it's about time

Developer Brief

Transaction Cost Analysis Using kdb+

Author:

Colm Earley, who joined First Derivatives in 2006, has worked on kdb+ tick database and algorithmic trading systems related to both the equity and FX markets. Based in New York, Colm is a senior kdb+ consultant at a leading brokerage firm, implementing a global tick database.

Table of Contents

1 INTRODUCTION

With the ever-increasing volatility of financial markets and multitude of trading venues as a result of market fragmentation, transaction cost analysis (TCA) has become one of the expected functions provided to a client by their broker dealer. The buy side has also become more sophisticated in recent years in acquiring their own TCA tools as they seek more transparency around their trading. These firms now strive to measure their real execution performance, feeding statistics into their portfolio strategies, market impact models and, of course, relaying their level of service satisfaction back to their brokers.

One attribute that is common across many TCA functions is the need to align market data with trade execution data and summarize the differences between perceived versus actual trade prices. kdb+ enables users to run joins and update benchmarks on a tick by tick basis intra day and scan hundreds of billions of records on disk in seconds. The two use cases synonymous with TCA are:

- Real time and historical slippage calculations
- Market impact profiling

Slippage, at its most basic, is a measure of the difference between the price at the time of your decision and the price actually paid at execution, i.e. the fill message reported by the broker. Slippage can be influenced by many things such as commissions, settlement costs and latency between venues and market impact.

Market impact testing not only enables traders and brokers to refine their execution strategies to ensure they get as good a price as possible, it is also an area where the business can develop algorithms that use backtested leading indicators to predict when a market impact event is going to take place, which in turn provides trading opportunities.

kdb+ has been deployed in both sell side and buy side firms to serve as the data and analysis backbone of TCA environments. These systems capture, process and store increasing amounts of tick and transaction data in intra day processes and historical partitions on disk. kdb+ as a technology has several characteristics including a rich set of time series primitives, which make data retrieval extremely fast and efficient.

This whitepaper, focusing primarily on post trade analysis, details a number of the most commonly used queries and the q language functions that relate to them. Methods are described for performing analysis efficiently across many terabytes of data. This is followed by an overview of a database design and tips on data access with query performance in mind. Reference is made on how to apply the key points learned to common use cases. The paper concludes with a brief overview of some general optimization techniques.

2 POST-TRADE ANALYSIS

One of the most powerful use cases for kdb+ is transaction cost analysis. kdb+ makes it both simple and efficient to sift through tens of terabytes of data. For this discussion, I am going to categorize queries into one of the following two types: i) point-in-time ii) interval.

2.1 POINT IN TIME

The point in time query is commonly used by analysts to compare performance against a benchmark at a specific point in time during the trading day. For TCA this may be used for comparing slippage between the decision price/price in the market to the effective execution price (trade price minus trading costs). This is also known as implementation shortfall.

A powerful built-in function in q makes this point in time query a trivial operation - the aj or asof join is described here: [code.kx.com/aj]. It takes three arguments a) the columns to join on b) a reference table and c) the table we wish to search through. An equivalency check is performed on the initial columns and a binary search on the last. The normal expectation is to join on some market environment data immediately preceding an event, thus returning the prevailing value, as a result that is what is carried out here.

