Astroinformatics Symposium - From Big Data to Understanding the Universe at Large

Large Scale Data Management of Astronomical Surveys with AstroSpark

Karine Zeitouni   **Mariem Brahem***   **Laurent Yeh**

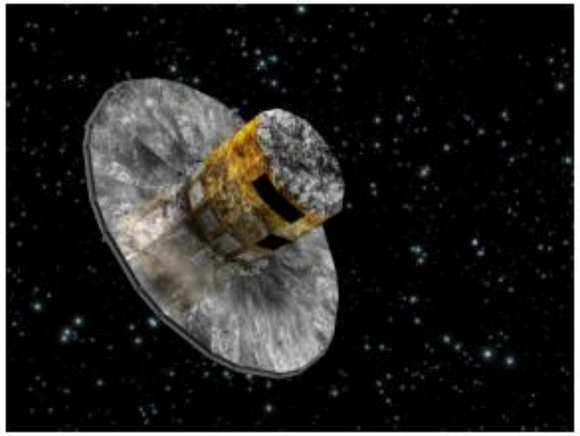
Computer Science Dept, University of Versailles (UVSQ), France
(*) PhD co-funded by UVSQ & CNES Toulouse

david
données et algorithmes
pour une ville intelligente et durable

UNIVERSITÉ DE VERSAILLES ST-QUENTIN-EN-YVELINES
université PARIS-SACLAY

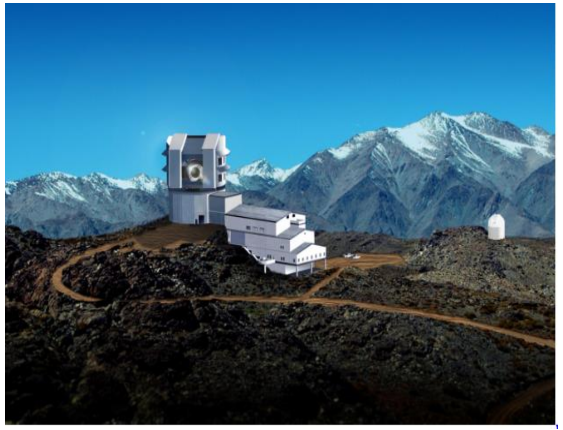
Introduction

Data Deluge in Sky surveying



Gaia mission, ESA

- The largest and most precise 3D map of the Galaxy
- 1 billion stars observed over 5 years
- Data Volume ->**1PB**
- Dec. 2013- 2020



LSST project

- Tens of billions of objects
- Data Volume -> **15PB** for the catalog
- >2020 ->+ 10 years

➔ How to efficiently manage such Big Data in astronomy ?

Today's Situation

- Astronomical Surveys are mostly accessed using SQL Dialect
  - ✓ By integrating a library of geometrical functions to SQL (eg. ADQL)
  - ✓ And spatial indexing techniques to optimize the query execution
- Mainly implemented within relational DBMSs
  - SkyServer for SDSS
  - Postgres Q3C & Pgsphere
  - …

➔ ***But do not scale with the expected data volume***

- Popular Big Data platforms propose a distributed frameworks
  - ✓ Most tend to implement SQL
  - ✓ Allow user-defined functions

➢ Apache Spark is among the most popular frameworks

➔ ***But do not support astronomical data access and manipulation***

Objective

- Combining the best of two worlds:
  - Expressivity of the declarative query language
    - We choose ADQL as a basis for the SQL dialect
  - Scalability of distributed frameworks
    - We choose Apache Spark as a cluster computing ptatform

Astronomical Data Query Language
(ADQL)

SELECT alpha, delta, sourceID
FROM gums
WHERE CONTAINS(POINT(‘ICRS’,alpha,delta),
CIRCLE('ICRS', 10, 5, 1)) = 1;

IVOA

1. **How to allow the support of ADQL within SPARK SQL ?**
2. **How to optimize the query processing in this context ?**

Why SPARK ?

- **Up to 100× faster** than Hadoop MapReduce
  - thanks to its execution engine that supports acyclic data flow
  - and in-memory computing.
- Improves **usability** (2 to 10 less code) through:
  - Rich APIs in Java, Scala, Python
  - Interactive shell, SQL
  - Works with any Hadoop-supported storage system (HDFS, Amazon S3, Avro, Parquet…)
- Provides 2 types of Operations:
  - Transformations (e.g. map, filter, groupBy, join) -> **Lazy operations** to build RDDs from other RDDs
  - Actions (e.g. reduce, count, collect, save) -> Return a result or write it to storage

Outline

- Introduction
- **AstroSpark Architecture**
- Data Patitioning Algorithm
- Cross-Matching
- Experimental Evaluation
- Conclusion

Towards AstroSpark

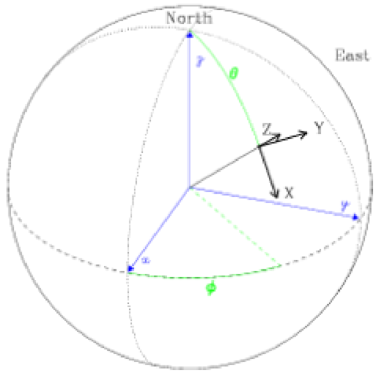
➢ We propose a distributed framework specifically tailored for data intensive applications in astronomy

