DATA+AI SUMMIT 2022

CKDelta

GIS Pipeline acceleration with Apache Sedona

Alihan Zihna
Senior Data Scientist, CKDelta (Hutchison Group)

Fernando Ayuso
Director of Data Science and Data Engineering, CKDelta (Hutchison Group)

ORGANIZED BY databricks

CKDelta

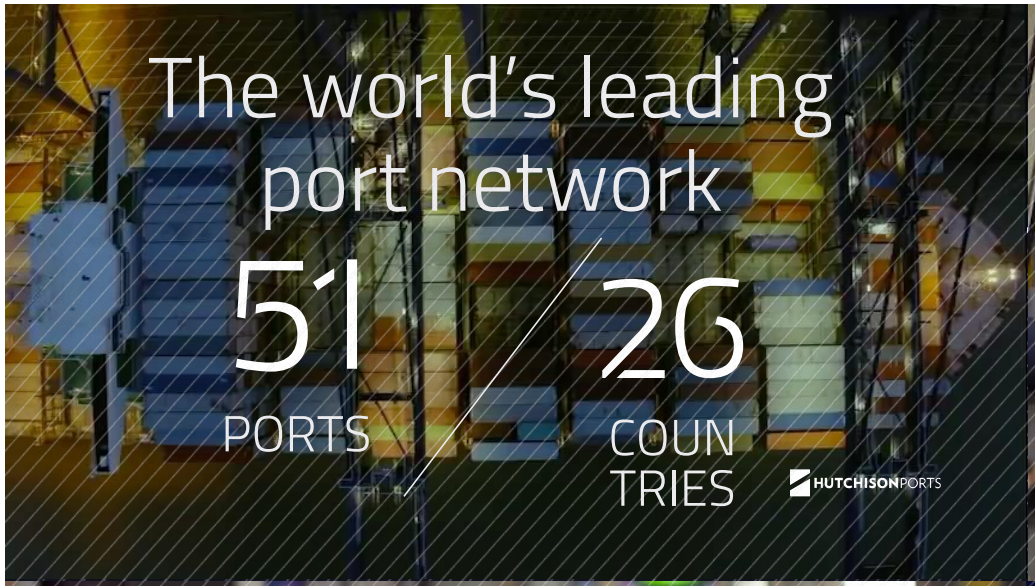
Agenda

What are we sharing today?