Two possible ways of measuring implementation shortfall are to compare a given set of internal execution data against either the prevailing mid-price or last trade price.

```
/ sample of 10 quotes
q)q:`time xasc ([]sym:10#`FDP; time:09:30t+00:30t*til 10; bid:100.+0.01*til 10;
ask:101.+0.01*til 10)
q)q
sym time         bid    ask
------------------------------
FDP 09:30:00.000 100    101
FDP 10:00:00.000 100.01 101.01
FDP 10:30:00.000 100.02 101.02
FDP 11:00:00.000 100.03 101.03
FDP 11:30:00.000 100.04 101.04
FDP 12:00:00.000 100.05 101.05
FDP 12:30:00.000 100.06 101.06
FDP 13:00:00.000 100.07 101.07
FDP 13:30:00.000 100.08 101.08
FDP 14:00:00.000 100.09 101.09

/ sample execution data
q)e:([]sym:1#`FDP;time:1#11:20t;exprice:1#100.55;exsize:1#200)
q)e
sym time         exprice exsize
------------------------------
FDP 11:20:00.000 100.55   200

/ comparing against market mid
q)update mid:0.5*bid+ask from aj[`sym`time;e;q]
```

```
sym time         price  size bid     ask     mid
-------------------------------------------------
FDP 11:20:00.000 100.55 200  100.03  101.03  100.53

/ similarly comparing against last price
/ table of trades
q)t:`time xasc ([]sym:10#`FDP; time:09:30t+00:30t*til 10; price:100.+0.01*til 10;
size:10#100)

/ comparing against last price
q)aj[`sym`time;e;t]
sym time         exprice  exsize price   size
-----------------------------------------
FDP 11:20:00.000 100.55   200    100.03  100
```

Depending on whether your table is in-memory or on disk, correct attribute usage should be employed. More detailed information on this can be found here: http://code.kx.com/wiki/Reference/aj#Remarks

2.2 INTERVAL

During a trading day, a common need is to analyze the market behavior over a set time interval. Interval use cases include comparing our execution price to the market price range during the interval, our vwap to the market vwap, our trading volume as a percentage of market participation. kdb+ provides another tool in its arsenal to retrieve these types of data both effectively and efficiently – the wj or window join is described here: [code.kx.com/wj]. It allows the user to perform an arbitrary number of functions across a time interval.

The following example details two 5,000 share orders. For each, they have both been worked over five minute periods, in five batches of 1000 share executions. The difference between them is the time of day at which they traded. The objective is to return volatility metrics, such as market trade price range, over each order.

```
/ sample market trades
q)nt:100000

q)t:`time xasc
([]sym:nt#`FDP;time:09:30t+nt?06:30t;price:100+(nt?100)%100;size:100*1+nt?10)
/ in-memory table should have both grouped and sorted attributes
q)update `g#sym from `t
`t
q)t
sym time         price  size
---------------------------
FDP 09:30:00.099 100.25 100
FDP 09:30:00.320 100.24 900
FDP 09:30:00.333 100.18 900
..


Continued overleaf
```

```
/ sample execution data
/ exprice – execution price
/ exsize  - execution size
/ orderid - order id, the parent order, of which, the individual
execution belongs to
q)e:`time xasc ([]sym:10#`FDP;time:raze 10:00
15:00t+\:00:01t*til[5];exprice:100+(10?100)%100;exsize:10#1000;orderid:(5#0),
5#1)
q)e
sym time         exprice exsize orderid
-------------------------------------
FDP 10:00:00.000 100.25  1000   0
FDP 10:01:00.000 100.3   1000   0
FDP 10:02:00.000 100.18  1000   0
FDP 10:03:00.000 100.08  1000   0
FDP 10:04:00.000 100.47  1000   0
FDP 15:00:00.000 100.49  1000   1
FDP 15:01:00.000 100.92  1000   1
FDP 15:02:00.000 100.98  1000   1
FDP 15:03:00.000 100.75  1000   1
FDP 15:04:00.000 100.42  1000   1

/ vwap for each order
q)summary:0!select (first[time];last[time]),vwap:exsize wavg exprice by
sym,orderid from e
q)summary
sym orderid time                        vwap
---------------------------------------------
FDP 0       10:00:00.000 10:04:00.000 100.256
FDP 1       15:00:00.000 15:04:00.000 100.712


/ market volume and price range over the order lifetimes
q)wj[flip exec time from
summary;`sym`time;summary;(t;(min;`price);(max;`price))]
sym orderid time                        vwap    price price
-----------------------------------------------------------
FDP 0       10:00:00.000 10:04:00.000 100.256 100   100.99
FDP 1       15:00:00.000 15:04:00.000 100.712 100   100.99
```

