Improving astronomical online services with Apache Spark and Docker

André Schaaff, François-Xavier Pineau
Centre de Données astronomiques de Strasbourg
Noémie Wali, Paul Trehiou
Université de technologie de Belfort-Montbéliard
Julien Nauroy
Direction Informatique, Université de Paris-Sud, Orsay

ADASS XXVI, Trieste, 2016

CENTRE DE DONNÉES ASTRONOMIQUES DE STRASBOURG

IVOA

Asterics
Astronomy ESFRI & Research Infrastructure Cluster

Outline

Context

Apache Spark and Docker in 90’’

Motivation and use case

The data and the « cross-match » service

Test beds

First experiment and what we have learned

On-going work and perspectives

Context

A continuous exploration of new technologies, especially in the “Big Data” field

How to face the data revolution in frontend services which need to be interactive ?

How to maintain and resize quickly the backend ?

Spark in 60''

- "Apache Spark is a cluster computing platform designed to be fast and general purpose."
- It extends the MapReduce model to support more types of computations (interactive queries, stream processing, etc.) and it offers APIs for Scala, Java, Python, R,...
- Important feature: computations in memory (as much as possible)
  - Introduction of data models
    - RDD (Resilient Distributed Datasets) to store objects
    - Datasets to represent tabular data, queryable via SQL
- It can use Hadoop Distributed File System (HDFS).

Docker in 30’’

- On Docker website: “Build, Ship, Run”
  - Build
    - Embed only what you need in a component (and use existing components !).
  - Ship
    - To an another machine or on a registry (to make it available for others).
  - Run
    - A component and the host share the same Operating System but following rules, restrictions, etc. (security...).

    (a Virtual Machine is a whole OS emulation on a host)

Motivation & use case

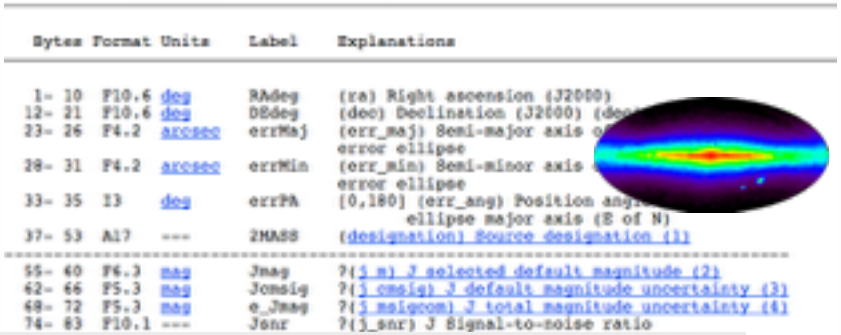
- Evaluation of Spark in the frame of a use case, the "cross-match" of source catalogues:
  - Improvement of the existing service ?

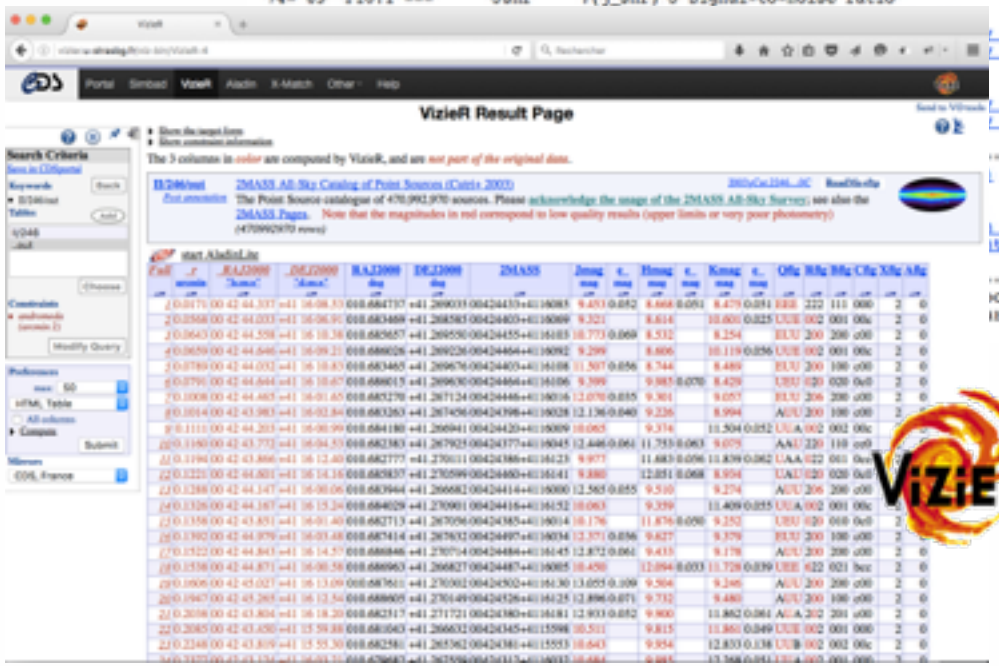
    1 server, 2x6 cores, 32GB, 12TB (15k tours) when we done the main test