- This leads to revisit the optimization techniques in this perspective :
  - ✓ Physical organization of data: Partitioning
  - ✓ Logical and physical query optimization and processing
  - ✓ Using a Cost model (I/O, CPU, Communication, Coordination, …) in the query evaluation
  - ✓ Multi-query optimization: caching techniques

Observations & Design Principle

- **Specificity of the data**
  - Astrometric data are typically **big spatial data**
  - Use **spherical coordinates** (e.g. International Celestial Reference System - ICRS)

  ⇒Data organization matters



- **Specificities of the queries**
  - Frequent use of distance-based filtering, joins and top-k: **Cone Search**, **Cross-Matching**, **Nearest Neighbors**
  - **Complex** data processing due to large volume of data and the variety of the queries

  ⇒Algorithms and query plan should be adapted

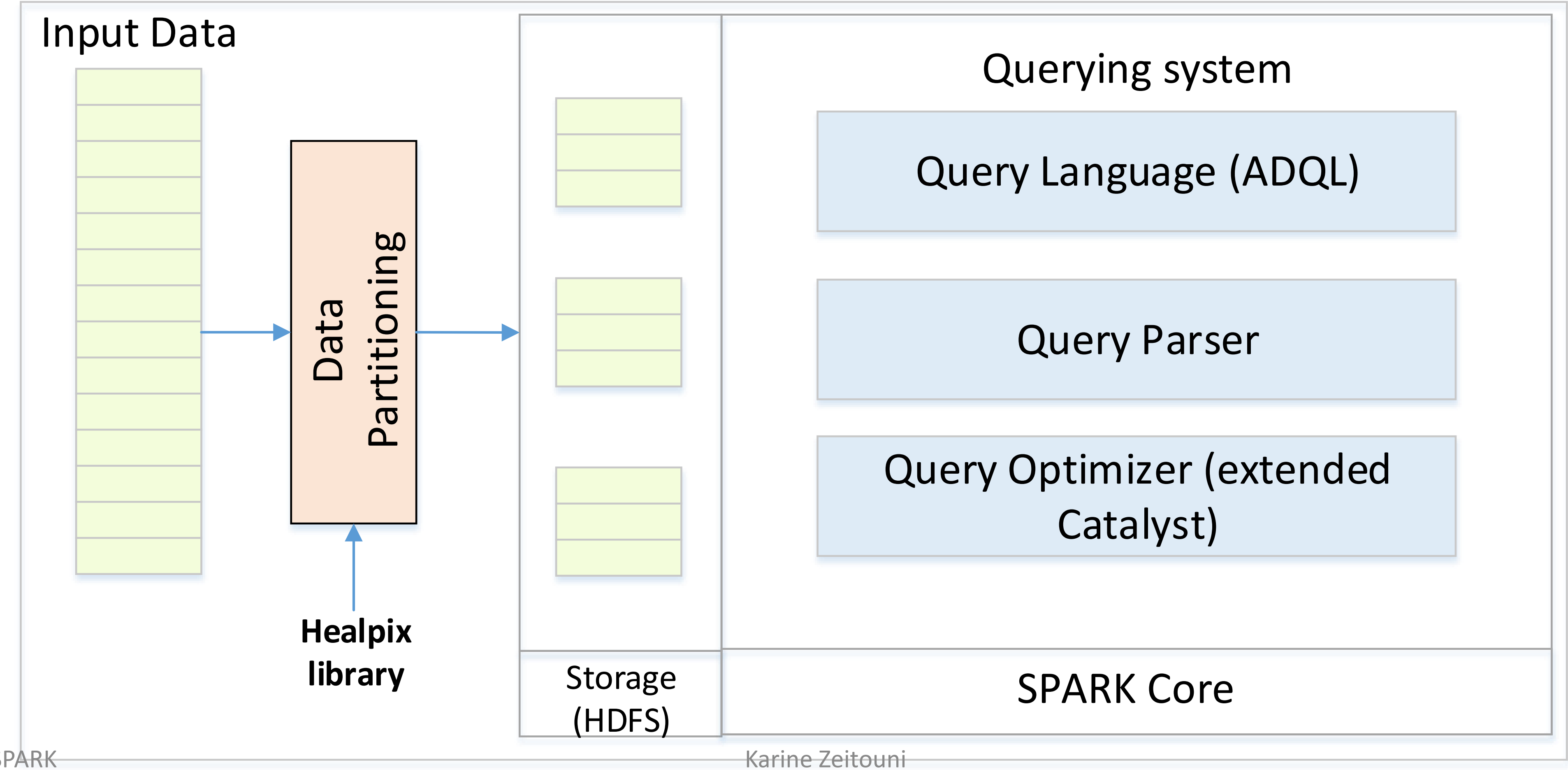
Observations & Design Principle

- ★ **Principle 1: Reuse proven methods and tools**
  - Spatial indexing techniques are widely used in sky surveying

  ⇒Reuse HEALPIX index & library

- ★ **Principle 2: leverage the power of the target framework**
  - Add the **the strictly necessary** extension
    - ✓To support the query syntax of ADQL
    - ✓To evaluate and optimize the queries