It should be noted that wj only works on monadic functions. Although, the full dataset can be returned and the results aggregated, this can be inefficient and result in excess memory usage. The example below shows how, for vwap, it is more efficient to use two ajs to generate an intermediate result and subsequently reduce it.

```
/ simulate 10000 orders to join on
q)orders:update orderid:til count i from 10000#0!summary
q)orders
sym orderid time                          vwap
-----------------------------------------------
FDP 0       10:00:00.000 10:04:00.000 100.558
FDP 1       15:00:00.000 15:04:00.000 100.534
Continued overleaf
```

```
FDP 2       10:00:00.000 10:04:00.000 100.558
FDP 3       15:00:00.000 15:04:00.000 100.534
..

/ window join
q)\ts select sym, orderid, time, vwap, marketVwap:wavg'[size;price] from
wj1[flip exec time from orders;`sym`time;orders;(t;(::;`size);(::;`price))]
300 328731424

/ 2 ajs
q)orders1:ungroup orders
q)\ts select first vwap, marketVwap:(last[val]-first val)%(last[volume]-
first[volume]) by sym, orderid from aj[`sym`time;orders1;select sym, time,
val:sums price*size, volume:sums size from t]
41 2918528
```

The above illustrates that for this type of query the aj is an order of magnitude faster and uses two orders of magnitude less space than the window join. This is because the data does not need to be copied for every order record.

3 MARKET DATA FILTERING

3.1 TABLE SCHEMA DESIGN AND QUERY PERFORMANCE

Sub-optimal query construction can be the cause of many bottlenecks. One of the causes of this is trying to include too much computation within a query. Very often, the same logic is used repeatedly. Therefore, we find ourselves carrying out the same calculations across different queries. It is preferable to generalize the logic, move it from the query to the feed-handlers, and amortize the calculation over the whole day.

Below are three hypothetical schemas ranging from tight coupling of market data logic and business requirements to complete isolation. Each has its own pros and cons. Our exchange in this example has four conditions:

A) Bunched Trade
   Multiple trades which are combined together and the average price published. We may wish to exclude these from last queries, because the reported price is not a true trade price.

B) Out of sequence
   These could include late reported trades. We may exclude these from last price queries, as we cannot tell for certain when they occurred.

C) Off-Exchange
   Examples of these are trades which occur on dark pools. A dark pool may wish to estimate their market share with other off-exchange venues.

D) Auction
   These are usually trades which are preceded by a bidding period used to fix the opening and closing prices. We may wish to only include trades within these bounds, as pre- and post-market trades can be too volatile.

We wish to retrieve the vwap over a period and for our purpose A, B and C are not eligible for vwap calculations. Reasons for excluding these from our implementation could include:

A) The Bunched Trade price calculation is not public. It may not be weighted by the size of each constituent trade.

B) The time of the Out of Sequence trade cannot be guaranteed to have occurred during the period under analysis.

C) Similar to B) above, the Off-Exchange trade may have been published late and thus, the time of it may not be trustworthy.

3.1.1 Schema I

Schema I has a character array column for storing trade conditions. This type of schema enables our query implementation to be very flexible. However, the downside is that string comparison can be very slow. In addition, if a new condition is added which affects vwap eligibility, each dependent query must also be updated.

```
/ sample market trade data
q)n:1000000
q)t:`time xasc
([]sym:n#`FDP;time:09:30t+n?06:30t;price:100+(n?100)%100;size:100*1+n?10;con
d:neg[1+n?2]?\:"ABCD")
sym time         price  size cond
--------------------------------
FDP 09:30:00.027 100.72 900  "DC"
FDP 09:30:00.029 100    800  "BA"
FDP 09:30:00.030 100.34 900  "CA"
..

/ string comparison
q)\ts select from t where not any cond like/:("*A*";"*B*";"*C*")
248 5243536
```