    Now: 2x10 cores, 64GB with the same disks
  - Up to scale capability (data volumes, hardware, deployments (Docker ?), etc.) ?
  - Which cost (€, manpower, performances) ?
- Back thought: "bring the code to the data"

The data

- Source catalogues (>10,000 available)
- Examples (number of sources):
  - 2MASS[1], 470,992,970
  - SDSS[2] DR9, 469,053,874





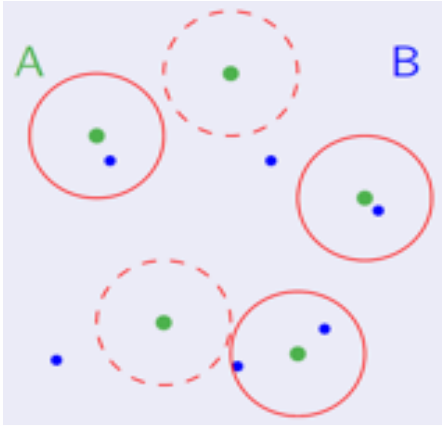
Example of a ReadMe file associated to 2MASS source catalogues available through the VizieR service

[1]2MASS, Two Micron All Sky Survey,
[2]SDSS, Sloan Digital Sky Survey

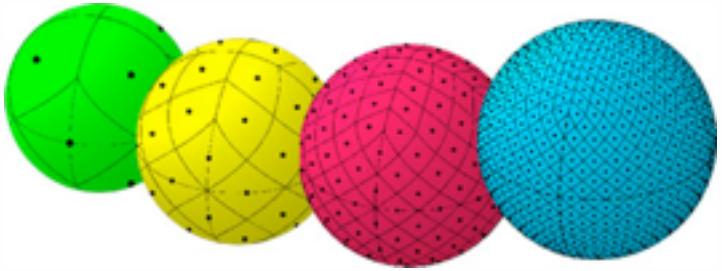
…and the CDS "cross-match" service

- The "cross-match" service does a cross correlation of sources between (very) large catalogues (current size: 10^9).

Fuzzy join between 2 tables (A and B)
of several hundred millions of data



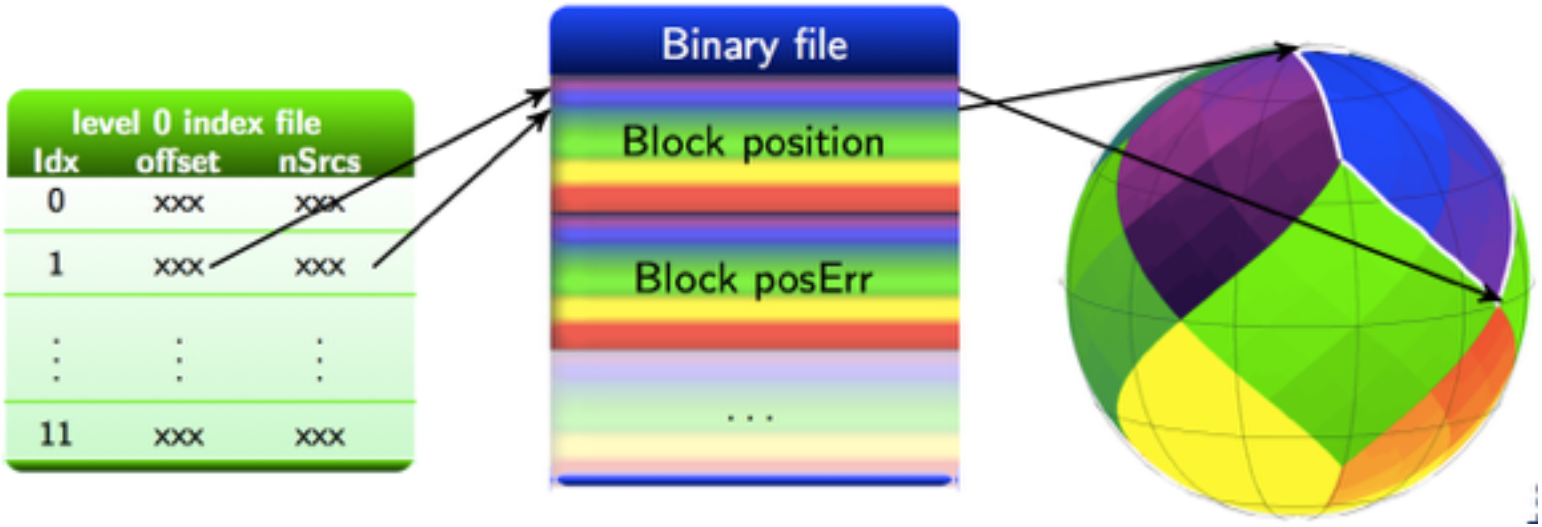
- Which area ?
  - Full sky: all the sources
  - A cone: only the sources which are at a certain angular distance from a given position
  - A HEALPix cell



