Tick Data Processor Performance Testing

|  |  |  |
| --- | --- | --- |
| Date | Document version | Changes |
| 11/01/2020 | 1.0 | Created |
|  |  |  |

Table of Contents

[Objective 2](#_Toc55372697)

[Testing Methodology 2](#_Toc55372698)

[Execution 2](#_Toc55372699)

[Test Data 2](#_Toc55372700)

[Hardware configurations 2](#_Toc55372701)

[Test Scripts 3](#_Toc55372702)

[Preparing Test Dataset 3](#_Toc55372703)

[Test Execution 3](#_Toc55372704)

[Test Driver 3](#_Toc55372705)

[Generating Reports 4](#_Toc55372706)

[Resting Order Duration 4](#_Toc55372707)

[Test Execution 4](#_Toc55372708)

[Generating Reports 4](#_Toc55372709)

[Test Results Analysis 4](#_Toc55372710)

[Conclusion 5](#_Toc55372711)

[Appendix A. Test Results for VWAP and NBBOPrice 6](#_Toc55372712)

[Appendix B. Resting Order Duration Test Results 8](#_Toc55372713)

# Objective

Performance testing of Tick Data Processor (TDP) has following goals:

* generate statistics by executing stress tests under various loads using multiple combinations of requested functions
* determine performance boundaries to enable better h/w management
* determine performance bottlenecks, and identify possible ways to improve query speed
* find optimal query size and h/w configuration for key use cases

# Testing Methodology

Execution   
Stress test is performed using two common TCA functions: NBBO price lookup and VWAP calculation.  
Multiple test cases are used for testing, each one representing different load scenario.  
Test cases differ in request size, number of concurrent clients, and combination of functions with or without markouts.  
Each test consists of series of sub-tests or runs, executed sequentially. Run invokes configurable number of client processes executing simultaneously on dedicated host. Each sub-process is given unique date.  
GNU parallel utility is used to control execution.  
Test result summary is produced after completion of all test cases. Summary includes test's key attributes, average, best and worst query response time, and breakdown of 3 distinct execution phases within server process: a) input parsing and sorting, b) query execution and c) output sorting and merging.

Test Data  
Tests use one month of tick data. The data is taken from 2020-03-31 NYSE TAQ files, and copied to other 21 dates of March 2020.  
To create stress test conditions 20 most liquid ticker symbols are selected from each symbol group - 520 in total. Daily trade count is used as a gauge. Selected set of symbols is responsible for approximately 75% all transactions recorded in CTA  
Reference times (quote for NBBO and start time for VWAP) are randomly generated within range between 9:30 and 16:00.  
Duration for VWAP queries are randomly generated with values between 1 second and 4 hours, with end time capped by 16:00.  
By design each test run prompts server to examine around 75% of all trades and/or NBBO changes of entire month of 2020-03.  
Markout durations used in test include several tick- and time-based values in both directions e.g.

* 0.5h - is positive time value of half of an hour, which results in VWAP calculation over period from start time until start time + 30min
* -10t - negative tick count, which results in VWAP calculation over period defined by ticks ending with current one (at start time) and starting on 10th tick prior current

Hardware configurations  
server: VM host with 16 CPU cores and 64GB RAM  
client: VM host with 4 CPU cores and 64GB RAM

Test Scripts  
Following scripts were developed to facilitate performance testing:

|  |  |
| --- | --- |
| Script Name | Description |
| taq-proc-perf-prep.py | Python script to generate base record set of given size |
| taq-proc-perf-exec.py | Python script to execute test It uses base records set to generate input data and execute test |
| perf-test-case.lst | Text file which defines all test cases |
| perf-test.sh | Shell script to execute all tests defined in perf-test-case.lst |
| taq-proc-perf-report.py | Python script to create test result summary using output from perf-test.sh |

## Preparing Test Dataset

Test data set is created using Python script *taq-proc-perf-prep.py*. Given list of symbols and test size the script generates test data and saves it as a dataframe in output file in HDF format. This file is subsequently used during test executions.  
Symbol list is passed in a file. Each entry consists of ticker symbol and corresponding reference price, which is average execution price on March 31, 2020. To achieve meaningful query outcomes the reference price is used for assigning limit price for VWAP requests during data generation.