AstroSpark Architecture

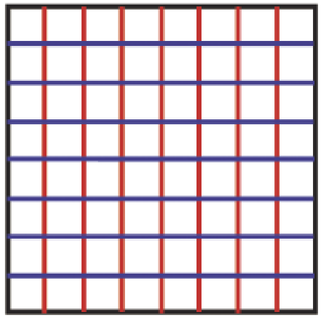


Partitioning

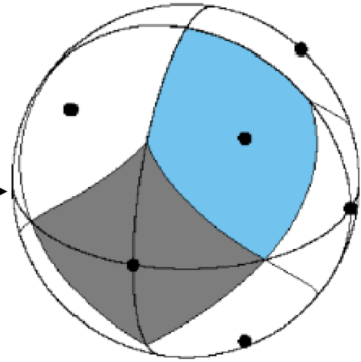
- Definition
  - Partitioning is the process of dividing data into subgroups
  - Partitions are processed in parallel with different nodes
  - One node can process many partitions
- Importance
  - Enables query processing in parallel
  - Reduces computer resources
  - Improves query performances

Partitioning Requirements in AstroSpark

1. Data locality
   - Points that are located close to each other should be in the same partition.
   - Adapt to the spherical space
2. Load balancing
   - Avoid imbalanced partitions
   - Partitioning should be adaptive to the data distribution



Grid partitioning

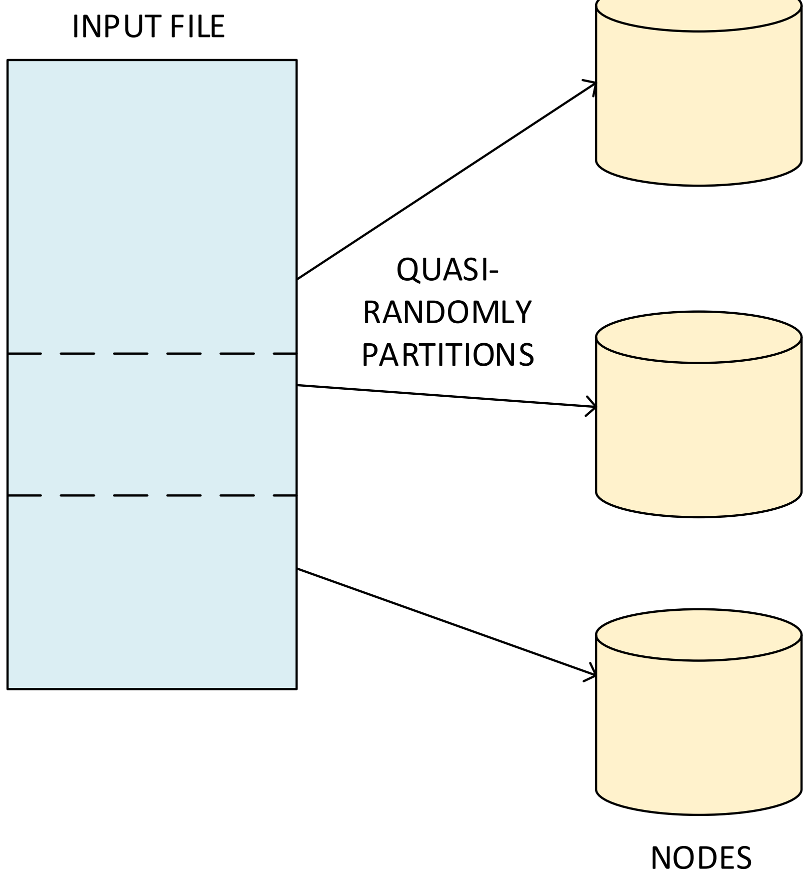


Spherical partitioning

Partitioning in Spark

- Hash partitioning
  - Partitions data quasi-randomly
  - No data locality => not adapted to proximity queries
- Range partitioning
  - Partitions data into roughly equal ranges
  - Partition key is only one dimensional

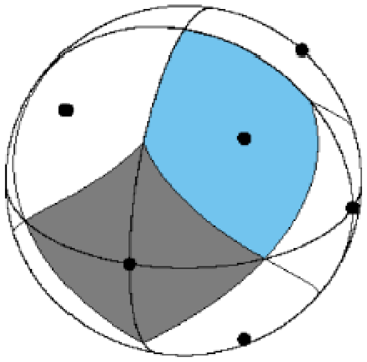
➔ **But, our target is multi-dimensional…**



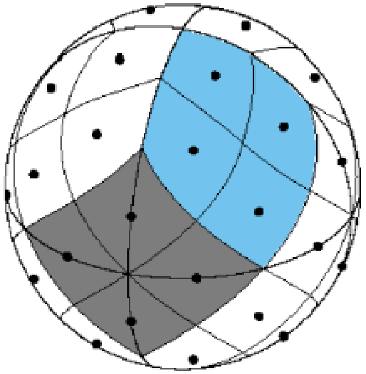
Healpix Based Range Partitioning

- Healpix: **H**ierarchical **E**qual **A**rea iso**L**atitude **Pix**elization of a sphere [NASA]
  - A structure for hierarchical pixelization of the data on the sphere
  - Assigns a **1D** index to each pixel in a way it keeps data locality
  - NSIDE = the amount of subdivision of base pixels