Credits: http://healpix.jpl.nasa.gov/

…and the CDS “cross-match” service (2)

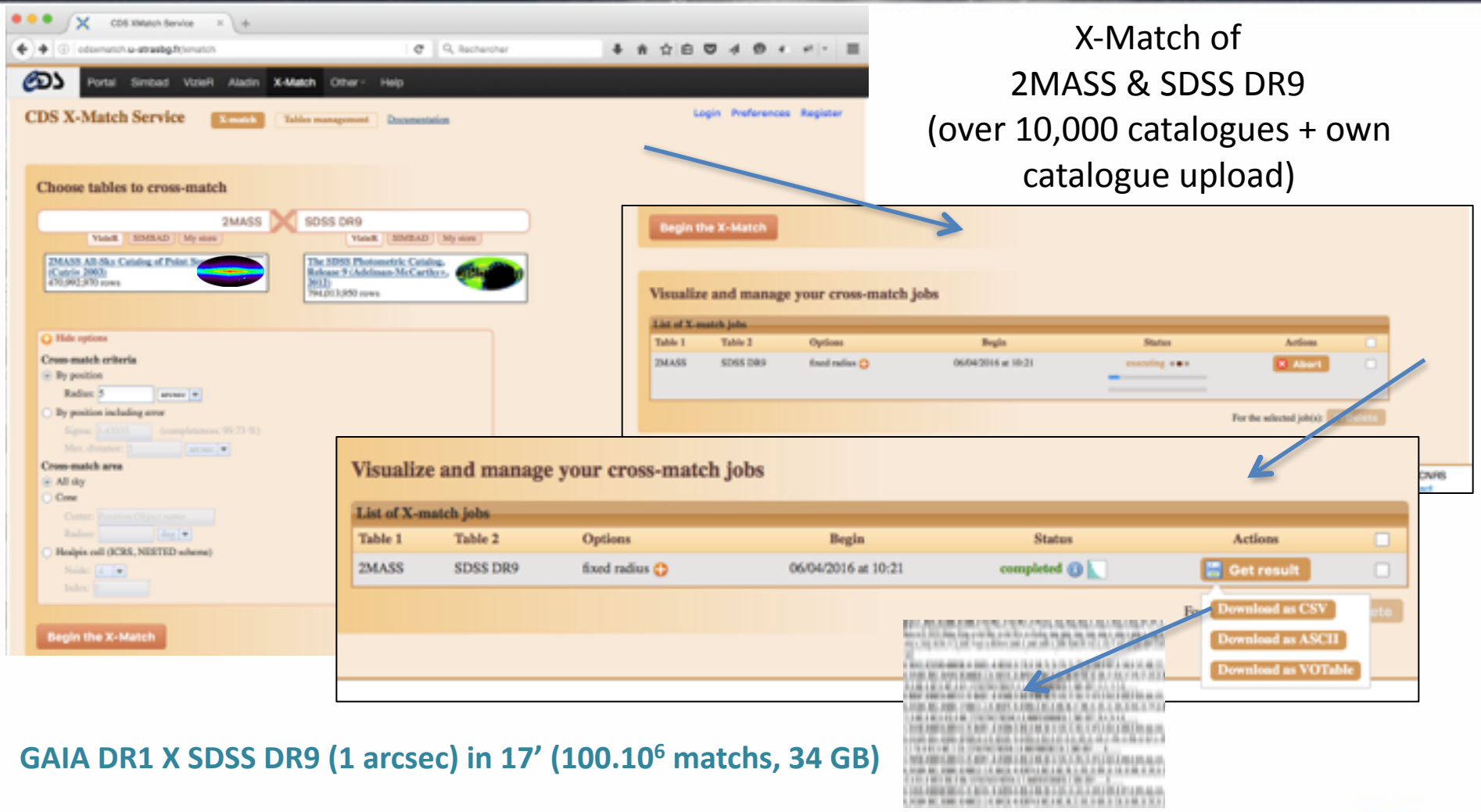
- Data is not distributed but organised and stored on one server



The sky is cut into diamonds of the same size, pixels, each source or sky object is a numbered pixel.

