated statistics from kdb+ to R. The required statistics in this case are the price returns between consecutive time buckets for each instrument. The following q function extracts time bucketed data:

```
timebucketedstocks:{[startdate; enddate; symbols; timebucket]

 / extract the time bucketed data
 data:select last price
  by date,sym,timebucket xbar time
  from trades
  where date within (startdate;enddate),sym in symbols;

 / calculate returns between prices in consecutive buckets
 data:update return:1^price%prev price by sym from data;

 / return the results unkeyed
 () xkey data}
```

An example is:

```
q)timebucketedstocks[2014.01.09;2014.01.13;`GOOG`IBM;0D00:05]
date       sym  time                                  price return
-----------------------------------------------------------------
2014.01.09 GOOG 2014.01.09D08:00:00.000000000 41.26 1
2014.01.09 GOOG 2014.01.09D08:05:00.000000000 40.48 0.9810955
2014.01.09 GOOG 2014.01.09D08:10:00.000000000 41.01 1.013093
2014.01.09 GOOG 2014.01.09D08:15:00.000000000 41.2  1.004633
2014.01.09 GOOG 2014.01.09D08:20:00.000000000 40.06 0.9723301
2014.01.09 GOOG 2014.01.09D08:25:00.000000000 39.21 0.9787818
2014.01.09 GOOG 2014.01.09D08:30:00.000000000 39.09 0.9969396
..
```

Once the data is in R it needs to be aligned and correlated. To align the data we will use a pivot function defined in the reshape package.

```
# Reduce the dataset as much as possible
# only extract the columns we will use
> res <- execute(h,"select time,sym,return from timebucketedstocks
    [2014.01.09; 2014.01.15; `GOOG`IBM`MSFT; 0D00:05]")
> t
                   time  sym    return
1    2014-01-09 08:00:00 GOOG 1.0000000
2    2014-01-09 08:05:00 GOOG 0.9810955
3    2014-01-09 08:10:00 GOOG 1.0130929
4    2014-01-09 08:15:00 GOOG 1.0046330
5    2014-01-09 08:20:00 GOOG 0.9723301
6    2014-01-09 08:25:00 GOOG 0.9787818

> install.packages('reshape')
> library(reshape)

# Pivot the data using the re-shape package
> p <- cast(res, time~sym)
Using return as value column.  Use the value argument to cast to override
    this choice
> p
                  time      GOOG       IBM      MSFT
```

```
1   2014-01-09 08:00:00 1.0000000 1.0000000 1.0000000
2   2014-01-09 08:05:00 0.9810955 1.0102660 1.0372460
3   2014-01-09 08:10:00 1.0130929 0.9727483 1.0190424
4   2014-01-09 08:15:00 1.0046330 0.9586895 1.0109450
5   2014-01-09 08:20:00 0.9723301 1.0252600 1.0081859
6   2014-01-09 08:25:00 0.9787818 0.9852657 1.0285490

# And generate the correlation matrix
> cor(p)
            GOOG        IBM         MSFT
GOOG   1.00000000 0.05646584 -0.03420395
IBM    0.05646584 1.00000000  0.01728553
MSFT -0.03420395 0.01728553  1.00000000
```

An interesting consideration is the timing for each of the steps and how that changes when the dataset gets larger.

```
> system.time(res <- execute(h,"select time,sym,return from
    timebucketedstocks[2014.01.09; 2014.01.15; `GOOG`IBM`MSFT; 0D00:05]"))
   user  system elapsed
   0.00    0.00    0.79

> system.time(replicate(10,p<-cast(res,time~sym)))
   user  system elapsed
   0.36    0.00    0.36

> system.time(replicate(100,cor(p)))
   user  system elapsed
   0.05    0.00    0.04
```

We can see from the above that

- the data extract to R takes 790ms. Almost all of this time is taken up by kdb+ producing the dataset. There is minimal transport cost (as the processes are on the same host);

```
q)\t select time,sym,return from timebucketedstocks[2014.01.09;
    2014.01.15; `GOOG`IBM`MSFT; 0D00:05]
780
```

- the pivot takes approximately 40ms
- the correlation time is negligible (less than 1ms)

We can also analyze how these figures change as the dataset grows. If we choose a more granular time period for bucketing the data set will be larger. In our case we will use 10 second buckets rather than 5 minute buckets, meaning the result data set will be 30x larger.

```
> system.time(res <- execute(h,"select time,sym,return from
    timebucketedstocks[2014.01.09; 2014.01.15; `GOOG`IBM`MSFT; 0D00
    :00:10]"))
   user  system elapsed
   0.00    0.00    0.89

> system.time(p<-cast(res,time~sym))
Using return as value column.  Use the value argument to cast to override
    this choice
```

```
   user  system elapsed
   1.41    0.00    1.44
```

We can see that the time to extract the data increases by 100ms. The kdb+ query time increases by 30ms, so the majority of the increase is due to shipping the larger dataset from kdb+ to R.

```
q)\t select time,sym,return from timebucketedstocks[2014.01.09;
    2014.01.15; `GOOG`IBM`MSFT; 0D00:00:10]
821
```

The pivot time on the larger data set grows from 40ms to 1440ms giving a total time to do the analysis of approximately 2300ms. As the dataset grows, the time to pivot the data in R starts to dominate the overall time.

