TimeStored

Python + KDB
Tuesday 13th August - 2:00 - 2:45 PM BST

Two Presentations:

- **Kola -** The Fastest kdb+ Python API
  Jo Shinonome

- **PythonDB -** The most Powerful Database?
  Ryan Hamilton

Introduction to kola

the “fastest” Python interface to kdb+

Jo Shinonome

Self Introduction

Author of

- vscode-q, vscode-k-pro, the vscode plugin for kdb+/q
- jkdb, a high performance and modern Javascript interface to kdb+/q
- geek, a golang interface to kdb+/q
- kola, a Python/Rust/R Polars interface to kdb+/q

Python interfaces to kdb+

- qPython/qPython3, Cython
- pyq, C, deprecated
- pykx, Cython + kdb+/q process wrapper
- kola, Rust

qPython/qPython3 - Cython(1%)

- only use Cython for uncompressing IPC message
  https://github.com/finos/qPython/blob/main/qpython/fastutils.pyx
- deserializing IPC messages in Python, low-performance
- kola is 10-20 times faster than qPython/qPython3

pyq - C(43%)

- allow to run python code in kdb, and run q code in python
- for most cases, q objects cannot be used directly by Python packages
- such projects are too complicated to maintain
  - python code in q cannot be linted and formatted
  - q code in Python is not necessary, use qStudio or vscode-k-pro

pykx - Cython(9%), C(4%), q(4%)

- a kdb+/q process wrapper in Python
- store q objects in q process
- provide dataframe interface to q table
  - need to convert to pandas/pyarrow for some Python ML packages, low performance
  - requires Python developers to learn some q knowledge to use the interface
- expensive license
- set up requires several dependencies to be installed for Windows
- start up questions are quite annoying, keep asking for the license file
- Cython code base is difficult to maintain, no proper IDE for Cython

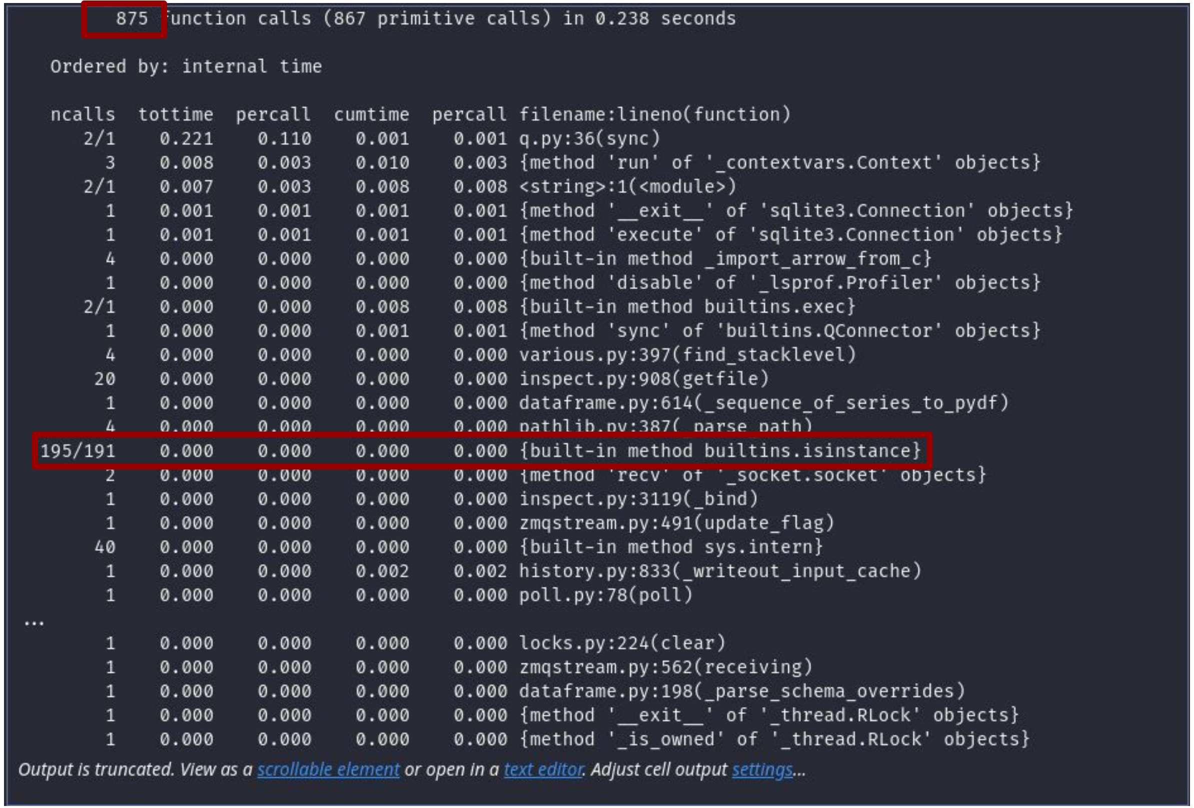
kola - Rust(84%)

