

Spark & Docker experiment status update

Talk & discussion by F.-X. Pineau & A. Schaaff

André Schaaff, François-Xavier Pineau
Centre de Données astronomiques de Strasbourg

Noémie Wali, Paul Trehou
Université de technologie de Belfort-Montbéliard

Julien Nauroy
Direction Informatique, Université de Paris-Sud, Orsay

IVOA, Trieste, 2016



□ Outline

Apache Spark and Docker in 90''

Motivation and use case

The data and the « cross-match » service

The dream !

Test beds

First experiment and what we have learned

On-going work and perspectives

□ 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

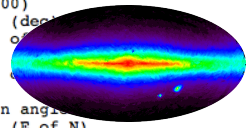
- Evaluation of **Spark** in the frame of a **use case**, the “**cross-match**” of **source catalogues**:
 - Improvement of the existing service ?
 - 1 server, 2x6 cores, 32GB, 12TB (15k tours) when we done the main test
 - Now: 2x10 cores, 64GB with the same disks
 - Up to scale capability (data volumes, 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):
 - 2MASS¹, 470,992,970
 - SDSS² DR9, 469,053,874

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

Bytes	Format	Units	Label	Explanations
1- 10	F10.6	deg	RAdeg	(ra) Right ascension (J2000)
12- 21	F10.6	deg	DEdeg	(dec) Declination (J2000) (deg)
23- 26	F4.2	arcsec	errMaj	(err_maj) Semi-major axis of error ellipse
28- 31	F4.2	arcsec	errMin	(err_min) Semi-minor axis of error ellipse
33- 35	I3	deg	errPA	(err_ang) Position angle of ellipse major axis (E of N)
37- 53	A17	---	2MASS	(designation) Source designation (1)
55- 60	F6.3	mag	Jmag	?(j m) J selected default magnitude (2)
62- 66	F5.3	mag	Jcmsig	?(j cmsig) J default magnitude uncertainty (3)
68- 72	F5.3	mag	e_Jmag	?(j msigcom) J total magnitude uncertainty (4)
74- 83	F10.1	---	Jsnr	?(j_snr) J Signal-to-noise ratio



VizieR Result Page

The 3 columns in color are computed by VizieR, and are *not part of the original data*.

II/246/out 2MASS All-Sky Catalog of Point Sources (Cutri+ 2003) 2003yCat.2246...DC ReadMe=ft

The Point Source catalogue of 470,992,970 sources. Please acknowledge the usage of the 2MASS All-Sky Survey; see also the 2MASS Pages. Note that the magnitudes in red correspond to low quality results (upper limits or very poor photometry) (470992970 rows)