…and the CDS “cross-match” service (3)

X-Match of
2MASS & SDSS DR9
(over 10,000 catalogues + own catalogue upload)



GAIA DR1 X SDSS DR9 (1 arcsec) in 17’ (100.10^6 matchs, 34 GB)

Test beds: hardware & software

- Internal resources to test
  - 6 physical nodes (4 cores, 16GB, 1 TB), Ubuntu 16.04LTS
- Renting of external resources
  - Cluster1: 12 physical nodes, 4 cores, 32GB, Raid 2*2TB, Ubuntu 14.04LTS (8000€ / year)
  - Configuration was defined “ad hoc” and low cost
  - Next one under definition (probably through collaborations)
- Software side:
  - Apache distributions of Spark (1.5.0 to 2.0.1) and Hadoop (2.6 to 2.7.3)
  - Java, Scala

First experiment (SDSS DR7 X 2MASS)

Data preparation phase

Input files to HDFS and loading into 2 RDDs (information about an object in the Sky)

Each RDD is transformed in a pairRDD (key = "source pixel number", value = "all the information whose the source (ra, dec) ")

PairRDD distribution over the nodes (hashpartionning => grouping elements having the same key (same pixel number) in the same partition)

Elements with the same key are on the same node, distribution is essential to the join phase

PairRDDs are stored into HDFS as binary files preserving the structure (Key, Value)

First experiment (2)

Join phase

Loading in two PairRDDs + duplication* of some sources in the neighbour pixels in one of it

*same method than for TOPCAT (M.Taylor)

PairRDDs joined following the Key into a new PairRDD where the elements are (Key, Value1, Value2) triples

Join done following the Key (cell number), 2 near sources can be in the different cells and are not joined (=> duplication* of sources in the neighbour cells to avoid the side effects)

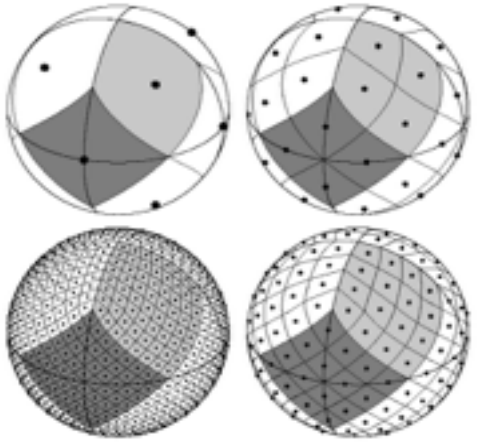
*a circle with a fixed radius is drawn around the source, If neighbour pixels are partially in this circle, the source is then duplicated in the neighbour cells

The joined elements are then filtered (distance between the 2 sources < a given threshold)

Final result stored in HDFS (in a text format for a later visualization and use)

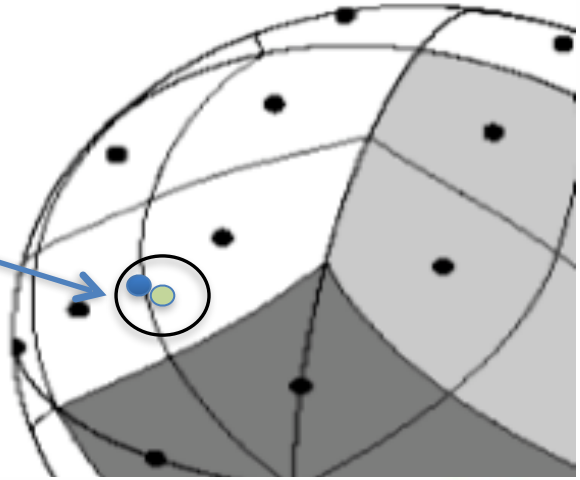
Illustration

- A X-Match implementation in MapReduce, couples (Key = pixel number, Value)