- core parts (uncompression and deserialization) are in Rust (84% code)
  - c level performance
  - better memory management, 30%-50% less memory when querying data from kdb https://github.com/jshinonome/kola/blob/main/py-kola/benchmark.md
  - better deserialization performance using parallel computing
- support Python 3.12 without changing code
- a much bigger polars community to support dataframe interface
  - most machine learning packages are going to support polars directly
  - better performance for converting data to numpy/pandas
- no need to know kdb+/q knowledge for Python developers

Dataframe - pykx vs pandas vs polars

- all provide dataframe for Python
  - pykx - kdb table backend
  - pandas - numpy/pyarrow backend
  - polars - pyarrow backend
- polars is between 10 and 100 times as fast as pandas for df operations
- polars has the same level of performance as or even faster than kdb+
- polars can be used for almost all pykx dataframe operations
- inequality join for polars, correspondent to pykx window join, is in progress
- pandas is the most supported dataframe for Python ML packages
- Python ML packages support for polars is a work in progress

Profiling - Num of Function Calls for Sync - kola



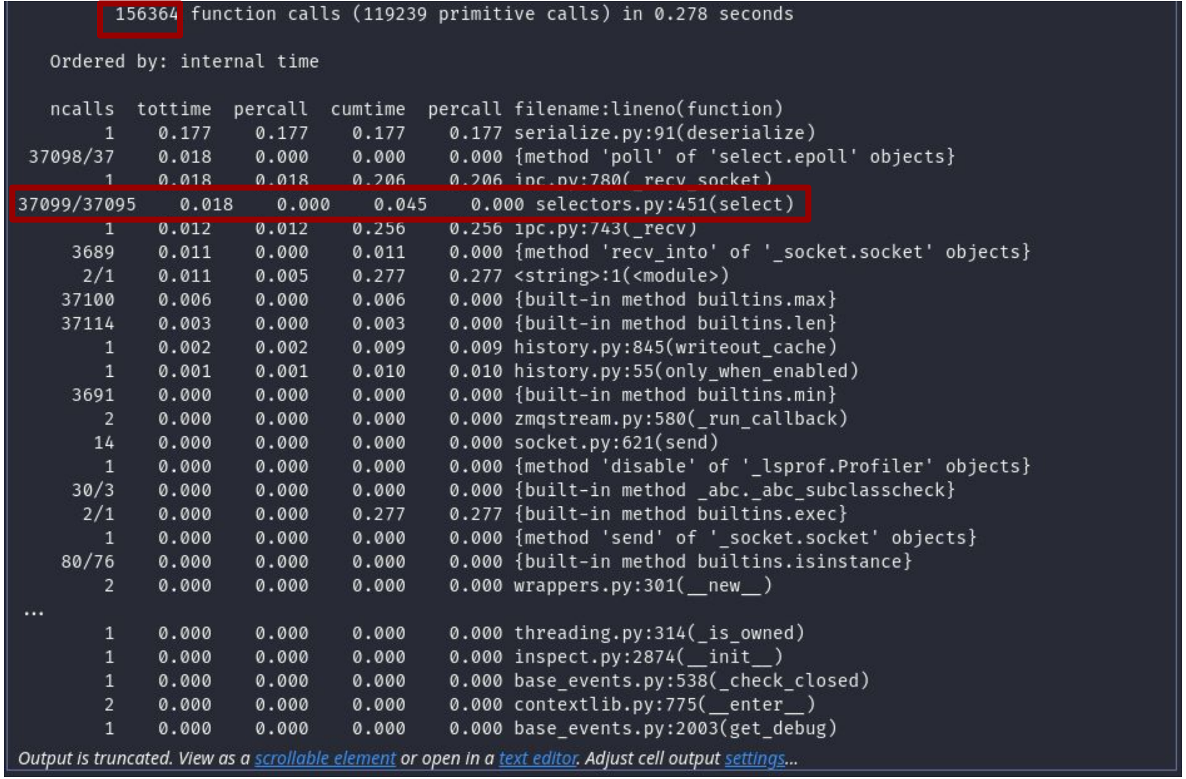
```
         875 function calls (867 primitive calls) in 0.238 seconds

   Ordered by: internal time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
      2/1    0.221    0.110    0.001    0.001 q.py:36(sync)
        3    0.008    0.003    0.010    0.003 {method 'run' of '_contextvars.Context' objects}
      2/1    0.007    0.003    0.008    0.008 <string>:1(<module>)
        1    0.001    0.001    0.001    0.001 {method '__exit__' of 'sqlite3.Connection' objects}
        1    0.001    0.001    0.001    0.001 {method 'execute' of 'sqlite3.Connection' objects}
        4    0.000    0.000    0.000    0.000 {built-in method _import_arrow_from_c}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
      2/1    0.000    0.000    0.008    0.008 {built-in method builtins.exec}
        1    0.000    0.000    0.001    0.001 {method 'sync' of 'builtins.QConnector' objects}
        4    0.000    0.000    0.000    0.000 various.py:397(find_stacklevel)
       20    0.000    0.000    0.000    0.000 inspect.py:908(getfile)
        1    0.000    0.000    0.000    0.000 dataframe.py:614(_sequence_of_series_to_pydf)
        4    0.000    0.000    0.000    0.000 pathlib.py:387(_parse_path)
  195/191    0.000    0.000    0.000    0.000 {built-in method builtins.isinstance}
        2    0.000    0.000    0.000    0.000 {method 'recv' of '_socket.socket' objects}
        1    0.000    0.000    0.000    0.000 inspect.py:3119(_bind)
        1    0.000    0.000    0.000    0.000 zmqstream.py:491(update_flag)
       40    0.000    0.000    0.000    0.000 {built-in method sys.intern}
        1    0.000    0.000    0.002    0.002 history.py:833(_writeout_input_cache)
        1    0.000    0.000    0.000    0.000 poll.py:78(poll)
...
        1    0.000    0.000    0.000    0.000 locks.py:224(clear)
        1    0.000    0.000    0.000    0.000 zmqstream.py:562(receiving)
        1    0.000    0.000    0.000    0.000 dataframe.py:198(_parse_schema_overrides)
        1    0.000    0.000    0.000    0.000 {method '__exit__' of '_thread.RLock' objects}
        1    0.000    0.000    0.000    0.000 {method '_is_owned' of '_thread.RLock' objects}
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

Profiling - Num of Function Calls for Sync - pykx