- Business context
- Use case
- Technical implementation
- Demo

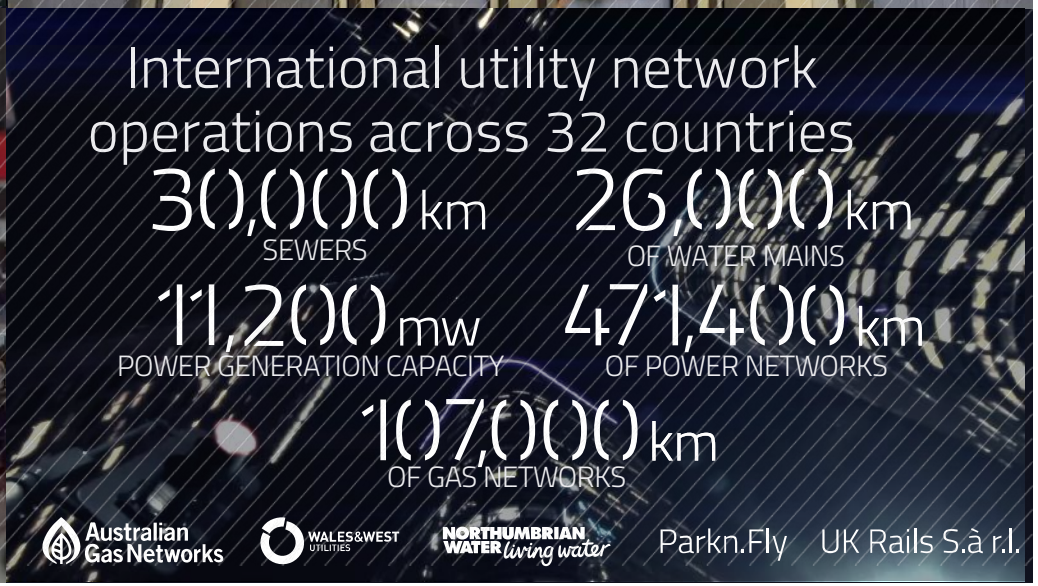
Business Context

CKDelta


The world's leading port network
51
PORTS
26
COUNTRIES
HUTCHISONPORTS

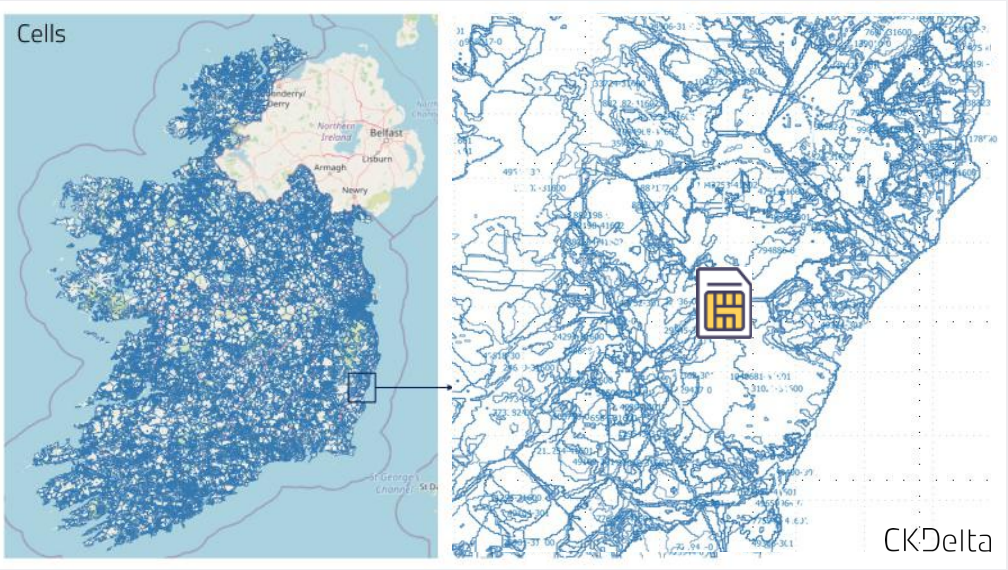

The world's largest health & beauty retailer
14,500
STORES IN 24 MARKETS
watsons
Superdrug
Marionnaud
PARIS


A world-leading mobile data network
128m+
DATA USERS ACROSS 11 COUNTRIES
HUTCH
vietnamobile


International utility network operations across 32 countries
30,000 km
SEWERS
26,000 km
OF WATER MAINS
11,200 mw
POWER GENERATION CAPACITY
471,400 km
OF POWER NETWORKS
107,000 km
OF GAS NETWORKS
Australian Gas Networks
WALES&WEST UTILITIES
NORTHUMBRIAN WATER living water
Parkn.Fly
UK Rails S.à r.l.

Telco data + GIS

With over **26 billion records** ingested and analysed per day in the platform CKDelta is striving to add **granularity** to telco analysis



+2M Cells in our platform – High Complexity

SQL or GIS tools not sufficient

+ granularity & Speed

Combination of python libraries with Spark UDFs on Databricks

Utilities
a GIS rich industry

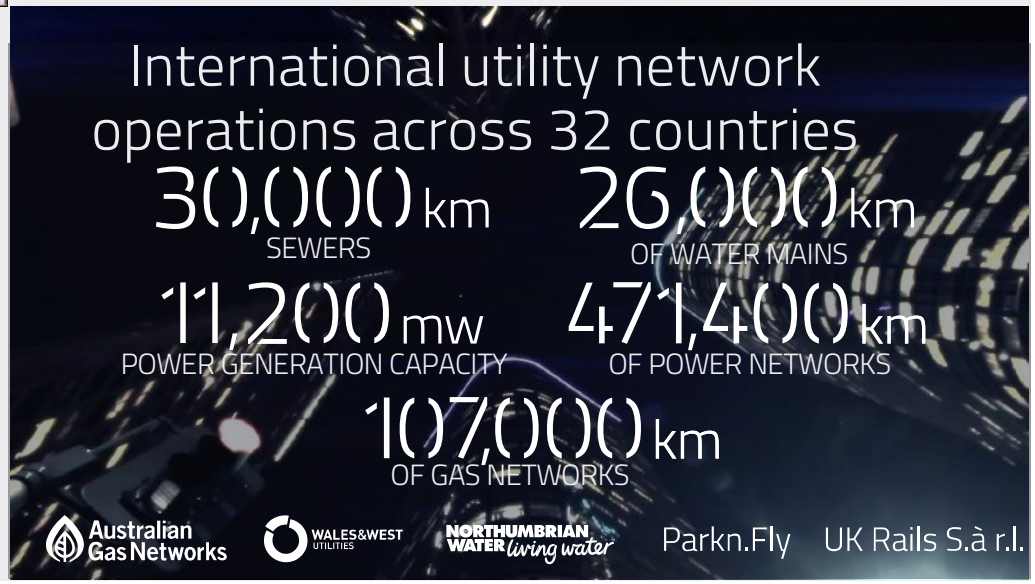
CKDelta

CKDelta works with Northumbria Water, UK Power Networks, ISTA or Northern Gas Networks solving problems through data.

Millions of GIS components

Overlaying networks: Water, Gas, Electricity...

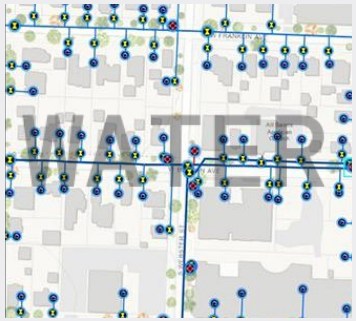
Limitation of what GIS can do using only specialised libraries