- **Use spark range partitioner with Healpix as partition key**

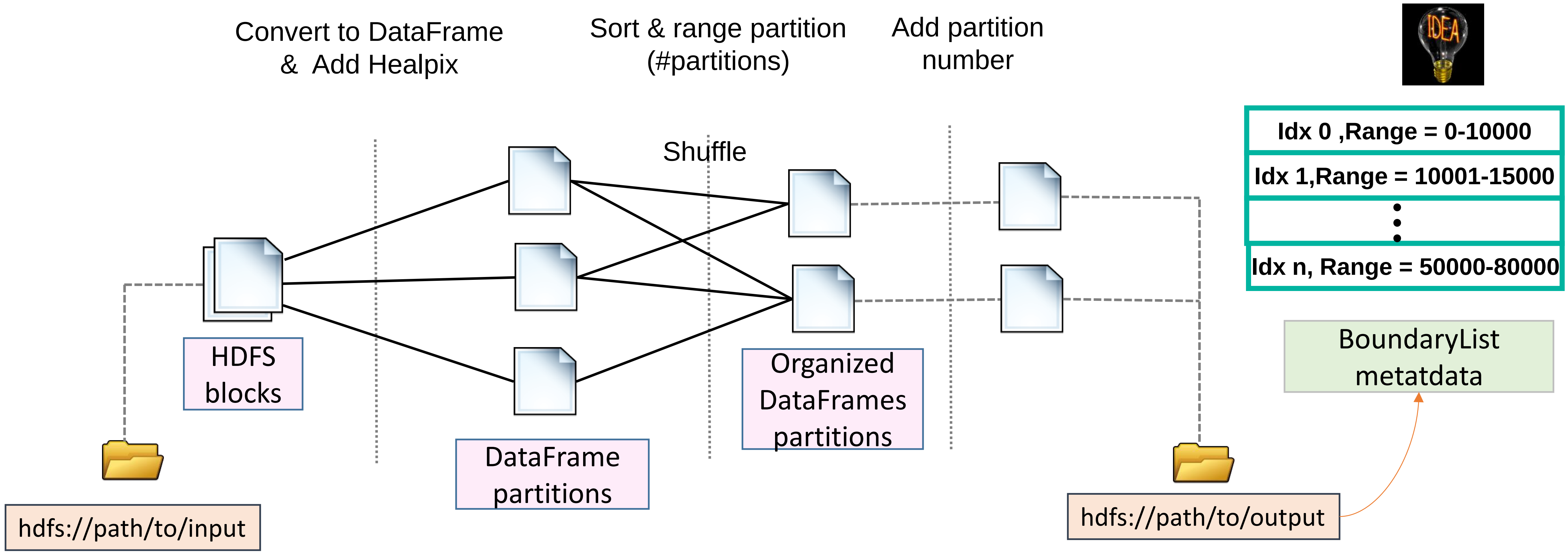


Healpix partition
(NSIDE = 1)



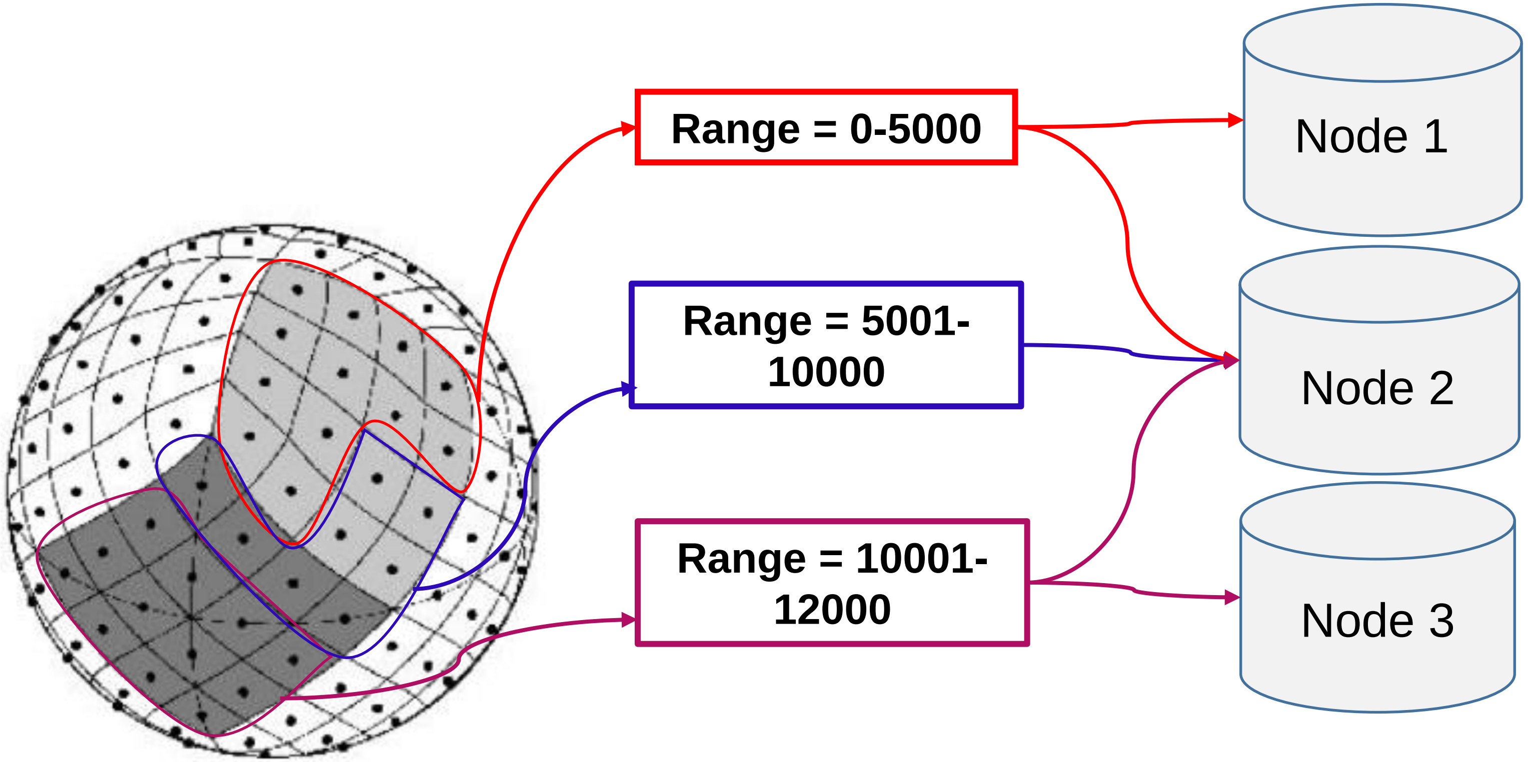
Healpix partition
(NSIDE = 2)