5.3 Align Data in kdb+

Given the pivot performance in R, an alternative is to pivot the data on the kdb+ side. This has the added benefit of reducing the volume of data transported due to the fact that we can drop the time and sym identification columns as the data is already aligned. The below q function pivots the data. An explanation can be found at

`http://code.kx.com/wiki/Pivot`

```
timebucketedpivot:{[startdate; enddate; symbols; timebucket]

 / Extract the time bucketed data
 data:timebucketedstocks[startdate;enddate;symbols;timebucket];

 / Get the distinct list of column names (the instruments)
 colheaders:value asc exec distinct sym from data;

 / Pivot the table
 pivot:exec colheaders#(sym!return) by time:time from data;

 / Fill with 1 because if no value, the price has stayed the same
 / return the results unkeyed
 () xkey 1^pivot}
```

An example is:

```
q)timebucketedpivot[2014.01.09;2014.01.13;`GOOG`IBM;0D00:05]
time                          GOOG      IBM
-----------------------------------------------
2014.01.09D08:00:00.000000000 1         1
2014.01.09D08:05:00.000000000 0.9810955 1.010266
2014.01.09D08:10:00.000000000 1.013093  0.9727483
2014.01.09D08:15:00.000000000 1.004633  0.9586895
2014.01.09D08:20:00.000000000 0.9723301 1.02526
2014.01.09D08:25:00.000000000 0.9787818 0.9852657
2014.01.09D08:30:00.000000000 0.9969396 1.011277
..
```

Using the larger dataset example, we can then do

```
> system.time(res <- execute(h,"delete time from timebucketedpivot
    [2014.01.09; 2014.01.15; `GOOG`IBM`MSFT; 0D00:00:10]"))
   user  system elapsed
   0.00    0.00    0.86

> cor(res)
              GOOG           IBM          MSFT
GOOG   1.000000e+00  8.044209e-05 -0.0014963017
IBM    8.044209e-05  1.000000e+00 -0.0001569679
MSFT -1.496302e-03 -1.569679e-04  1.0000000000
```

thus reducing the total query time from 2300ms to 860ms and additionally reducing the network usage.

5.4 Correlations in kdb+

A final approach is to calculate the correlations in kdb+ meaning that R is not used for any statistical analysis. The below function invokes the previously defined functions and creates the correlation matrix.

```
correlationmatrix:{[startdate; enddate; symbols; timebucket]

 / Extract the pivoted data
 data:timebucketedpivot[startdate;enddate;symbols;timebucket];

 / Make sure the symbol list is distinct
 / and only contains values present in the data
 symbols:asc distinct symbols inter exec distinct sym from data;

 / Calculate the list of pairs to correlate
 pairs:raze {first[x],/:1 _ x}each {1 _ x}\[symbols];

 / Return the pair correlation
 / Calculate two rows for each pair, with the same value in each
 correlatepair:{[data;pair]
  ([]s1:pair;s2:reverse pair; correlation:cor[data pair 0; data pair 1])};
 paircor:raze correlatepair[flip delete time from data] each pairs;

 / Pivot the data to give a matrix
 pivot:exec symbols#sym1!correlation by sym:s2 from paircor;

 / fill with 1 for the diagonal, unkey
 () xkey 1f^pivot}
```

which can be run like this:

```
q)correlationmatrix[2014.01.09; 2014.01.15; `GOOG`IBM`MSFT; 0D00:00:10]
sym  GOOG         IBM           MSFT
-----------------------------------------------
GOOG 1            8.044209e-05  -0.001496302
IBM  8.044209e-05 1             -0.0001569679
MSFT -0.001496302 -0.0001569679 1
q)\t correlationmatrix[2014.01.09; 2014.01.15; `GOOG`IBM`MSFT; 0D00:00:10]
828
```

This solution executes the quickest and with the least network usage as the resultant correlation matrix returned to the user is small.

5.5 Summary

It can be seen from the above examples that it is most efficient to calculate the correlation matrix directly in kdb+ with the performance gains increasing as the size of the dataset increases. The kdb+ only approach also simplifies the architecture as it uses a single technology. It is up to the user to decide which approach is the best fit for their use-case and expertise, although some amount of initial work should always be done on the kdb+ side to avoid raw data extracts.