HEALPix sky cutting

- Side effects
  - Fuzzy join
  - Source duplication in the neighbour cells if needed



Credits: HEALPix – arXiv:astro-ph/0409513

First experiment result (Cluster1)

- Input data (SDSS DR7 (primary sources) and 2MASS): 54GB and 58GB file size; 357 175 411 and 470 992 970 elements
- Output data: 49 208 820 elements

X-Match service reference time was: 10 minutes

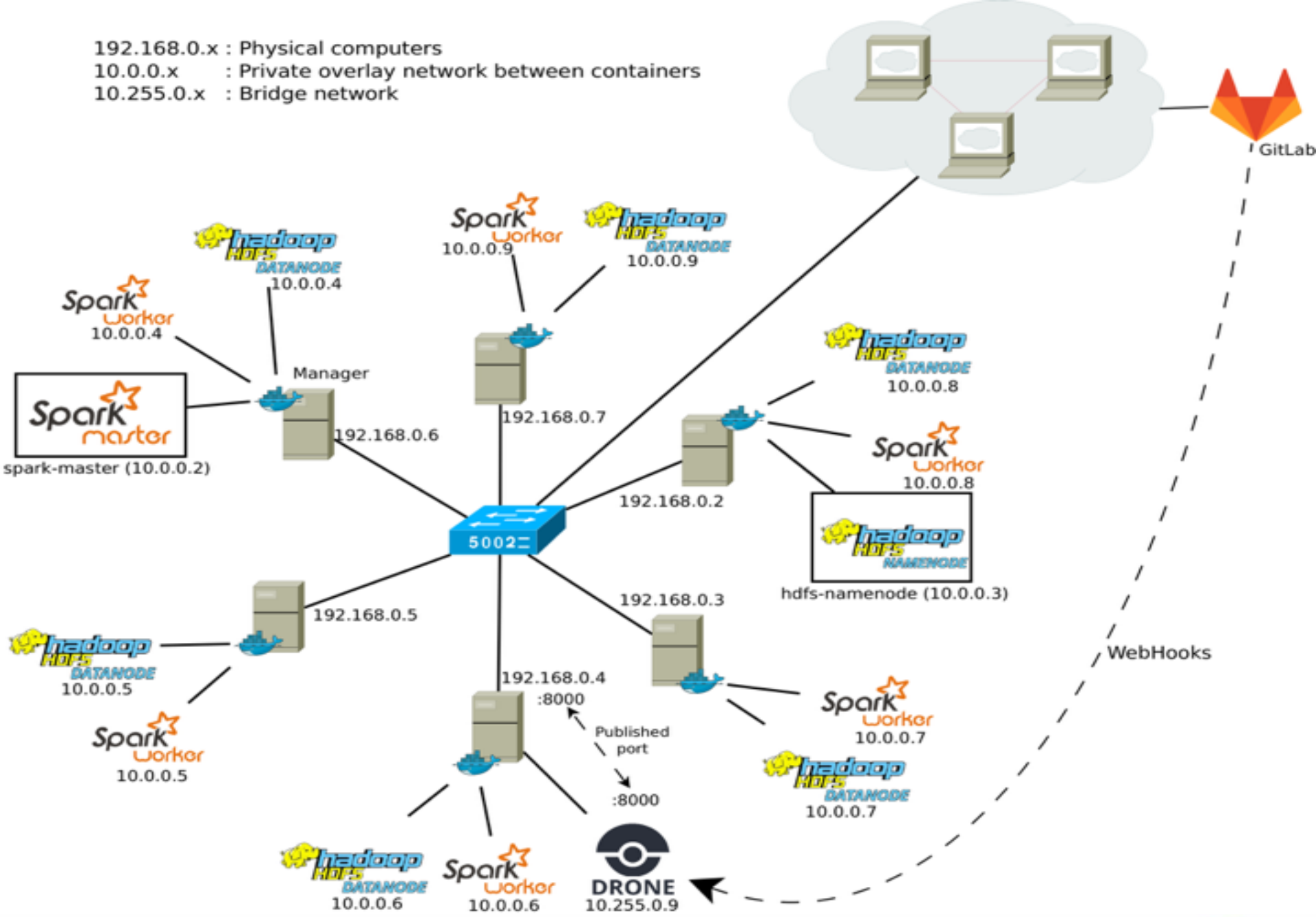
Cross-Match (source duplication done in phase 2 with all the data as output)					
HDFS block size= 128MB for the input files ; sdss7.csv and t 2mass.csv replicated 2 times					
HashPartitioner	**60 partitions**				
HDFS output files size	**32MB**				
Number of nodes Spark/HDFS	5	7	9	10	11
Phase 1: prepare	**23,0**	**16,0**	**14,0**	**14,0**	**13,0**
mapToPair (sdss7.csv)	5,1	4,9	4,9	4,8	4,7
saveAsHadoopFile (sdss7.bin)	5,7	2,7	2,0	2,3	1,5
mapToPair (2mass.csv)	5,7	5,2	5,2	5,1	5,0
saveAsHadoopFile (2mass.bin)	6,5	3,6	1,9	1,6	[illegible]
Phase 2: join	**31,0**	**21,0**	**13,0**	**11,0**	**9,9**
mapToPair (sdss7.bin)	7,2	4,7	3,5	3,0	[illegible]
flatMapToPair (2mass.bin)	11,8	8,3	5,5	4,9	4,3
saveAsTextFile (crossMatch_D.txt)	12,0	7,6	3,4	2,4	2,3
TOTAL	**54,0**	**37,0**	**27,0**	**25,0**	**22,9**

