Startingkdbplus/rdb

6.1 Overview

A real-time database (rdb) stores today's data. Typically, it would be stored in memory during the day, and written out to the historical database (hdb) at end of day. Storing the rdb in memory results in extremely fast update and query performance.

As a minimum, it is recommended to have RAM of at least 4 times expected data size, so for 5 GB data per day, the rdb machine should have at least 20 GB RAM. In practice, much larger RAM might be used.

6.2 Data Feeds

Data feeds can be any market or other time series data. A feedhandler converts the data stream into a format suitable for writing to kdb+. These are usually written in a compiled language, such as c or c++.

In the example described here, the data feed is generated at random by a q process.

6.3 Tickerplant

The data feed could be written directly to the rdb. More often, it is written to a q process called a tickerplant, which may run several actions whenever data is received, for example:

- write all incoming records to a log file
- push all data to the rdb
- push all or subsets of the data to other processes
- run any other q code that should be executed as new data arrives

Other processes would subscribe to a tickerplant to receive new data, and each would specify what data should be sent (all or a selection).

The kdb+tick [1] product from Kx is a tickerplant that is recommended for production systems with large volumes of real time data.

6.4 Example

The scripts in start/tick [2] run a simple tickerplant/rdb configuration. Note that they are not suitable for production use (no logging, error handling, end of day roll over etc).

The layout is:

```
               feed
                 |
           tickerplant
    /     /     |     \     \    \
 rdb   vwap  hlcv   tq    last  show
  /\   /\     /\    /\     /\
    ... client applications ...
```

Here:

feed is a demo feedhandler, that generates random trades and quotes and sends them to the tickerplant. In practice, this would be replaced by real feedhandlers.

The *tickerplant* gets data from feed and pushes it to clients that have subscribed. Once the data is written, it is discarded.

The *rdb*, *vwap*, *hlcv*, *tq* and *last* processes are databases that have subscribed to the tickerplant. Note that these databases can be queried by a client application.

- rdb has all of today's data
- vwap has volume weighted averages for selected stocks
- hlcv has high, low, close, volume for selected stocks
- tq has a trade and quote table for selected stocks. Each row is a trade joined with the most recent quote.
- last has the last entries for each stock in the trade and quote tables

The *show* process displays the incoming feed for selected stocks.

Note that all the client processes load the same script file cx.q, with a parameter that selects the corresponding code for the process in that file. Alternatively, each process could load its own script file, but since the definitions tend to be very short, it is convenient to use a single script for all. See c.q [3] for more examples (written for kdb+tick).

6.5 Running the Demo

The start/tick [2] scripts run the demo, which should display each q process in a separate window. If necessary, update for the actual directories used.

In Windows, call start/tick/run.bat. In !Linux/Gnome, call start/tick/run.sh. In any other system, either modify the scripts for your environment or start the processes manually, see next section.

The calls starting each process are essentially:

1. tickerplant - the parameter ticker.q is the script defining the tickerplant, and the port is 5010:

```
..$ q ticker.q -p 5010
```

2. feed - connects to the tickerplant and sends a new batch every 107 milliseconds:

```
..$ q feed.q localhost:5010 -t 107
```

3. rdb - the parameter cx.q defines the realtime database and its own listening port (similarly for other databases):

```
..$ q cx.q rdb -p 5011
```

4. show - the show process, which does not need a port:

```
..$ q cx.q show
```

6.6 Running Processes Manually

If the run scripts are unsuitable for your system, then you can call each process manually. In each case, open up a new terminal window, change to the q directory and enter the appropriate command. The tickerplant should be started first.

For example on a Mac, for each of the following commands, open a new terminal, change to ~/q, then call the command:

```
m32/q start/tick/ticker.q -p 5010

m32/q start/tick/feed.q localhost:5010 -t 107

m32/q start/tick/cx.q rdb -p 5011
```

Refer to run1.sh for the remaining processes.

6.7 Process Examples

Set focus on the `last` window, and view the trade table. Note that each time the table is viewed, it will be updated with the latest data:

```
q)trade
sym | time         price size stop cond ex
----| ------------------------------------
AIG | 14:26:48.844 27.62 18   0    Z    O
DELL| 14:26:49.058 11.83 57   0    K    N
DOW | 14:26:49.058 19.81 69   1    G    O
...
```

Set focus on the `vwap` window, and view the vwap table. Note that the "price" is actually price*size. This can be updated much more efficiently than storing all prices and sizes:

```
q)vwap
sym | price    size
----| -------------
IBM | 42153.14 998
MSFT| 51620.66 717
```

To get the correct weighted average price:

```
q)select price%size,size by sym from vwap
sym | price    size
----| --------------
IBM | 41.74374 31824
MSFT| 73.38304 28612
```

Prev: 5. Historical Database

Table of Contents

References

[1] http://kx.com/kdb+tick.php
[2] http://code.kx.com/wsvn/code/contrib/cburke/start/tick
[3] http://kx.com/q/tick/c.q

Article Sources and Contributors

Startingkdbplus/rdb *Source*: http://code.kx.com/mediawiki/index.php?oldid=2285 *Contributors*: Chris Burke

License