```
         156364 function calls (119239 primitive calls) in 0.278 seconds

   Ordered by: internal time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.177    0.177    0.177    0.177 serialize.py:91(deserialize)
 37098/37    0.018    0.000    0.000    0.000 {method 'poll' of 'select.epoll' objects}
        1    0.018    0.018    0.206    0.206 ipc.py:780(_recv_socket)
37099/37095    0.018    0.000    0.045    0.000 selectors.py:451(select)
        1    0.012    0.012    0.256    0.256 ipc.py:743(_recv)
     3689    0.011    0.000    0.011    0.000 {method 'recv_into' of '_socket.socket' objects}
      2/1    0.011    0.005    0.277    0.277 <string>:1(<module>)
    37100    0.006    0.000    0.006    0.000 {built-in method builtins.max}
    37114    0.003    0.000    0.003    0.000 {built-in method builtins.len}
        1    0.002    0.002    0.009    0.009 history.py:845(writeout_cache)
        1    0.001    0.001    0.010    0.010 history.py:55(only_when_enabled)
     3691    0.000    0.000    0.000    0.000 {built-in method builtins.min}
        2    0.000    0.000    0.000    0.000 zmqstream.py:580(_run_callback)
       14    0.000    0.000    0.000    0.000 socket.py:621(send)
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
     30/3    0.000    0.000    0.000    0.000 {built-in method _abc._abc_subclasscheck}
      2/1    0.000    0.000    0.277    0.277 {built-in method builtins.exec}
        1    0.000    0.000    0.000    0.000 {method 'send' of '_socket.socket' objects}
    80/76    0.000    0.000    0.000    0.000 {built-in method builtins.isinstance}
        2    0.000    0.000    0.000    0.000 wrappers.py:301(__new__)
...
        1    0.000    0.000    0.000    0.000 threading.py:314(_is_owned)
        1    0.000    0.000    0.000    0.000 inspect.py:2874(__init__)
        1    0.000    0.000    0.000    0.000 base_events.py:538(_check_closed)
        2    0.000    0.000    0.000    0.000 contextlib.py:775(__enter__)
        1    0.000    0.000    0.000    0.000 base_events.py:2003(get_debug)
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