Use cases: Telco + Utilities

CKDelta

Limits in technology limit the imagination of our solutions

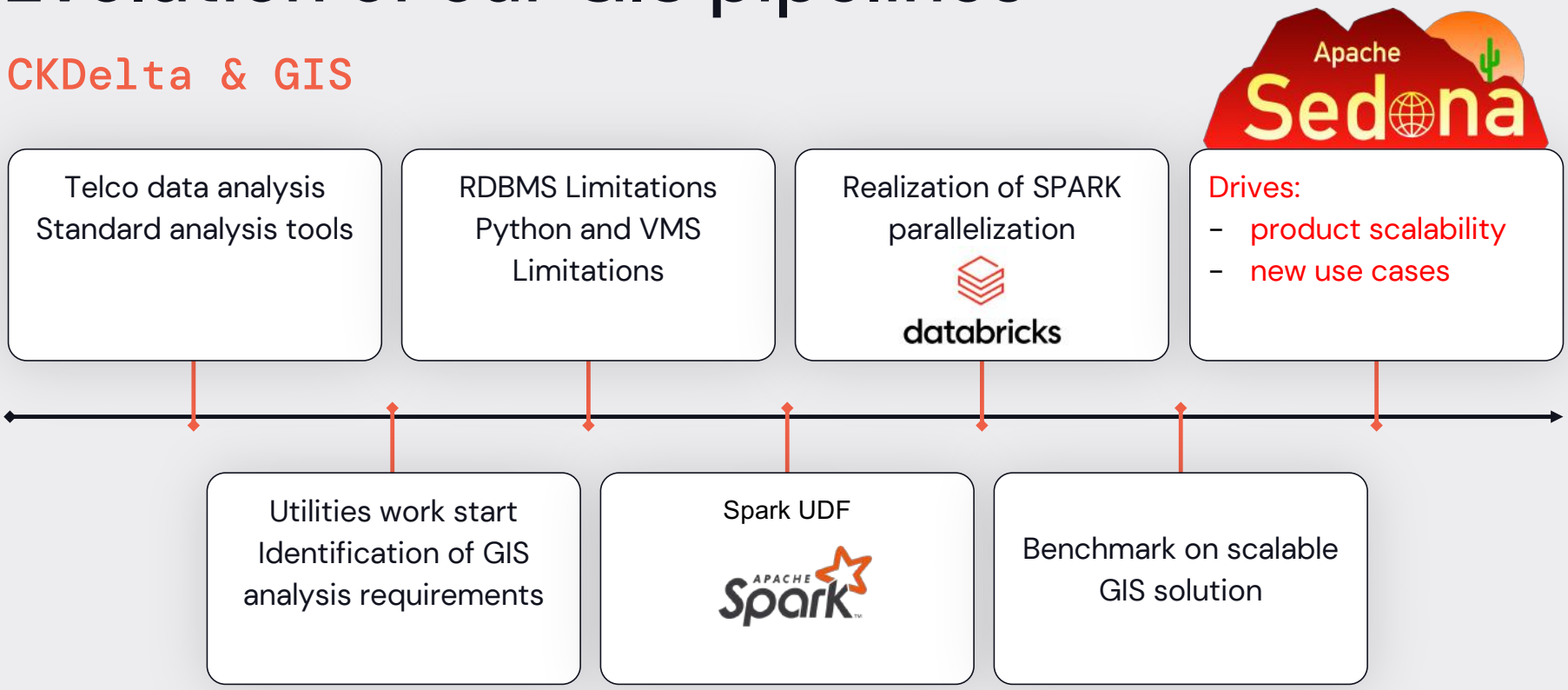


- Water Leakage
- Customer satisfaction
- Electrical Vehicle Charging Station installation and consumption location
- Supply VS Demand