3.1.2 Schema II

Schema II has separated out the trade conditions by adding a Boolean flag column for each type of condition. This is very flexible on the implementation of the query. It is also more efficient as we can see from our load test result below. However, with this flexibility it suffers the same drawback as Schema I therefore all dependent queries will need to be changed if there is a condition change. Furthermore, the table schema may need to be amended if a new condition is added. The extra disk space is not an issue if on-disk compression is used as Boolean columns compress very efficiently.

```
/ boolean columns for conditions added
sym time         price  size cond A B C D
----------------------------------------
FDP 09:30:00.027 100.72 900  "DC" 0 0 1 1
FDP 09:30:00.029 100    800  "BA" 1 1 0 0
FDP 09:30:00.030 100.34 900  "CA" 1 0 1 0
..

q)\ts select from t where not A or B or C
16 5374496
```

3.1.3 Schema III

Schema III is a more general form of Schema II above. Boolean flags are added for the general idea of vwap eligibility but not for each condition. The loss in query implementation flexibility is offset against the benefit gained through the introduction of a layer of abstraction between the market data and business logic. Queries can now be developed in a more general form that covers all exchanges, and not every developer needs to be a market data expert. If a trade condition changes the vwap eligibility, a data migration to update all historical dates may be needed, but this should be seamless to the query. An efficient migration can rewrite only the Boolean flag column without having to write all of the data.

Note: Great care should be taken when performing migrations on on-disk databases.

```
/ Compound Boolean Column for vwap eligibility
sym time         price  size cond vwapEl
---------------------------------------
FDP 09:30:00.027 100.72 900  "DC" 0
FDP 09:30:00.029 100    800  "BA" 0
FDP 09:30:00.030 100.34 900  "CA" 0
..

q)\ts select from t where vwapEl
16 5374368
```

3.2 PRE-CALCULATION

Generally, the simplest way to optimize performance is to reduce the dataset being accessed. As trade and quote volumes continue to grow, one should take advantage of any way to avoid searching through a full day of trade or quote data.

One basic example is to use pre-calculated one-minute bars if one-minute granularity is sufficient for your calculation. Below we can see that the native bar retrieval time for this example is more than twice as fast calculating it from the raw trades and quotes at run-time. The data sets for the above examples are 360MM trade and quote records versus 9MM bar records for 37,000 unique symbols.

```
/ Operating System Cache is Cold

/ generating bars from raw trades and quotes
q)\ts aj[`sym`time;select firstTradePrice:first price, volume:sum size by date,
`p#sym, time:`time$time.minute from trade where date=d, sym in `FDP1`FDP2`FDP3`FDP4;
select date, sym, time, openBid:bid, openAsk:ask from quote where date=d]
2691 2155880464j

/ retrieving bars from native bar table
q)\ts select date, sym, bar, firstTradePrice, volume, openBid, openAsk from bar
where date=d, sym in `FDP1`FDP2`FDP3`FDP4
1090 183184j
```

Tools for flushing the Operating System Disk Cache are detailed here:
[http://code.kx.com/wiki/Cookbook/PerformanceTips]

On subsequent runs, when the data is now in the operating system cache, the speed-up is even more pronounced at almost 500x. If this is a calculation, which is repeatedly performed, the argument to pre-calculate is even stronger.

```
/ Data is now in Operating System Cache

/ generating bars from raw trades and quotes
q)\ts aj[`sym`time; select firstTradePrice:first price, volume:sum size by date,
sym, time:`time$time.minute from trade where date=d, sym in `FDP1`FDP2`FDP3`FDP4;
select date, sym, time, openBid:bid, openAsk:ask from quote where date=d]
489 2155880464j

/ retrieving bars from native bar table
q)\ts select date, sym, bar, firstTradePrice, volume, openBid, openAsk from bar
where date=d, sym in `FDP1`FDP2`FDP3`FDP4
1 183184j
```