## Test Execution

Every test is executed with help of Python script *taq-proc-perf-exec.py*. Script requires following arguments:

* HDF file to use as base; implicitly defines test size
* transaction date
* function list: combination of VWAP and NBBOPrice
* markouts size: integer in 0 to 8 range

The script creates actual record set for service request using data sourced from HDF file. Python extension module taqpy is used as API.

## Test Driver

Entire series of tests is executed using bash script *perf-test.sh*. The script reads test definition from configuration file *perf-test-case.lst*. Each line in the file defines a specific test case, and consists of space delimited fields in following format

|  |  |
| --- | --- |
| Position | Description |
| 1 | Test name or ID |
| 2 | Number of runs |
| 3 | Numb er of client processes executing in parallel |
| 4 | HDF file name |
| 5 | Markouts size |
| 6 | List of functions, comma-separated |

Example of test cases definition file

1 10 10 perf-test-10k.hdf m0 NBBOPrice  
2 10 10 perf-test-100k.hdf.hdf m0 NBBOPrice  
3 10 10 perf-test-500k.hdf m0 NBBOPrice  
4 10 10 perf-test-1m.hdf m0 NBBOPrice. . .   
57 10 10 perf-test-2m.hdf m0 NBBOPrice,VWAP  
58 10 10 perf-test-2m.hdf m6 NBBOPrice,VWAP  
59 10 4 perf-test-2m.hdf m0 NBBOPrice,VWAP  
60 10 4 perf-test-2m.hdf m6 NBBOPrice,VWAP

## Generating Reports

Python script *taq-proc-perf-report.py* is used to produce test summary report. Required input for this script is the output from the test driver *perf-test.sh*, which can be passed as file or read from stdin.  
The final report is presented as csv file, which can be used 'as is', or loaded into database or Excel for further processing.

# Resting Order Duration

Function ROD calculates resting order duration, which is Astral-specific requirement. It was implemented to address drawbacks in original Factsheet data enrichment process.  
Stress testing of ROD function differs is performed using two scripts:

|  |  |
| --- | --- |
| Script Name | Description |
| test-rod.py | Python script to build and execute single request using transactional data sample taken from 12 Dec, 2019 |
| taq-proc-perf-rod.sh | Test driver bash script to execute all test cases for resting order duration |

## Test Execution

12 Dec 2019 was selected because processing on that date took long time, and therefore the dataset is good benchmark candidate. Test size is 4,173,312 records, which yields 3,854,838 result records, one per child order.

Main execution script *test-rod.py* interacts with the server directly without relying on Python API extension taqpy.

## Generating Reports

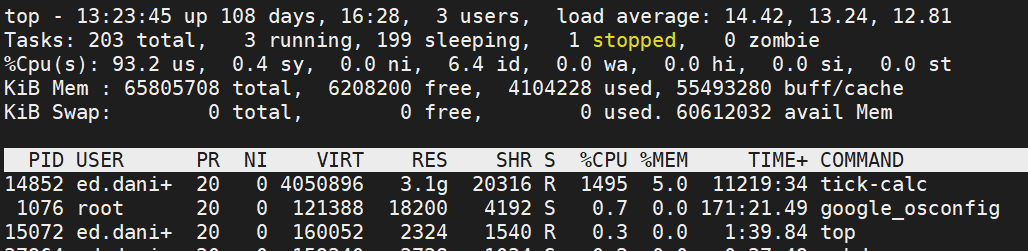
The same Python script *taq-proc-perf-report.py* is used to produce test summary report. Required input for this script is the output from the test driver *taq-proc-perf-rod.sh*, which can be passed as file or read from stdin.

# Test Results Analysis

Time spent within server process is by far the most contributing factor in overall latency. It is broken into 3 phases:   
a) input - parsing and sorting input records  
b) execution - accessing and examining tick data, and making appropriate calculations  
c) output - sorting/merging output into result record set

During performance test following observation were made:

* in smaller tests most of the time spent within server process is attributed execution phase A i.e. 70-90%
* this ratio changes with increase in test size - execution become relatively short i.e. 10-30%, with other two phases dominating
* query complexity significantly influences latency breakdown for larger queries. More complex queries, e.g. with markouts, make last phase C more prevailing by pushing its share to 2/3
* in lager queries with less complexity e.g. single function without markouts, handling of input phase A becomes the longest, taking around 1/2 of overall time
* with increased load on server process, the CPU and memory consumption on the host reaches and stays on maximum possible levels:  
  all allocated CPU cores used at 100%  
  almost all physical memory is allocated for buffering of tick data (see buff/cache + free on picture)



## Conclusion

Substantial performance boost can be quickly achieved by using larger capacity VM to host server process. Expected performance gains can be proportional to increase in h/w capacity.