CKDelta

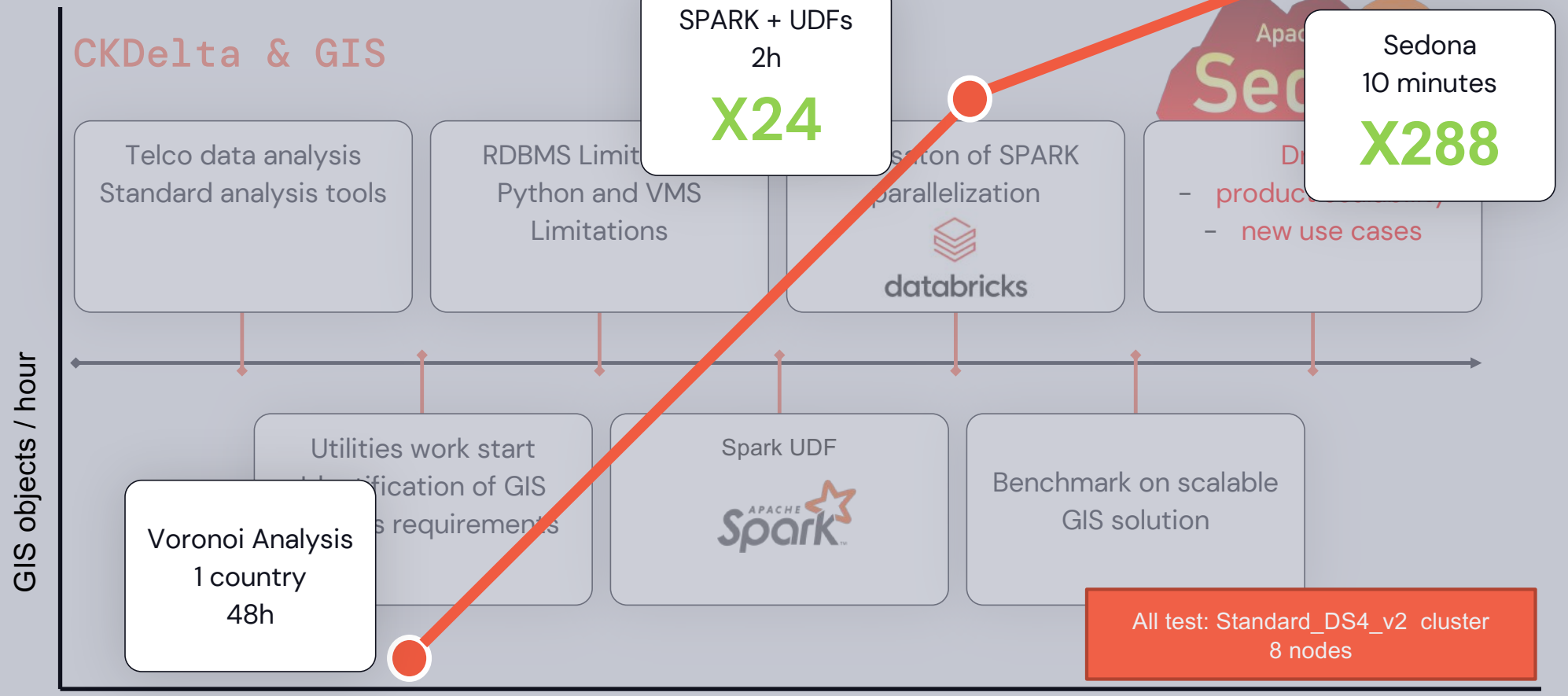
Evolution of our GIS pipelines

CKDelta & GIS



CKDelta

Evolution of our GIS pipelines



Technical Implementation

CKDelta

CKDelta

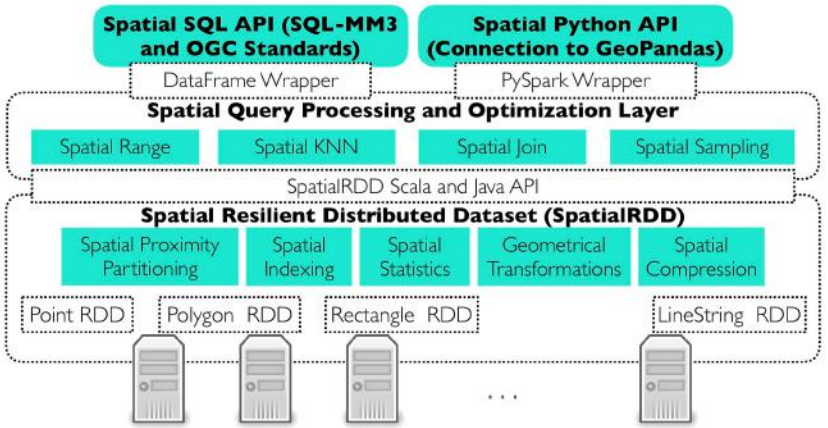
Sedona on Databricks

How does Sedona work?

Extending Spark

- Sedona extends the Spark context by new spatial level functions
- Introduces the basic building blocks of GIS to spark such as Polygons, Lines, Spatial Statistics, Spatial Indices

Architecture



CKDelta

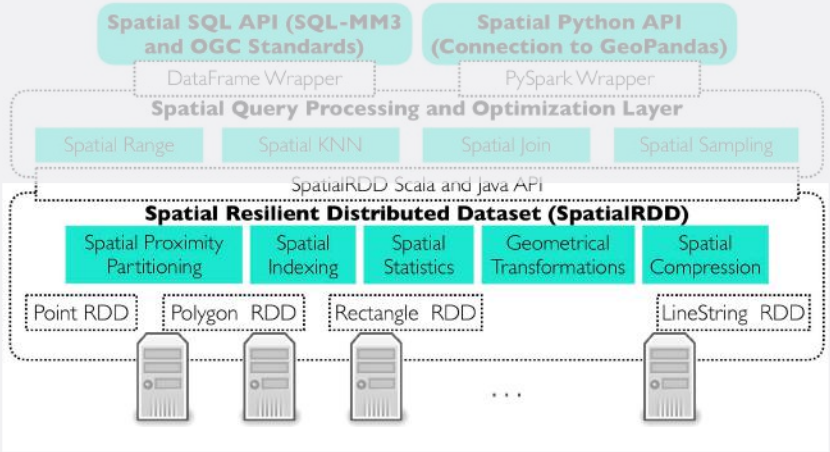
Sedona on Databricks

How does Sedona work?