6 Example: Working with Smart Meter Data

To demonstrate the power of kdb+, an example using randomly generated smart meter data has been developed. This can be downloaded from

`http://code.kx.com/wsvn/code/contrib/jpress/tutorial`

By following the instructions in the README, an example database can be built. The default database contains information on 100,000 smart meter customers from different sectors and in different regions over 61 days. The default database contains 9.6m records per day, 586m rows in total. A set of example queries are provided, and a tutorial to step through the queries and test the performance of kdb+. Users are encouraged to experiment with:

- using slaves to boost performance
- running queries with different parameters
- modifying or writing their own queries
- compression to reduce the size of on-disk data
- changing the amount of data generated- more days, more customers, different customer distributions etc.

The data can be extracted from R for further analysis or visualisation. As an example, the code below will generate an average daily usage profile for each customer type (res = residential, com = commercial, ind = industrial) over a 10 day period.

```
# load the xtsExtra package
# this will overwrite some of the implementations
# loaded from the xts package (if already loaded)
> install.packages("xtsExtra", repos="http://r-forge.r-project.org")
> library(xtsExtra)
```

```
# load the connection library
> source("c:/r/qServer.R")
> h <- open_connection("127.0.0.1",9998,NULL)

# pull back the profile data
# customertypeprofiles takes 3 parameters
# [start date; end date; time bucket size]
> d<-execute(h,"customertypeprofiles[2013.08.01;2013.08.10;15]")
> dxts<-xts(d[,-1],order.by=d['time'])

# plot it
> plot.xts(dxts, screens=1, ylim=c(0,500000), auto.legend=TRUE, main="
    Usage Profile by Customer Type")
```

which produces the plot in Figure 5:

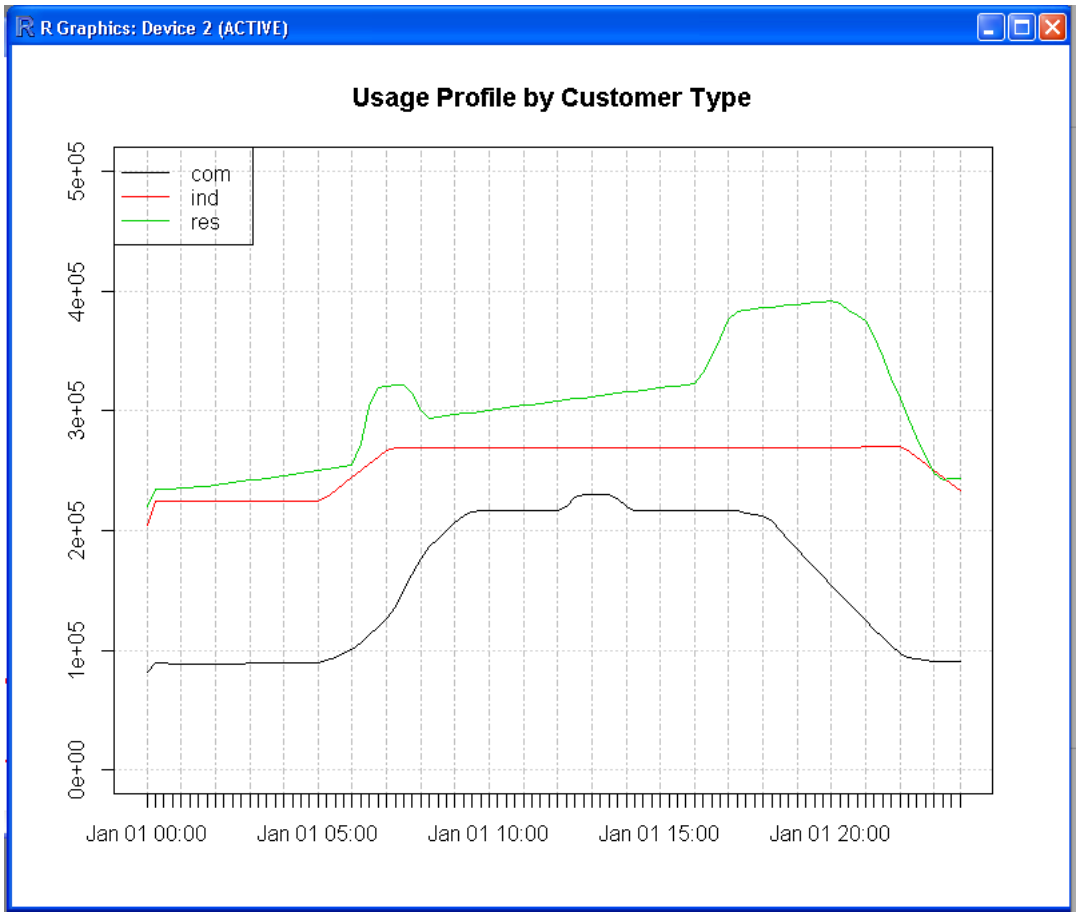

Figure 5: Customer usage profiles generated in kdb+ and drawn in R