If investing in further performance improvement is necessary and justified, then the focus should probably be on optimizing input/output phases. Reducing latency can be achieved in several ways  
- using binary data representation for data exchange between client and server  
- optimizing memory allocations i.e. use fewer memory allocations when reading input by pre-allocating all required memory in one operation  
- processing phases A and C in thread pool rather than in main thread

# Appendix A. Test Results for VWAP and NBBOPrice

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Test | Test Cnt. | Parallel Cnt. | Test Size | Markouts | Function List | Avg. Time | Min. Time | Max. Time | TDP input proc. | TDP exec. | TDP output proc. |
| 1 | 10 | 10 | 10K | m0 | NBBOPrice | 0:00:00.251825 | 0:00:00.164021 | 0:00:00.775925 | 31% | 57% | 7% |
| 2 | 10 | 10 | 100K | m0 | NBBOPrice | 0:00:05.681998 | 0:00:01.286986 | 0:00:32.212192 | 30% | 41% | 26% |
| 3 | 10 | 10 | 500K | m0 | NBBOPrice | 0:00:27.235716 | 0:00:07.829877 | 0:00:34.886107 | 43% | 14% | 43% |
| 4 | 10 | 10 | 1M | m0 | NBBOPrice | 0:00:57.253505 | 0:00:14.119690 | 0:01:12.290494 | 42% | 14% | 39% |
| 5 | 10 | 10 | 10K | m6 | NBBOPrice | 0:00:00.887313 | 0:00:00.497150 | 0:00:03.337930 | 11% | 68% | 12% |
| 6 | 10 | 10 | 100K | m6 | NBBOPrice | 0:00:17.929581 | 0:00:07.299611 | 0:00:35.585457 | 27% | 42% | 29% |
| 7 | 10 | 10 | 500K | m6 | NBBOPrice | 0:01:32.258367 | 0:00:31.325081 | 0:02:13.354794 | 19% | 16% | 64% |
| 8 | 10 | 10 | 1M | m6 | NBBOPrice | 0:03:06.314467 | 0:01:01.253642 | 0:04:21.369130 | 19% | 16% | 65% |
| 9 | 10 | 4 | 10K | m0 | NBBOPrice | 0:00:00.178133 | 0:00:00.148736 | 0:00:00.511646 | 21% | 68% | 6% |
| 10 | 10 | 4 | 100K | m0 | NBBOPrice | 0:00:01.641495 | 0:00:00.869297 | 0:00:03.045313 | 38% | 37% | 24% |
| 11 | 10 | 4 | 500K | m0 | NBBOPrice | 0:00:10.000160 | 0:00:04.021860 | 0:00:13.991629 | 39% | 26% | 34% |
| 12 | 10 | 4 | 1M | m0 | NBBOPrice | 0:00:20.757604 | 0:00:08.068461 | 0:00:27.673817 | 39% | 25% | 35% |
| 13 | 10 | 4 | 10K | m6 | NBBOPrice | 0:00:00.521858 | 0:00:00.410757 | 0:00:01.351959 | 12% | 71% | 12% |
| 14 | 10 | 4 | 100K | m6 | NBBOPrice | 0:00:06.486828 | 0:00:03.555032 | 0:00:12.227688 | 28% | 46% | 26% |
| 15 | 10 | 4 | 500K | m6 | NBBOPrice | 0:00:35.486284 | 0:00:18.182359 | 0:00:48.180352 | 30% | 38% | 32% |
| 16 | 10 | 4 | 1M | m6 | NBBOPrice | 0:01:12.158542 | 0:00:38.891970 | 0:01:36.912668 | 30% | 38% | 32% |
| 17 | 10 | 10 | 10K | m0 | VWAP | 0:00:01.687315 | 0:00:00.491581 | 0:00:06.890286 | 6% | 88% | 1% |