Extending Spark

- Sedona extends the Spark context by new spatial level functions
- Introduces the basic building blocks of GIS to spark such as Polygons, Lines, Spatial Statistics, Spatial Indices

Architecture



CKDelta

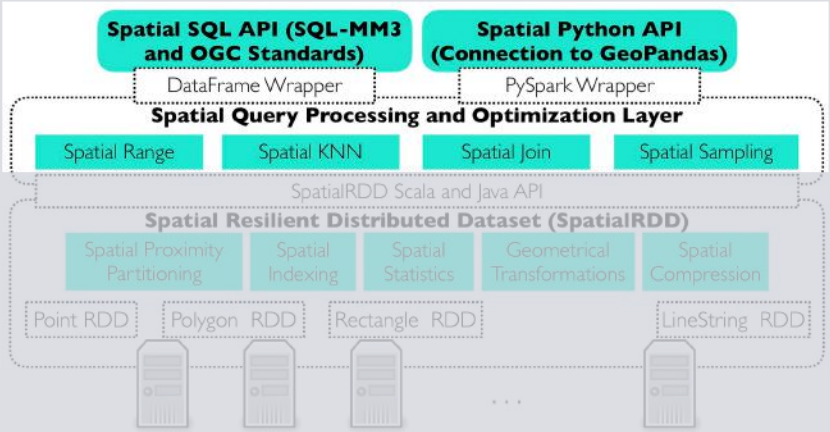
Sedona on Databricks

How does Sedona work?

Extending Spark

- Sedona extends the Spark context by new spatial level functions
- Introduces the basic building blocks of GIS to spark such as Polygons, Lines, Spatial Statistics, Spatial Indices

Architecture



CKDelta

Sedona on Databricks

Spatial Functions

Constructors

- ST_GeomFromWKT
- ST_GeomFromGeoJson
- ST_Point

Functions

- ST_Distance
- ST_Length
- ST_Area
- ST_Intersection
- ST_IsValid

Aggregate

- ST_Envelope_Aggr
- ST_Union_Aggr
- ST_Intersection_Aggr

CKDelta

Sedona on Databricks

Spatial Functions

Predicates

- ST_Contains
- ST_Intersects
- ST_Within
- ST_Overlaps

Join Query Optimizers

Other functions can be used to optimize GeoSpatial functions such as joining where shapes:

- intersect
- the distance between is low

CKDelta

Sedona Sample

Read File and Select only valid Shapes

```
# Read to RDD and Convert to Spark DataFrame
rdd = GeoJsonReader.readToGeometryRDD(sc, path)
Adapter.toDf(rdd, spark).createOrReplaceTempView('temp')
# Filter only Valid
spark.sql(
    """
    SELECT * FROM temp
    WHERE ST_IsValid(geometry)
    """
)
```

CKDelta

What does Sedona Introduce

Points, Lines, and Polygons

Points

Cmd 3

```
from shapely.geometry import Point

data = [
    [1, Point(21.0, 52.0)],
    [1, Point(23.0, 42.0)],
    [1, Point(26.0, 32.0)]
]


gdf = spark.createDataFrame(
    data,
    ['index', 'geom']
)

gdf.show()
```

▸ (2) Spark Jobs

▸ gdf: pyspark.sql.dataframe.DataFrame = [index: long, geom: udt]

```
+-----+-------------+
|index|         geom|
+-----+-------------+
|    1|POINT (21 52)|
|    1|POINT (23 42)|
|    1|POINT (26 32)|
+-----+-------------+
```

Lines

Cmd 4

```
from shapely.geometry import LineString

line = [(40, 40), (30, 30), (40, 20), (30, 10)]

data = [
    [1, LineString(line)]
]

gdf = spark.createDataFrame(
    data,
    ['index', 'geom']
)

gdf.show(1, False)
```

▸ (2) Spark Jobs

▸ gdf: pyspark.sql.dataframe.DataFrame = [index: long, geom: udt]

```
+-----+----------------------------------------+
|index|geom                                    |
+-----+----------------------------------------+
|1    |LINESTRING (40 40, 30 30, 40 20, 30 10)|
+-----+----------------------------------------+
```

Polygons

Cmd 6

```
from shapely.geometry import Polygon

polygon = Polygon(
    [
        [19.51121, 51.76426],
        [19.51056, 51.76583],
        [19.51216, 51.76599],
        [19.51280, 51.76448],
        [19.51121, 51.76426]
    ]
)

data = [
    [1, polygon]
]

gdf = spark.createDataFrame(
    data,
    ['index', 'geom']
)

gdf.show(1, False)
```

▸ (2) Spark Jobs

▸ gdf: pyspark.sql.dataframe.DataFrame = [index: long, geom: udt]

```
+-----+-----------------------------------------------------------------------
|index|geom
+-----+-----------------------------------------------------------------------
|1    |POLYGON ((19.51121 51.76426, 19.51056 51.76583,
+-----+-----------------------------------------------------------------------
```

What does Sedona Introduce

Spatial Indexing and Partitioning

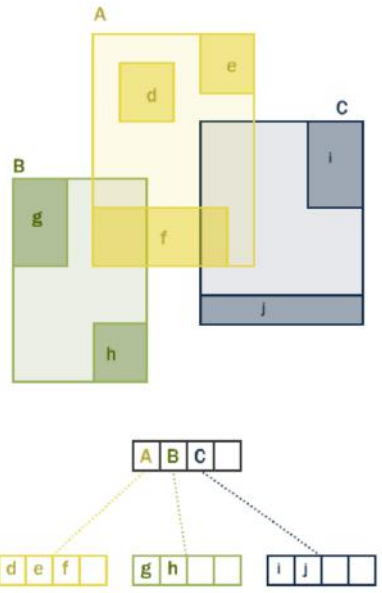
Spatial indexes:

- Allow faster access to records

Spatial Partitioning:

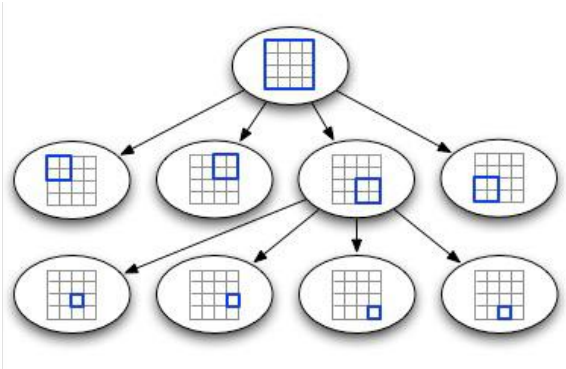
- Allow close objects to be in the same partitions

Rtree



http://postgis.net/workshops/postgis-intro/indexing.html

Quad Tree



https://veendeta.wordpress.com/2012/02/07/algorithm-space-partitioning-using-quadtree/

Serialization

What do we need serialization?

Serialization Basics

Required to reduce the footprint when:

- Writing to a file or a database
- Transferring through network (e.g. to other workers)

Spark uses Java Serialization by default.

Kryo vs. Java Serialization

Kryo serializer is more efficient and safer than Java serializer by 20%-50%.

New data objects must be defined individually. If undefined:

- Security implications
- Reduced performance

CKDelta

What does Sedona Introduce

Efficient Serialization

Reducing Memory Impact

Represents the shape objects as bytes using:

- wkb serializer
- shape serializer

Can be configured using spark.configs

Serialization Example

Point (50.323281, 19.029889)

0 0 0 1|-121 53 –107 69 97 73 64|-103 –126 53 –50 –90 7 51 64

Geometry Type

X Coordinate

Y Coordinate

Sedona on Databricks

Which version to install?

Depends on Databricks Runtime Version and Spark Version

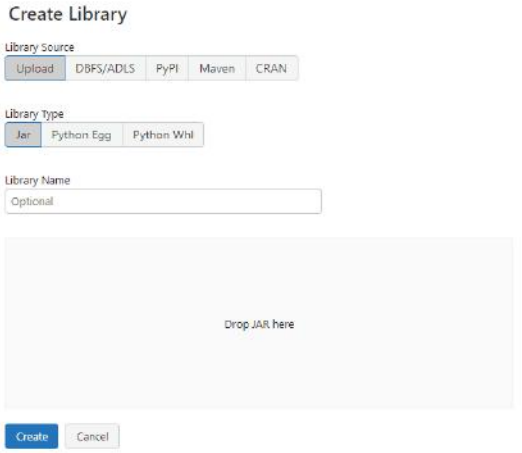
Cluster Runtime Major	Spark	Sedona version
7 - 9	3.0 - 3.1	1.0.1 - 1.1.0
10 - 11	3.2	1.1.1 - 1.2.0

Sedona on Databricks

CKDelta

How to set-up

Installing Using UI



Simple Setup

- Add packages as libraries
- Find the external packages
- Install to the clusters

Drawbacks

- No version control
- Can't benefit from serialization

Sedona on Databricks

CKDelta

How to set-up

Installing Using init scripts

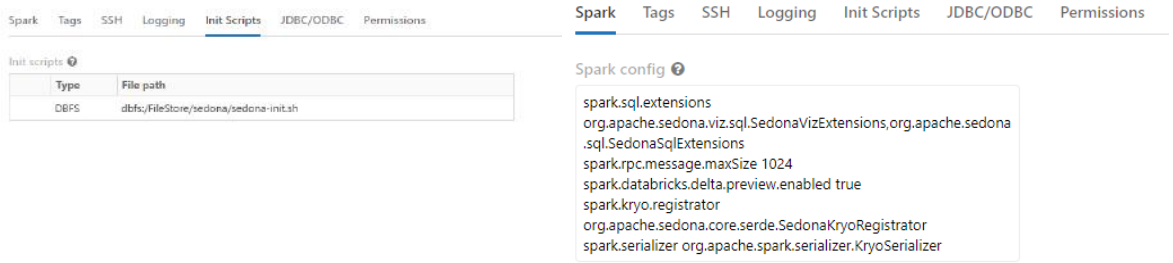
```
%sh

# Create init script directory for Sedona
mkdir -p /dbfs/FileStore/sedona/

# Create init script
cat > /dbfs/FileStore/sedona/sedona-init.sh <<'EOF'
#!/bin/bash
#
# File: sedona-init.sh
# Author: Erni Durdevic
# Created: 2021-11-01
#
# On cluster startup, this script will copy the Sedona jars to the cluster's default jar directory.
# In order to activate Sedona functions, remember to add to your spark configuration the Sedona extensions:
# "spark.sql.extensions org.apache.sedona.viz.sql.SedonaVizExtensions,org.apache.sedona.sql.SedonaSqlExtensions"

cp /dbfs/FileStore/jars/sedona/1.2.0-incubating/*.jar /databricks/jars

EOF
```



Copy the sample scripts from Sedona setup page

Add the bash file to the cluster's init scripts.