Taking this a step further, one should also store data, which can be aggregated by day and accessed regularly in a separate table for optimal access. In the example below, only a small number of columns are included, but this could be easily expanded. Other useful daily data may include auction volumes, block trades, trade count and average quote size to name but a few.

```
/ Retrieving Daily from Raw Ticks.
/ openAuc and closeAuc are assumed boolean columns as discussed in 3.1.3
above
q)\ts select open:first price where openAuc,
             openTime:first time where openAuc,
             close:last price where closeAuc,
             closeTime:first time where closeAuc
          by date, sym from trade where date=d, sym in `FDP1`FDP2`FDP3`FDP4
671 5246432j

/ retrieving from dedicated daily table
q)\ts select date, sym, open, openTime, close, closeTime from daily where date=d,
sym in `FDP1`FDP2`FDP3`FDP4
28 2622528j
```

The partitioning structure you choose will depend on your access pattern, number of columns and records. If you are retrieving data for multiple dates at once and the number of records and columns is small plain vanilla splayed table may suffice. Otherwise, partitioning by month or by year should investigated.

A benefit of having the pre-fetched daily data above is the ease with which we can prime dictionary caches; for example, to restrict queries to within market hours. However, it is highly improbable that every symbol will open exactly as the exchange opens. The actual opening times (for some exchanges auction trade times) can vary from symbol to symbol. We can create a symbol and date to bounds cache without having to query the complete trade history:

```
/ Generate Cache
q)dateSymbol2hours:exec (first openTime;first closeTime) by date, symbol from daily

/ Dummy Dictionary to show a Different Mapping is Used
dateSymbol2hours[([]date:3#2013.04.22;sym:`FDP1`FDP2`FDP3)]:(13:30 13:35t;14:30
14:35t;15:30 15:35t)

/ Mapping within a Query
q)select firstTime:min time, firstHoursTime:min time where extime within flip
dr2t[([]date;sym)] by date, sym from rtrade where date=d, sym in `FDP1`FDP2`FDP3
date       sym  | firstTime    firstHoursTime
----------------| --------------------------
2013.04.22 FDP1 | 08:03:42.113 13:30:00.023
2013.04.22 FDP2 | 10:57:30.935 14:30:02.532
2013.04.22 FDP3 | 08:29:58.950 15:30:17.567
```

4 CONCLUSION

This whitepaper provides a brief introduction to performing transaction cost analysis using kdb+. The technology enables both brokers and buy side firms to run analysis on high frequency data that can often overwhelm traditional database and event processing systems. Speedy analytics such as aligning trades with the prevailing best bid&ask, implementation shortfall, market impact profiles and slippage statistics gives traders the ability to minimize opportunity costs as well as develop ideas to take advantage of market trends. The execution and enhancement of trading strategies can then be easily adjusted intra day to reflect occurrences in the market.

Having this ability to capture massive volumes of high and low frequency market, execution and news data not only aids firms in pinpointing direct execution costs, it also provides a mechanism to measure negative aspects of the trade life cycle, which can often be hidden. While this paper does not discuss examples of this use case the analysis of individual process bottlenecks is another simple yet effective use of kdb+ that often goes hand in hand with TCA analysis. At this point a timestamp can be tagged to any messaging system or trading engine to monitor for the publishing of stale data, which is difficult to determine when dealing with microsecond resolution and non-performing data analysis tools.

Later in the paper, the benefits of market data source agnostic schema are discussed. Here there are two main advantages in that such a schema will offer run-time query speed-ups and a shortened development cycle. The built-in functions kdb+ provides are very powerful, but the data must also be stored intelligently so that the queries can access it in an efficient manner.

Overall, the paper provides a snapshot of the type of TCA queries that can be run. It aims to highlight the capabilities of kdb+ to offer a best of breed high performance data analysis layer. When combined with either in house or FD's Delta visualization tools, it provides all the components of a TCA application to ensure that investment strategies can be executed more efficiently.

All tests performed with kdb+ 3.0 (2012.11.12).