| 18 | 10 | 10 | 100K | m0 | VWAP | 0:00:08.698418 | 0:00:05.417003 | 0:00:26.939408 | 12% | 83% | 5% |
| 19 | 10 | 10 | 500K | m0 | VWAP | 0:00:36.032648 | 0:00:21.521894 | 0:00:51.983134 | 53% | 20% | 26% |
| 20 | 10 | 10 | 1M | m0 | VWAP | 0:01:12.588098 | 0:00:39.819087 | 0:01:46.700608 | 53% | 20% | 26% |
| 21 | 10 | 10 | 10K | m6 | VWAP | 0:00:01.109036 | 0:00:00.502392 | 0:00:03.210578 | 11% | 82% | 3% |
| 22 | 10 | 10 | 100K | m6 | VWAP | 0:00:06.294491 | 0:00:02.604250 | 0:00:15.167195 | 44% | 25% | 23% |
| 23 | 10 | 10 | 500K | m6 | VWAP | 0:00:32.463334 | 0:00:13.022624 | 0:00:54.670577 | 43% | 14% | 35% |
| 24 | 10 | 10 | 1M | m6 | VWAP | 0:01:09.017345 | 0:00:31.326733 | 0:01:50.847335 | 41% | 12% | 43% |
| 25 | 10 | 4 | 10K | m0 | VWAP | 0:00:00.494341 | 0:00:00.390604 | 0:00:01.519613 | 12% | 84% | 2% |
| 26 | 10 | 4 | 100K | m0 | VWAP | 0:00:01.989745 | 0:00:01.521653 | 0:00:04.917830 | 35% | 50% | 13% |
| 27 | 10 | 4 | 500K | m0 | VWAP | 0:00:12.080641 | 0:00:09.229797 | 0:00:24.641882 | 40% | 38% | 20% |
| 28 | 10 | 4 | 1M | m0 | VWAP | 0:00:24.528277 | 0:00:18.336323 | 0:00:48.893834 | 41% | 37% | 21% |
| 29 | 10 | 4 | 10K | m6 | VWAP | 0:00:00.497727 | 0:00:00.413160 | 0:00:01.632642 | 13% | 77% | 5% |
| 30 | 10 | 4 | 100K | m6 | VWAP | 0:00:02.441621 | 0:00:01.449523 | 0:00:04.377863 | 41% | 35% | 20% |
| 31 | 10 | 4 | 500K | m6 | VWAP | 0:00:12.242500 | 0:00:07.903025 | 0:00:22.834961 | 45% | 25% | 25% |
| 32 | 10 | 4 | 1M | m6 | VWAP | 0:00:24.547663 | 0:00:16.076103 | 0:00:44.096981 | 46% | 25% | 27% |
| 33 | 10 | 10 | 10K | m0 | NBBOPrice,VWAP | 0:00:03.387404 | 0:00:00.930864 | 0:00:04.489867 | 3% | 95% | 1% |
| 34 | 10 | 10 | 100K | m0 | NBBOPrice,VWAP | 0:00:13.115288 | 0:00:07.245538 | 0:00:19.795016 | 47% | 18% | 34% |
| 35 | 10 | 10 | 500K | m0 | NBBOPrice,VWAP | 0:01:05.403891 | 0:00:33.833394 | 0:01:38.606188 | 50% | 14% | 36% |
| 36 | 10 | 10 | 1M | m0 | NBBOPrice,VWAP | 0:02:10.297530 | 0:01:06.862449 | 0:03:13.703327 | 51% | 13% | 36% |
| 37 | 10 | 10 | 10K | m6 | NBBOPrice,VWAP | 0:00:05.689400 | 0:00:01.207336 | 0:00:06.829087 | 3% | 96% | 1% |
| 38 | 10 | 10 | 100K | m6 | NBBOPrice,VWAP | 0:00:24.807408 | 0:00:12.839681 | 0:00:36.669604 | 37% | 26% | 27% |
| 39 | 10 | 10 | 500K | m6 | NBBOPrice,VWAP | 0:02:12.763879 | 0:00:44.608883 | 0:03:07.210583 | 21% | 11% | 67% |
| 40 | 10 | 10 | 1M | m6 | NBBOPrice,VWAP | 0:04:31.050145 | 0:01:26.469535 | 0:06:08.623191 | 22% | 12% | 66% |
| 41 | 10 | 4 | 10K | m0 | NBBOPrice,VWAP | 0:00:00.763058 | 0:00:00.523668 | 0:00:02.333632 | 12% | 84% | 3% |