Update the spark configs

CKDelta

Sedona on Databricks

Configurations

Spark Configs for Spark

```
spark.sql.extensions org.apache.sedona.viz.sql.SedonaVizExtensions,org.apache.sedona.sql.SedonaSqlExtensions
spark.serializer org.apache.spark.serializer.KryoSerializer
spark.kryo.registrator org.apache.sedona.core.serde.SedonaKryoRegistrator
spark.rpc.message.maxSize 1024
```

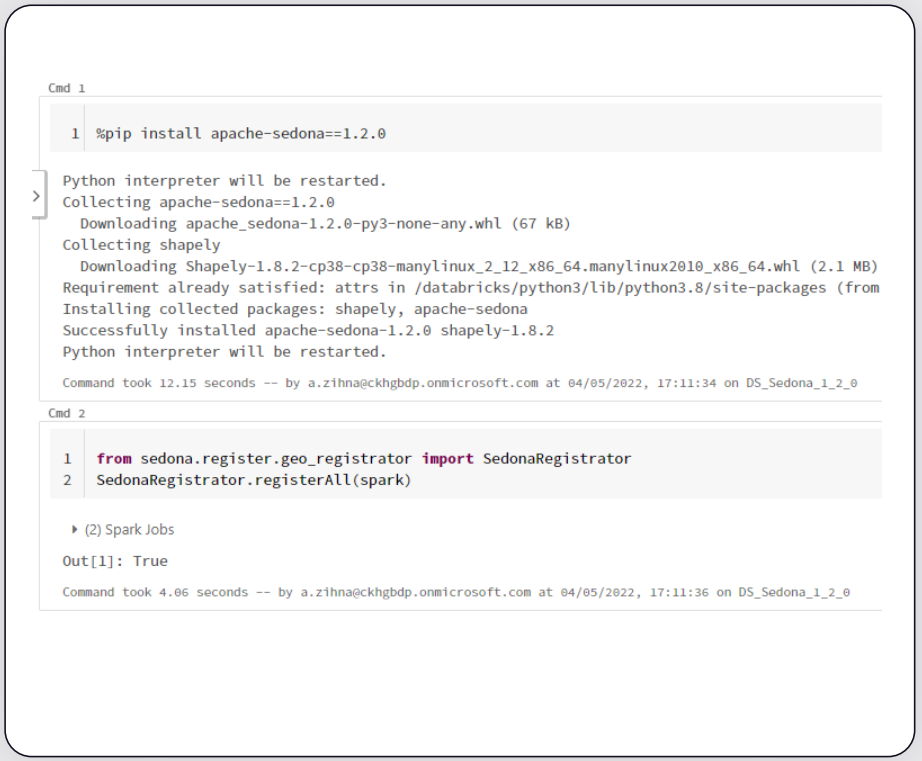
CKDelta

Sedona on Databricks

Using in notebook

SedonaRegistrator.registerAll() enables all new functions.

Sedona can now be used with pyspark or SQL functions.



CKDelta

Demo

Conclusion

- Apache Sedona extends Spark
- Minimal changes required for migration
- Easy setup on Databricks from Sedona's docs
- Significant performance upgrade

DATA+AI SUMMIT 2022

Thank you

Alihan Zihna
Senior Data Scientist

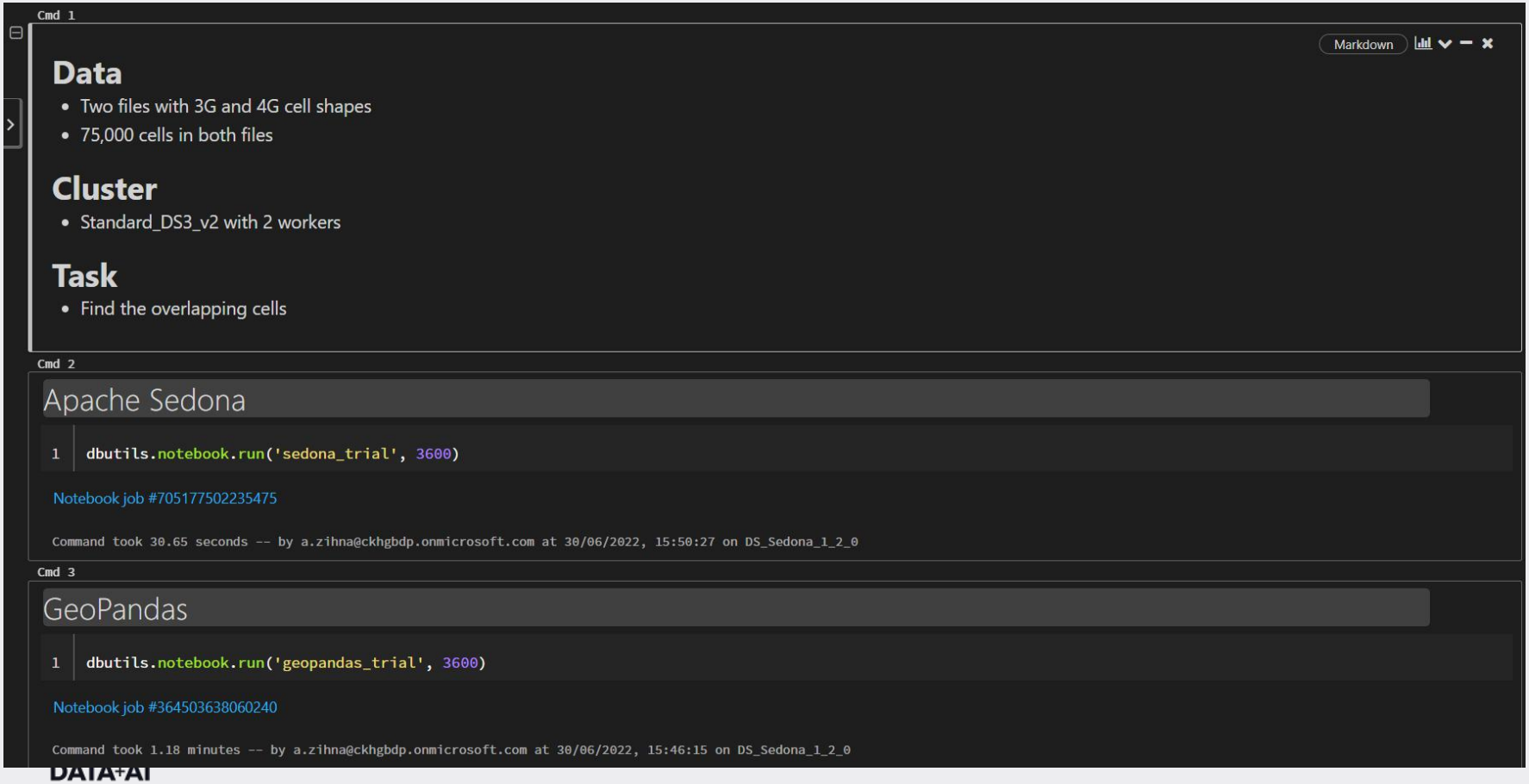
Fernando Ayuso
Director of Data Science and Data Engineering

