Interoperability vs. Performance

Bringing the best of both to kdb+

Connor Gervin
Partner Engineering Lead, KX

KX STAC M3 Benchmarks - Results

Azure

Hewlett Packard Enterprise

Google Cloud

DELL

aws

NetApp®

WEKA

intel.

Independ **ably** shown as the **fastest** **-series** analysis

- KX is the database in the best SUT in all 24 Kanaga tests (3 queries, large data: 60 TB)
- KX is the database in the best SUT in 15 of 17 Antuco tests (7 queries, medium data: 3.5 TB)

Interoperability – functionality and performance

Native SQL Support

- **Unlock** KX datasets and analytics with PostgreSQL compliant ANSI-SQL
- Hybrid access to **advanced q functionality** through SQL
- SQL executes within **1% performance** of q*
- **Integrate seamlessly** using PostgreSQL Wire Protocol

Power BI

tableau

Grafana

trino

PyKX

- Truly **Python first** approach to leveraging kdb+
- Run **q analytics** in **any Python env** with **official support** for kdb+

```
$ pip install pykx
```

- **Zero copy** type conversions between kdb+ tables and Python
- Execute **q-sql**, **SQL** or **advanced q analytics** through context interface for **local** or **remote** datasets

REST & OpenAPI

- Kurl is a **REST client** that provides sync and async methods callable from q
- Automatic cloud credential discovery for **ease-of-use** and **day zero security** when connecting to services e.g. Object Storage and Logging
- **REST-server** library: expose a RESTful interface to any kdb+ database

OPEN API INITIATIVE

*Not a STAC Benchmark

SQL Support for Streaming, Real-time and Historical Queries

https://code.kx.com/insights/1.3/core/sql.html

```sql
SELECT
  AVG(trip_distance) AS avg_distance,
  AVG(fare_usd + tip_usd) AS avg_totalcost,
  SUM((fare_usd + tip_usd + surcharge_usd + toll_usd)) * trip_distance) / SUM(trip_distance) AS wavg_price,
  COUNT(*) AS trip_total
FROM
  trips
WHERE
  passenger_count > 0
  AND
  fare_amount BETWEEN 0 AND 10000
GROUP BY
  payment_type,
  vendor_id,
  date_trunc('hour', pickup_ts)
ORDER BY
  trip_total DESC
LIMIT
```

payment_type	vendor_id	trip_hour	avg_distance	avg_cost	avg_tip	wavg_price	trip_total
Credit Card	Yellow	2021-01-14 17:00:00	2.48805	10.61531	2.758537	25.18158	1969
Credit Card	Yellow	2021-01-22 15:00:00	2.219567	10.44967	2.5084	23.71187	1987
Credit Card	Uber	2021-01-22 17:00:00	2.472608	10.85417	2.779709	24.99368	1990
Credit Card	Yellow	2021-01-28 14:00:00	2.196332	10.53419	2.500552	23.1389	2012
Debit Card	Uber	2021-01-28 15:00:00	2.242471	10.54985	2.555535	22.41565	2036
Debit Card	Yellow	2021-01-22 18:00:00	2.467032	10.60276	2.718792	22.33199	2062
Credit Card	Yellow	2021-01-29 18:00:00	2.42828	10.62693	2.69837	21.654	2110
Credit Card	Uber	2021-01-29 17:00:00	2.529853	11.15275	2.837771	28.96866	2180
Credit Card	Uber	2021-01-28 17:00:00	2.313559	10.53046	2.669478	24.06371	2242
Credit Card	Uber	2021-01-28 18:00:00	2.456106	10.66506	2.722278	22.81794	2401

Taxi & Limousine Commission

Python Native Syntax for local and remote time-series analysis

https://code.kx.com/pykx/

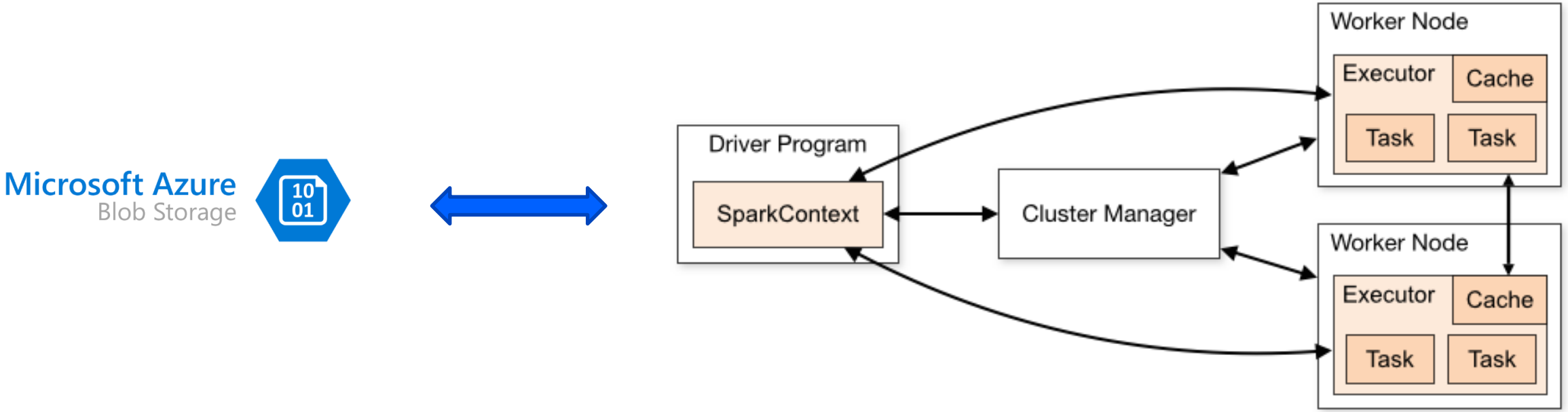
```python
import pykx as kx

with kx.QConnection('demo.kxlab.com', 5000) as conn:
    trade = conn.sql("SELECT * FROM trade WHERE date = '2020-12-29' and sym ='MSFT'")
    quote = conn.sql("SELECT * FROM quote WHERE date = '2022-12-29' and sym ='MSFT'")

stock_joined = kx.q.aj('date', trade, quote)

stock_joined.pd()
```