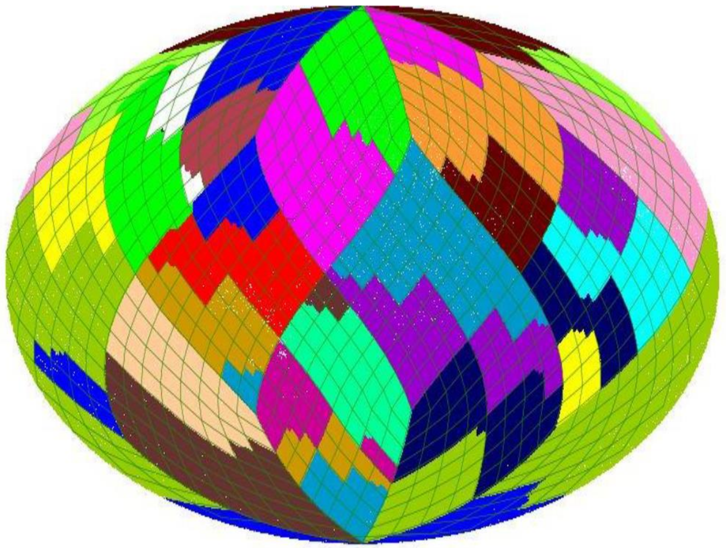
Partitioning Algorithm



#partitions = (InputSize/PartitionSize)(1+α)

Partitioning Result





Partition Visualization with Aladin*

(*) http://aladin.u-strasbg.fr

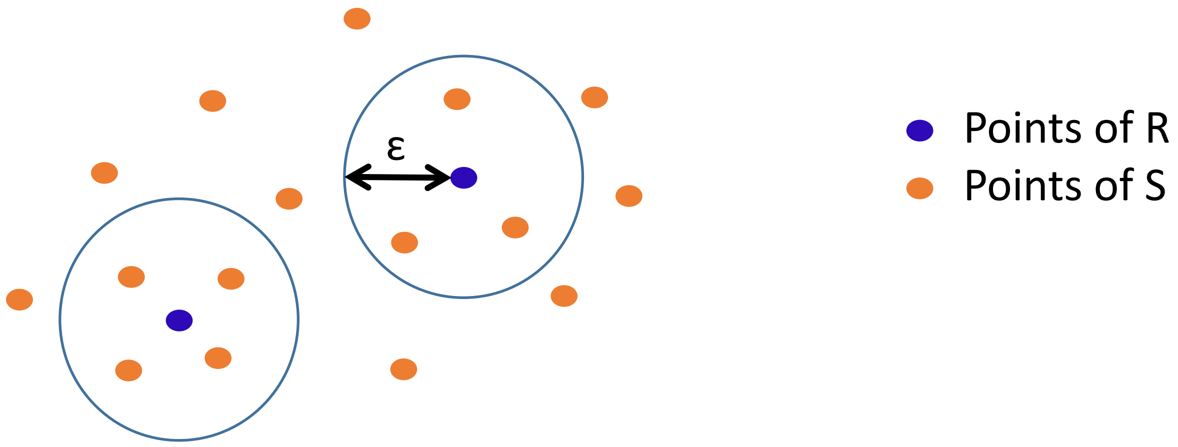
Outline

- Introduction
- AstroSpark Architecture
- Data Patitioning Algorithm
- **Cross-Matching**
- Experimental Evaluation
- Conclusion

Cross-Matching

- Identify and correlate objects belonging to different observations
- Given two sets, *R* and *S*, of data points
- Find all pairs (r,s) ϵ *R*x*S*, such that sphericalDistance(*r*,*s*) ≤ ε.

$$R \ xmatch_{\varepsilon} \ S = \{ \forall (r,s) \in R \times S \mid sphericalDistance(r,s) \leq \varepsilon \}$$