| 42 | 10 | 4 | 100K | m0 | NBBOPrice,VWAP | 0:00:04.346469 | 0:00:03.104058 | 0:00:07.773237 | 33% | 31% | 23% |
| 43 | 10 | 4 | 500K | m0 | NBBOPrice,VWAP | 0:00:23.575264 | 0:00:17.271621 | 0:00:40.206706 | 41% | 29% | 29% |
| 44 | 10 | 4 | 1M | m0 | NBBOPrice,VWAP | 0:00:47.514262 | 0:00:32.030827 | 0:01:18.948663 | 42% | 28% | 30% |
| 45 | 10 | 4 | 10K | m6 | NBBOPrice,VWAP | 0:00:01.658571 | 0:00:01.011821 | 0:00:02.935931 | 8% | 88% | 4% |
| 46 | 10 | 4 | 100K | m6 | NBBOPrice,VWAP | 0:00:09.786152 | 0:00:04.971695 | 0:00:17.554926 | 32% | 40% | 25% |
| 47 | 10 | 4 | 500K | m6 | NBBOPrice,VWAP | 0:00:49.900742 | 0:00:27.732019 | 0:01:23.383989 | 34% | 33% | 33% |
| 48 | 10 | 4 | 1M | m6 | NBBOPrice,VWAP | 0:01:42.223194 | 0:01:02.720799 | 0:02:35.174213 | 34% | 32% | 33% |
| 49 | 10 | 10 | 2M | m0 | NBBOPrice | 0:01:46.003544 | 0:00:27.292526 | 0:02:20.844946 | 40% | 15% | 44% |
| 50 | 10 | 10 | 2M | m6 | NBBOPrice | 0:06:07.617493 | 0:01:59.909224 | 0:08:49.356176 | 19% | 15% | 66% |
| 51 | 10 | 4 | 2M | m0 | NBBOPrice | 0:00:40.522792 | 0:00:17.585812 | 0:00:55.888816 | 38% | 25% | 36% |
| 52 | 10 | 4 | 2M | m6 | NBBOPrice | 0:02:21.365294 | 0:01:18.562753 | 0:03:20.156287 | 30% | 37% | 33% |
| 53 | 10 | 10 | 2M | m0 | VWAP | 0:02:23.814065 | 0:01:19.676078 | 0:03:33.916004 | 53% | 20% | 27% |
| 54 | 10 | 10 | 2M | m6 | VWAP | 0:02:27.207085 | 0:01:16.503337 | 0:03:37.581476 | 38% | 10% | 51% |
| 55 | 10 | 4 | 2M | m0 | VWAP | 0:00:49.971646 | 0:00:39.277588 | 0:01:34.809492 | 42% | 36% | 21% |
| 56 | 10 | 4 | 2M | m6 | VWAP | 0:00:49.868397 | 0:00:26.471039 | 0:01:29.974748 | 45% | 25% | 29% |
| 57 | 10 | 10 | 2M | m0 | NBBOPrice,VWAP | 0:04:18.415791 | 0:02:12.569059 | 0:06:30.877146 | 51% | 13% | 36% |
| 58 | 10 | 10 | 2M | m6 | NBBOPrice,VWAP | 0:09:10.988219 | 0:02:56.253460 | 0:13:16.922796 | 22% | 11% | 67% |
| 59 | 10 | 4 | 2M | m0 | NBBOPrice,VWAP | 0:01:34.992871 | 0:01:01.310295 | 0:02:36.318436 | 42% | 27% | 31% |
| 60 | 10 | 4 | 2M | m6 | NBBOPrice,VWAP | 0:03:23.181974 | 0:02:06.369382 | 0:05:56.995950 | 34% | 31% | 34% |

# Appendix B. Resting Order Duration Test Results

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Test Name | Test Cnt. | Parallel Cnt. | Avg. Time | Min. Time | Max. Time | TDP input proc. | TDP exec. | TDP output proc. |
| single | 10 | 1 | 0:00:47.659973 | 0:00:40.192953 | 0:00:49.252440 | 71% | 8% | 6% |
| parallel-4 | 10 | 4 | 0:02:37.424089 | 0:01:39.140013 | 0:03:03.696961 | 73% | 4% | 18% |
| parallel-10 | 10 | 10 | 0:02:52.704477 | 0:02:28.897155 | 0:03:05.102121 | 72% | 3% | 20% |