	date	sym	time	price	size	stop	cond	ex	bid	ask	bsize	asize	mode
0	2020-12-29	MSFT	0 days 15:59:36.269000	14.447011	903	False	b'Z'	b'E'	10.436717	18.275007	359	833	b'R'
1	2020-12-29	MSFT	0 days 15:59:36.269000	17.527927	851	True	b'H'	b'E'	10.436717	18.275007	359	833	b'R'
2	2020-12-29	MSFT	0 days 15:59:36.269000	12.315513	694	False	b'R'	b'E'	10.436717	18.275007	359	833	b'R'
3	2020-12-29	MSFT	0 days 15:59:36.269000	11.110300	11	True	b'R'	b'E'	10.436717	18.275007	359	833	b'R'
4	2020-12-29	MSFT	0 days 15:59:36.269000	15.380307	832	True	b'H'	b'E'	10.436717	18.275007	359	833	b'R'

Accelerating Spark workflow with PyKX

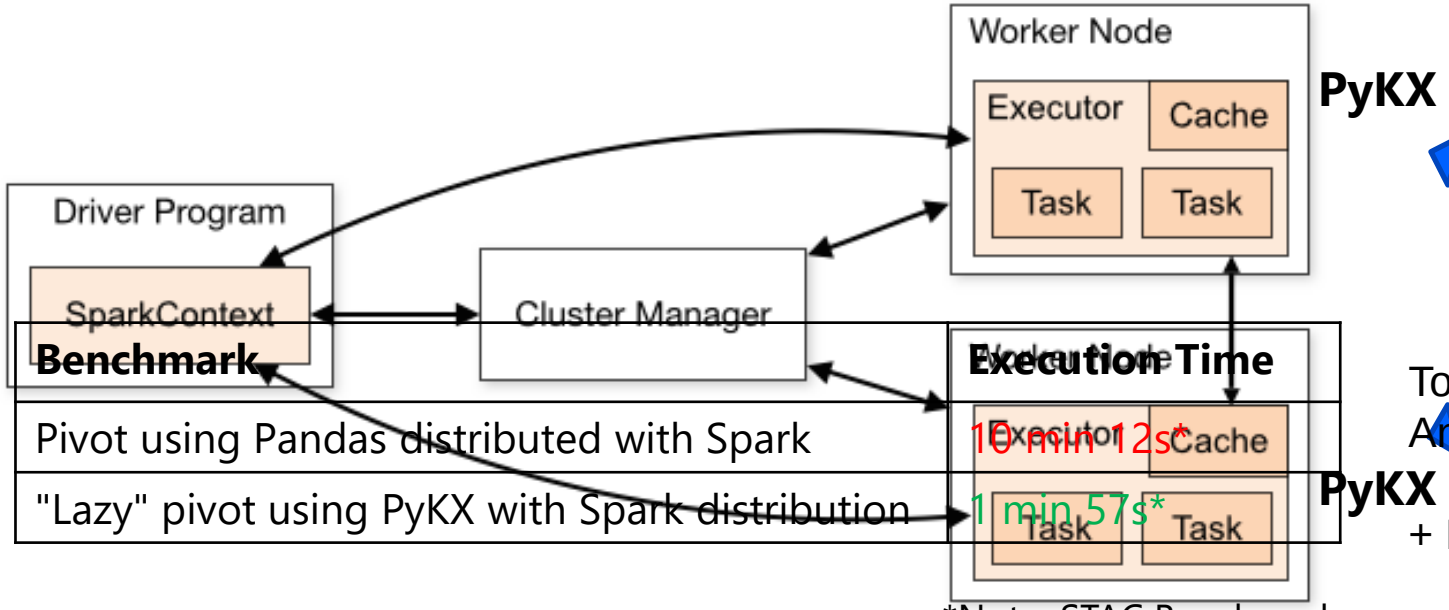
- Open-source NASA aircraft sensor data
- Data stored on Blob Storage in Parquet format
- Perform a large-scale distributed pivot on the dataset

Pivot using Pandas distributed with Spark



Accelerating Spark workflow with PyKX

"Lazy" pivot using PyKX with Spark distribution



Benchmark	Execution Time
Pivot using Pandas distributed with Spark	10 min 12s*
"Lazy" pivot using PyKX with Spark distribution	1 min 57s*

*Not a STAC Benchmark

Microsoft Azure
Blob Storage

Total number of flights: 180,000
Analysis time saved over pandas: **~ 35hrs**

+ Future improvements coming with APACHE ARROW

Thank You

For more information:

Connor Gervin

cgervin@kx.com

Visit **www.kx.com**

KX Kx Kxsystems Kxsystems\