Cross Matching using Spark SQL

```
SELECT *
FROM R JOIN S ON (2 * ASIN(SQRT(SIN(($DEC_2-DEC)/2) *
SIN(($DEC_2-DEC)/2) + COS($DEC_2) * COS(DEC) *
SIN(($RA_2 - RA)/2) * SIN(($RA_2 - RA)/2))) <= ε)
```

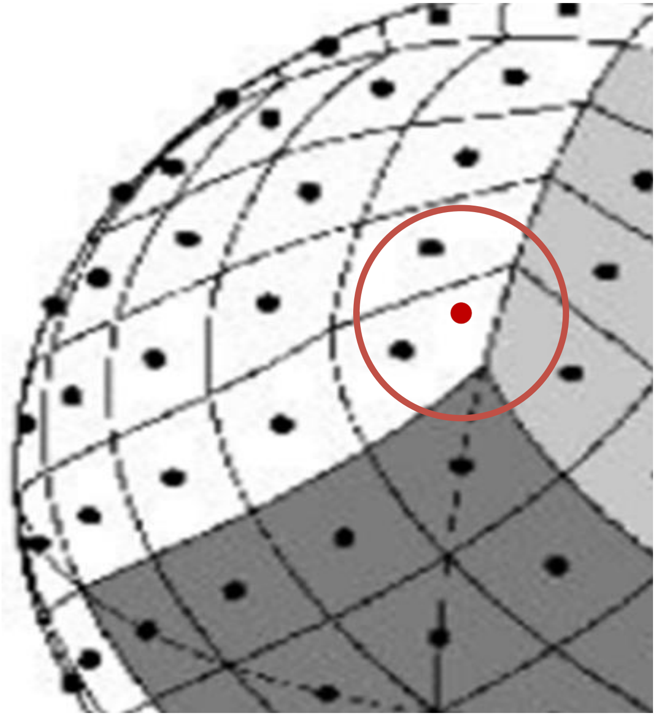
➔Spark: Cartesian product of two input tables

➔Then filters from the Cartesian product based on the distance predicate

- Producing the Cartesian product is costly in Spark

- The execution time of a cross-match between 5 millions of Gaia and all records of Tycho-2 is more than 300 hours (12 days)

Cross Matching using AstroSPARK



<u>Challenge</u>: Comparing vast amount of astronomical objects with low latency

- Limit the comparison to the objects **according to their healpix index**

- But objects on the border of different cells could match.

=> Join should be extended to neighbors !

➔ We propose HX-Match - A Healpix based cross(X)-match

HX-Match - Algorithm

1. **Partitioning** the two input datasets R and S using healpix and range partitioning

2. **Duplicate all objects in S and assign these duplicates the healpix index of each neighbor cell** -> Let's call it **S'**

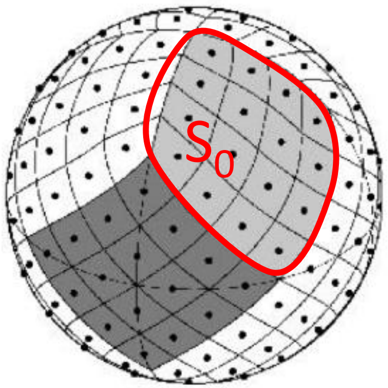
=> comparing candidate objects of S' with R is reduced to a **basic equi-join**.

3. **Equi-join** (R, **S'**) on Healpix indices

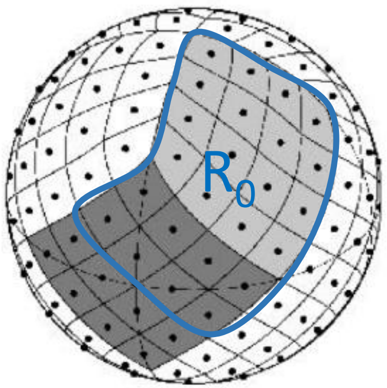
4. **Filter** joined results on the Harvesine formula

HX-MATCH Functioning

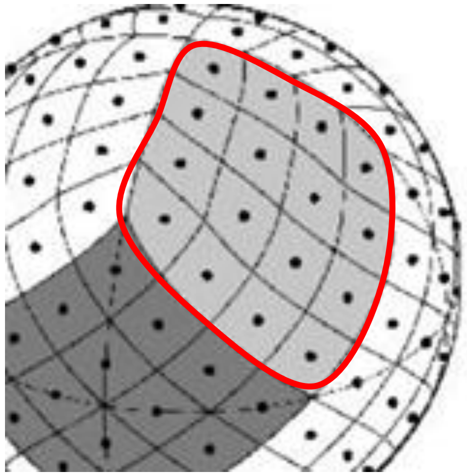
S



R



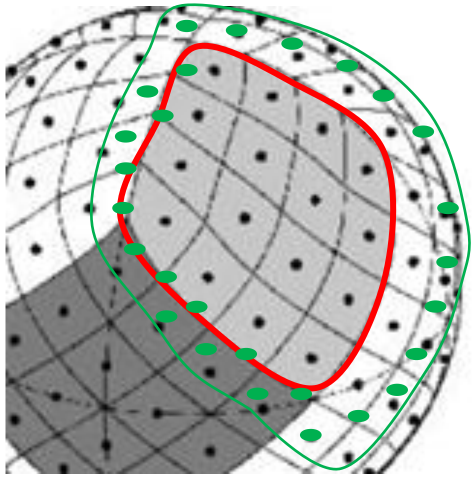
S_0



How to deal with objects along the borders?