What we have learned

- Time was similar to the X-Match service from 11 nodes but
  - Keys common to 2 RDDs are not necessarily on the same node
    - It implies a transfer overhead between the nodes during the join => impact on the performances
    - We had clearly a bottleneck in the join phase ("shuffle")
    - "block affinity groups" is an on-going work at Apache.
  - We spent time on the "data co-location", tests were also done on an another Spark implementation by a colleague from Université Paris-Sud => we found no Spark solution.
  - We found a solution to do it "manually" via scripts.

On-going work

- Introduction of Docker and Drone (continuous integration) to “automate” the process and to focus mainly on the development side. It is becoming easy to migrate to external resources when needed.
- Use of Scala which is native in Spark (a part of the Java API is “experimental”).
- Sharing of our experiments (in/outside the community).
- G. Landais (CDS) and L. Michel (SSC XMM-Newton) joined recently the effort


192.168.0.x : Physical computers
10.0.0.x : Private overlay network between containers
10.255.0.x : Bridge network
GitLab
WebHooks
Spark Worker 10.0.0.9
hadoop HDFS DATANODE 10.0.0.9
hadoop HDFS DATANODE 10.0.0.4
Spark Worker 10.0.0.4
Manager
Spark master
spark-master (10.0.0.2)
192.168.0.6
192.168.0.7
hadoop HDFS DATANODE 10.0.0.8
Spark Worker 10.0.0.8
192.168.0.2
hadoop HDFS NAMENODE
hdfs-namenode (10.0.0.3)
5002
192.168.0.5
hadoop HDFS DATANODE 10.0.0.5
Spark Worker 10.0.0.5
192.168.0.3
192.168.0.4 :8000
Published port
:8000
Spark Worker 10.0.0.7
hadoop HDFS DATANODE 10.0.0.7
hadoop HDFS DATANODE 10.0.0.6
Spark Worker 10.0.0.6
DRONE 10.255.0.9

Perspectives

X-Match service reference time is now 7 minutes !

- What we expect:
  - Significant improving of the performances, with a reasonable hardware cost.
  - Re-use of the Docker and continuous integration experience, apply it to other services like VizieR for the mirror maintenance (to be evaluated).
- Evaluation (and comparison) of other technologies like Spark and Docker, minimize as much as possible the dependency to a specific one.
- Implement a prototype allowing a user “to move his code to the data”.

Submission deadline: 30/11/2016
http://cds.unistra.fr/meetings/Lisa8/

Links

- Apache Spark, http://spark.apache.org/
- Apache Hadoop, http://hadoop.apache.org/
- Spark : Cluster Computing with Working Sets, Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, Ion Stoica, University of California, Berkeley, http://static.usenix.org/legacy/events/hotcloud10/tech/full_papers/Zaharia.pdf
- Optimizing Shuffle Performance in Spark, Aaron Davidson, Andrew Or, UC Berkeley, http://www.cs.berkeley.edu/~kubitron/courses/cs262a-F13/projects/reports/project16_report.pdf
- Resilient Distributed Datasets : A Fault-Tolerant Abstraction for In-Memory Cluster Computing, Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, Ion Stoica, University of California, Berkeley, https://www.cs.berkeley.edu/~matei/papers/2012/nsdi_spark.pdf
- JavaSpark Api, http://spark.apache.org/docs/latest/api/java/
- HEALPix, http://healpix.jpl.nasa.gov/