Query Performance Comparison

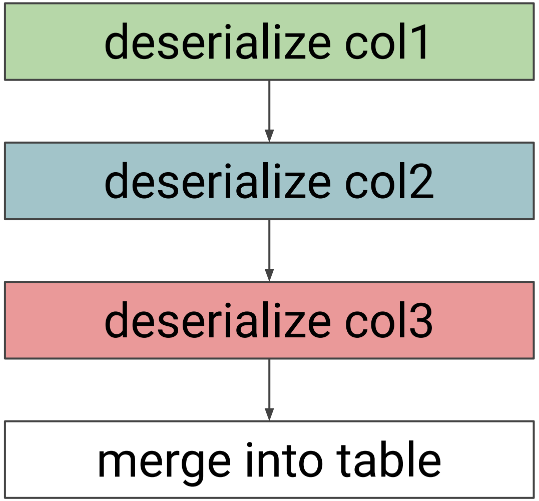
Case	column num	operation	kola + polars	mem(MB)	pykx	mem(MB)	speed
1	14	query from kdb	301 ms ± 4.25 ms	348	381 ms ± 8.52 ms	632	1.27x
1	14	send to kdb	387 ms ± 8.75 ms	708	267 ms ± 11.5 ms	632	0.69x
1	14	cast to pd df	57.1 ms ± 1.85 ms	976	1.36 s ± 39.8 ms	894	23.82x
1	14	send pd df to kdb	506 ms ± 20.6 ms	1203	2.73 s ± 95.9 ms	1093	5.40x
2	64	query from kdb	973 ms ± 18.1 ms	1183	1.39 s ± 22.9 ms	2170	1.43x
2	64	send to kdb	1.21 s ± 42.9 ms	1337	726 ms ± 46.2 ms	2170	0.60x
2	64	cast to pd df	201 ms ± 6.23 ms	1523	1.31 s ± 9.31 ms	2203	6.52x
2	64	send pd df to kdb	1.48 s ± 66.5 ms	1896	3.1 s ± 102 ms	3379	2.09x
3	5 (3+5+5)	query from kdb	397 ms ± 11.1 ms	484	466 ms ± 34.4 ms	694	1.17x
3	5 (3+5+5)	cast to pd df	748 ms ± 23.9 ms	863	1.56 s ± 70.7 ms	1092	2.09x

Larger number in speed column kola+Polars is faster

Parallel Computing

Deserialization

00000000tablecol100000000000000col2000000000000col30000000000000



deserialize col1

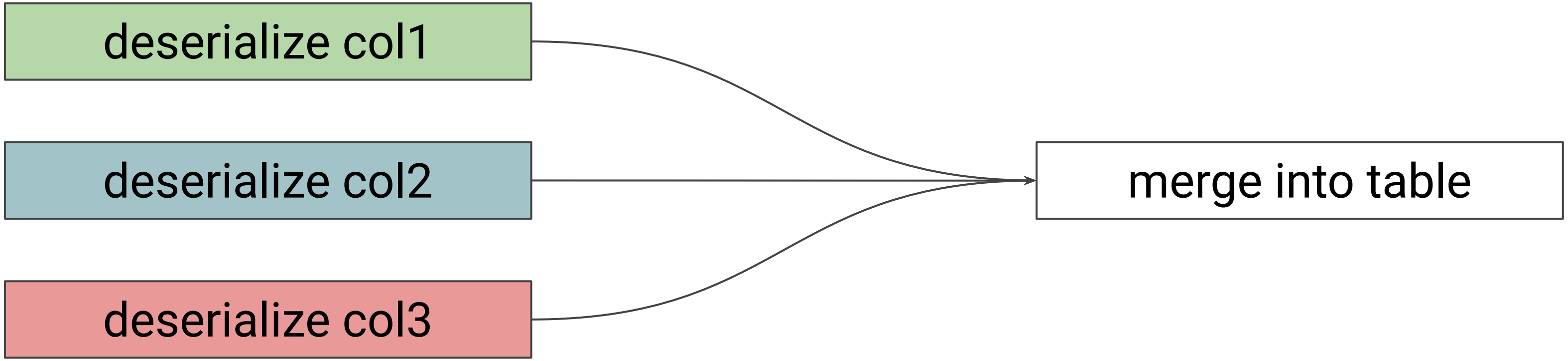
deserialize col2

deserialize col3

merge into table

Parallel Deserialization

00000000tablecol100000000000000col20000000000000col30000000000000



Demo - Querying data within 20M rows * 64 columns

```
n: 2000000;

table: ([]sym: n?`3; time: .z.D + 1000 * "n"$til n; volume:
n?1000; cond: n # enlist "aaa");

columns: `$("ask"; "bid") cross string til 30;

table: ![table; (); 0b; columns!(count columns)#enlist
(?;n;1.0)];
```

New Features since 1.0.0

- IPC protocal ver 6, up to 1TB IPC message
- timeout, if the process is busy
- retries, if the process is not started yet
- a function to generate kdb+ ipc, just like -8! and -18!
- subscription, subscribe to a kdb feedhandler

Why kola?

- open source and free for latest Linux, macOS and Windows
- the most efficient/fastest way to
  - query data from kdb+
  - non-kdb data to kdb+
- extremely fast dataframe operations powered by polars
- very likely support Python 3.13 in Oct 2024, right after Python 3.13 is released
- can be extended to support R (Already works, never make a proper release)

Thank you!

try kola today

pip install kola

let me know if any issues

Questions?