3. Augment S_0 with neighbors

S'_0 with neighbors



3. Equi-join with R_0
4. Filter joined results on the Harvesine formula

Implementing HX-Match using SPARK tools

Mainly 3 ways :

1. Extend the **DataFrame API**
2. Use spark strategies to extend the **spark catalyst optimizer**
3. SQL **Query rewriting**

Solution 1 - Extending DataFrame API

- DataFrame is a distributed collection of data organized into named columns.
- DataFrame API is **extended to support HX-MATCH**

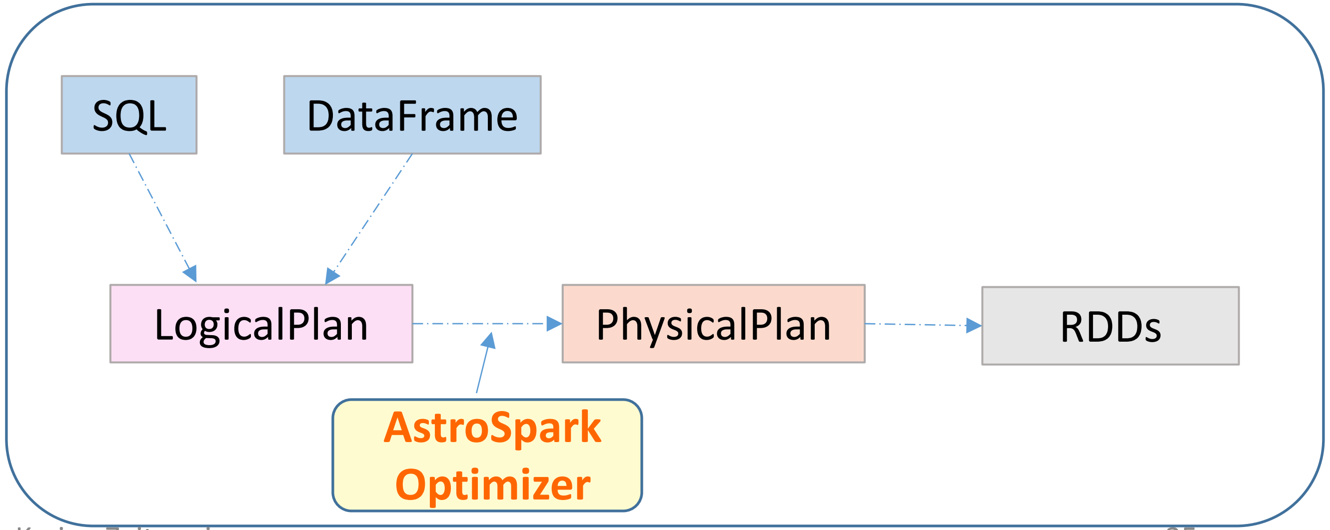
```
DF1.HXMatch(DF2, 2/3600)
```

- ➢ This function will match the current dataframe DF1 with another dataframe DF2 using radius= 2/3600

Solution 2 - Using Spark Strategies

- Extend the query plan optimizer
- Transform a spark **logical plan to an optimized physical plan**
- AstroSpark converts the spark join logical plan **to a list of internal catalyst operations** (SortMergeJoinExec, …) **using strategies**

SELECT * **FROM** gaia
JOIN tycho2 **ON** 1=**CONTAINS** (
POINT('ICRS', gaia.ra, gaia.dec),
CIRCLE('ICRS', tycho2.ra, tycho2.dec, 2/3600))



Solution 3 - Query rewriting

- AstroSpark rewrites the ADQL query to an SQL query
- The ADQL query is parsed, and translated into a Spark SQL expression
  - **Explode** is a built)in spark function and **Neighbours** is a user-defined function

```
SELECT * FROM gaia JOIN
tycho2
ON 1=CONTAINS (
POINT('ICRS', gaia.ra, gaia.dec),
CIRCLE('ICRS', tycho2.ra, tycho2.dec, 2/3600))
```

```
SELECT * FROM gaia JOIN
(SELECT *,explode(Neighbours(ipix)) As ipix_nei FROM tycho2 )
ON (ipix=ipix_nei) WHERE
(SphericalDistance(gaia.ra,gaia.dec,tycho2.ra,tycho2.dec) <2/3600)
```

Outline

- Introduction
- AstroSpark Architecture
- Data Patitioning Algorithm
- Cross-Matching
- **Experimental Evaluation**
- Conclusion