Appendix

We are
CKHutchison's
Group data innovation company

A member of CK Hutchison Holdings

Performance Comparison



GeoPandas Implementation

Cmd 2

Import Python Modules

Python

```
1  import geopandas as gpd
```

```
/local_disk0/.ephemeral_nfs/envs/pythonEnv-19d88e75-85df-4291-b273-22c9d2ecb8a5/lib/python3.8/site-packages/geopandas/_compat.py:111: UserWarning: The Shapely GEOS version (3.10.2-CAPI-1.16.0) is incompatible with the GEOS version PyGEOS was compiled with (3.10.1-CAPI-1.16.0). Conversions between both will be slow.
  warnings.warn(
```

Command took 0.44 seconds -- by a.zihna@ckhgbdp.onmicrosoft.com at 13/06/2022, 16:22:26 on DE_General

Cmd 3

Read Files

```
1  df_3g = gpd.read_file('/dbfs/dais/vor_dais_3g.geojson')
2  df_4g = gpd.read_file('/dbfs/dais/vor_dais_4g.geojson')
```

Command took 1.56 minutes -- by a.zihna@ckhgbdp.onmicrosoft.com at 13/06/2022, 16:22:26 on DE_General

Cmd 4

Get the intersections of polygons

```
1  gpd.overlay(df_3g, df_4g, keep_geom_type=False).head()
```

Sedona Implementation – 1

Cmd 2

Import Sedona Modules and Register

```
from sedona.core.formatMapper import GeoJsonReader
from sedona.register.geo_registrator import SedonaRegistrator
from sedona.utils.adapter import Adapter

SedonaRegistrator.registerAll(spark)
sc = spark.sparkContext
sc.setSystemProperty("sedona.global.charset", "utf8")
```

Cmd 3

Read in the data as Spatial RDDs

```
rdd_3g = GeoJsonReader.readToGeometryRDD(sc, 'dbfs:/dais/vor_dais_3g.geojson')
df_3g = Adapter.toDf(rdd_3g, spark).createOrReplaceTempView('df3g')

rdd_4g = GeoJsonReader.readToGeometryRDD(sc, 'dbfs:/dais/vor_dais_4g.geojson')
df_4g = Adapter.toDf(rdd_4g, spark).createOrReplaceTempView('df4g')
```

Sedona Implementation – 2

Cmd 4

Define the opertation with SQL

```
spatial_join_result = spark.sql(
    """
        SELECT aa.ci3g
             , ca.ci4g
        FROM df3g AS aa
           , df4g AS ca
        WHERE ST_Intersects(aa.geometry, ca.geometry)
    """
)
spatial_join_result.display()
```

Result Comparison

GeoPandas

	ci3g	ci4g
0	10-89-13810-6247	10-89-0-313681-2
1	10-89-13810-6247	10-89-0-313681-2
2	10-89-13810-6247	10-89-0-313681-2
3	10-89-13810-3093	10-89-0-313681-2
4	10-89-13810-3093	10-89-0-313681-2

Apache Sedona

	ci3g	ci4g
1	510-89-10942-3032	10-89-10699-110096
2	510-89-10942-3032	10-89-10699-110096
3	510-89-10942-3702	10-89-10699-110096
4	510-89-10942-3702	10-89-10699-110096
5	510-89-10942-3032	10-89-10699-110096
6	510-89-10942-3702	10-89-10699-110096
7	510-89-10942-3218	10-89-10699-110096