Experimental Setup

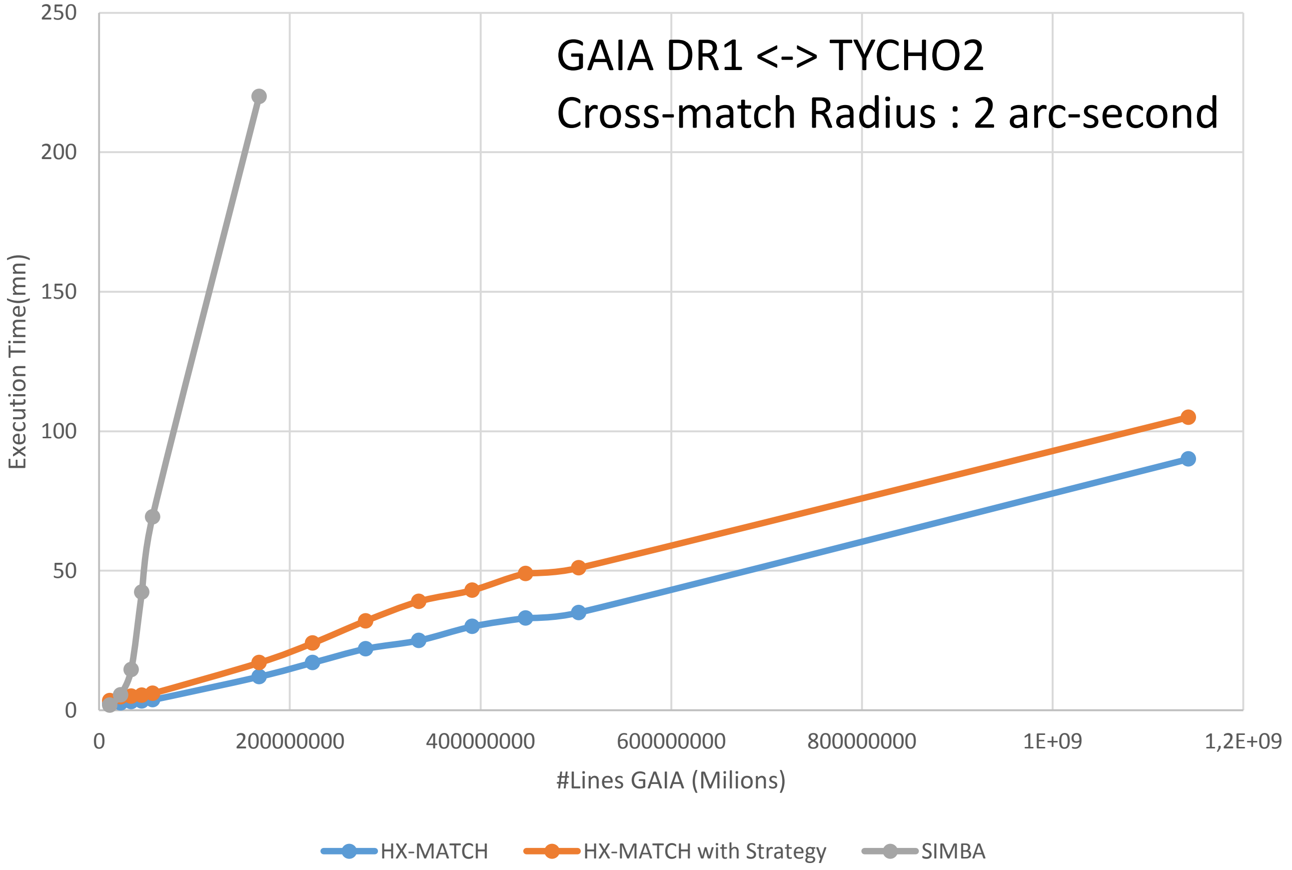
- Environmnent
  - 6 nodes / 180 GB spark main memory/ Partition size: 256 MB
  - Spark 2.0.1 / Hadoop 2.7.2

- Dataset
  - **GAIA DR1**
    - More than 1 billion records, 57 attributes
  - **Tycho-2**
    - 2,5 millions records.

➢ Radius chosen for the cross-match: 2 arc-seconds

Performance of Cross-Matching

HX-Match outperforms SIMBA, a state-of-the-art approach



HX-Match is also 6000 X faster than "plain" Spark SQL

Conclusion and Future Work

Contributions

- Design of AstroSpark, a distributed system based on Spark to process astronomical data.
- Data partitioning with Healpix to speed up query processing
- Implement a cross-matching algorithm based on Spark and Healpix
- Extend the spark 2.0 Catalyst optimizer to implement the query optimizer

Future Work

- Propose other algorithms for NN queries, NN join, histograms, … with ADQL
- Explore other techniques of optimization
  - Cost based optimisation, multi-query (workload) execution , …

➢ **Do not hesitate to challenge us !**

References

Brahem, M., S.Lopes, L.Yeh and K.Zeitouni. **AstroSpark - Towards a Distributed Data Server for Big Data in Astronomy.** ACM SIGSPATIAL PhD Workshop'16.

Brahem, M., L.Yeh and K.Zeitouni , **HX-MATCH: In-Memory Cross-Matching Algorithm for Astronomical Big Data,** to appear in International Symposium on Spatial and Temporal Databases (SSTD'17).

Dong, X. & al. **Simba: Efficient In-Memory Spatial Analytics.** SIGMOD 2016.

Eldawy, A., & Mokbel, M. F. **A demonstration of SpatialHadoop: an efficient mapreduce framework for spatial data.** VLDB 2013.

Gorski, K.M., & al. **HEALPix: a framework for high-resolution discretization and fast analysis of data distributed on the sphere.** The Astrophysical Journal 622.2 (2005): 759.

Nishimura, S., & al **MD-HBase: design and implementation of an elastic data infrastructure for cloud-scale location services.** Distributed and Parallel Databases 31.2 (2013): 289-319.

Yu, J., & al. **Geospark: A cluster computing framework for processing large-scale spatial data.** In Proceedings of the 23rd SIGSPATIAL International, page 70. ACM, 2015.