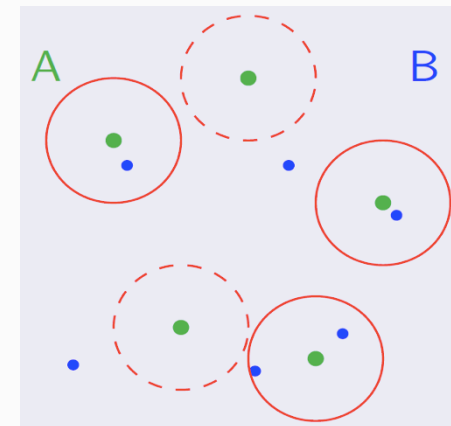
Full	r	RAJ2000	DEJ2000	RAJ2000	DEJ2000	2MASS	Jmag	e	Hmag	e	Kmag	e	Oflg	Rflg	Bflg	Cflg	Xflg	Aflg
deg	arcmin	"hms"	"dms"	deg	deg	mag	mag	mag	mag	mag	mag	mag						
1	0.0171	00 42 44.337	+41 16 08.53	010.684737	+41.269035	00424433+4116085	9.453	0.052	8.668	0.051	8.475	0.051	EEB	222	111	000	2	0
2	0.0568	00 42 44.033	+41 16 06.91	010.683469	+41.268585	00424403+4116069	9.321		8.614		10.601	0.025	UEE	002	001	00c	2	0
3	0.0643	00 42 44.558	+41 16 10.38	010.685657	+41.269550	00424455+4116103	10.773	0.069	8.532		8.254		EUU	200	200	c0c	2	0
4	0.0659	00 42 44.646	+41 16 09.21	010.686026	+41.269226	00424464+4116092	9.299		8.606		10.119	0.056	UAU	002	001	00c	2	0
5	0.0789	00 42 44.032	+41 16 10.83	010.683465	+41.269630	00424403+4116108	11.507	0.056	8.744		8.489		EUU	200	100	c0c	2	0
6	0.0791	00 42 44.644	+41 16 10.67	010.686015	+41.269630	00424464+4116106	9.399		9.985	0.070	8.429		UEU	020	020	0c0	2	0
7	0.1008	00 42 44.465	+41 16 01.65	010.685270	+41.267124	00424464+4116016	12.070	0.035	9.301		9.057		EUU	206	200	c0c	2	0
8	0.1014	00 42 43.983	+41 16 02.84	010.683263	+41.267456	00424398+4116028	12.136	0.040	9.226		8.994		AUU	200	100	c0c	2	0
9	0.1111	00 42 44.203	+41 16 00.99	010.684180	+41.266941	00424420+4116009	10.065		9.374		11.504	0.052	UAU	002	002	00c	2	0
10	0.1160	00 42 43.772	+41 16 04.53	010.682383	+41.267925	00424377+4116045	12.446	0.061	11.753	0.063	9.075		AAU	220	110	cc0	2	0
11	0.1194	00 42 43.866	+41 16 12.40	010.682777	+41.270111	00424386+4116123	9.977		11.683	0.056	11.839	0.062	UAA	022	011	cc0	2	0
12	0.1221	00 42 44.601	+41 16 14.16	010.685837	+41.270599	00424460+4116141	9.880		12.051	0.068	8.934		UAU	020	020	0c0	2	0
13	0.1288	00 42 44.147	+41 16 00.06	010.683944	+41.266682	00424416+4116000	12.565	0.055	9.510		9.274		AUU	206	200	c0c	2	0
14	0.1326	00 42 44.167	+41 16 15.24	010.684029	+41.270901	00424416+4116152	10.063		9.359		11.409	0.055	UAU	002	001	00c	2	0
15	0.1358	00 42 43.851	+41 16 01.40	010.682713	+41.267056	00424385+4116014	10.176		11.876	0.050	9.252		UEU	020	010	cc0	2	0
16	0.1392	00 42 44.979	+41 16 03.48	010.687414	+41.267632	00424497+4116034	12.371	0.036	9.627		9.379		BEU	200	100	c0c	2	0
17	0.1522	00 42 44.843	+41 16 14.57	010.686846	+41.270714	00424484+4116145	12.872	0.061	9.433		9.178		AUU	200	200	c0c	2	0
18	0.1538	00 42 44.871	+41 16 00.58	010.686963	+41.266827	00424487+4116005	10.450		12.094	0.033	11.728	0.039	UEE	622	021	bcc	2	0
19	0.1606	00 42 45.027	+41 16 13.09	010.687611	+41.270302	00424502+4116130	13.055	0.109	9.504		9.246		AUU	200	200	c0c	2	0
20	0.1947	00 42 45.265	+41 16 12.54	010.688605	+41.270149	00424526+4116125	12.896	0.071	9.732		9.480		AUU	200	100	c0c	2	0
21	0.2038	00 42 43.804	+41 16 18.20	010.682517	+41.271721	00424380+4116181	12.933	0.052	9.900		11.862	0.061	UAU	022	201	c0c	2	0
22	0.2085	00 42 43.450	+41 15 59.88	010.681043	+41.266632	00424345+4115598	10.511		9.815		11.861	0.049	UEE	002	001	00c	2	0
23	0.2248	00 42 43.819	+41 15 55.30	010.682581	+41.265362	00424381+4115553	10.643		9.954		12.833	0.138	UUB	002	002	00c	2	0
24	0.2372	00 42 43.124	+41 16 03.31	010.679682	+41.263550	00424312+4116032	10.684		9.085		12.268	0.051	UUA	002	001	00c	2	0

¹2MASS, Two Micron All Sky Survey,
²SDSS, Sloan Digital Sky Survey

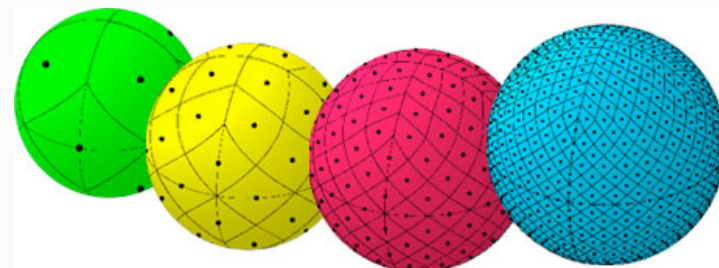
□ ...and the CDS “cross-match” service

- 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



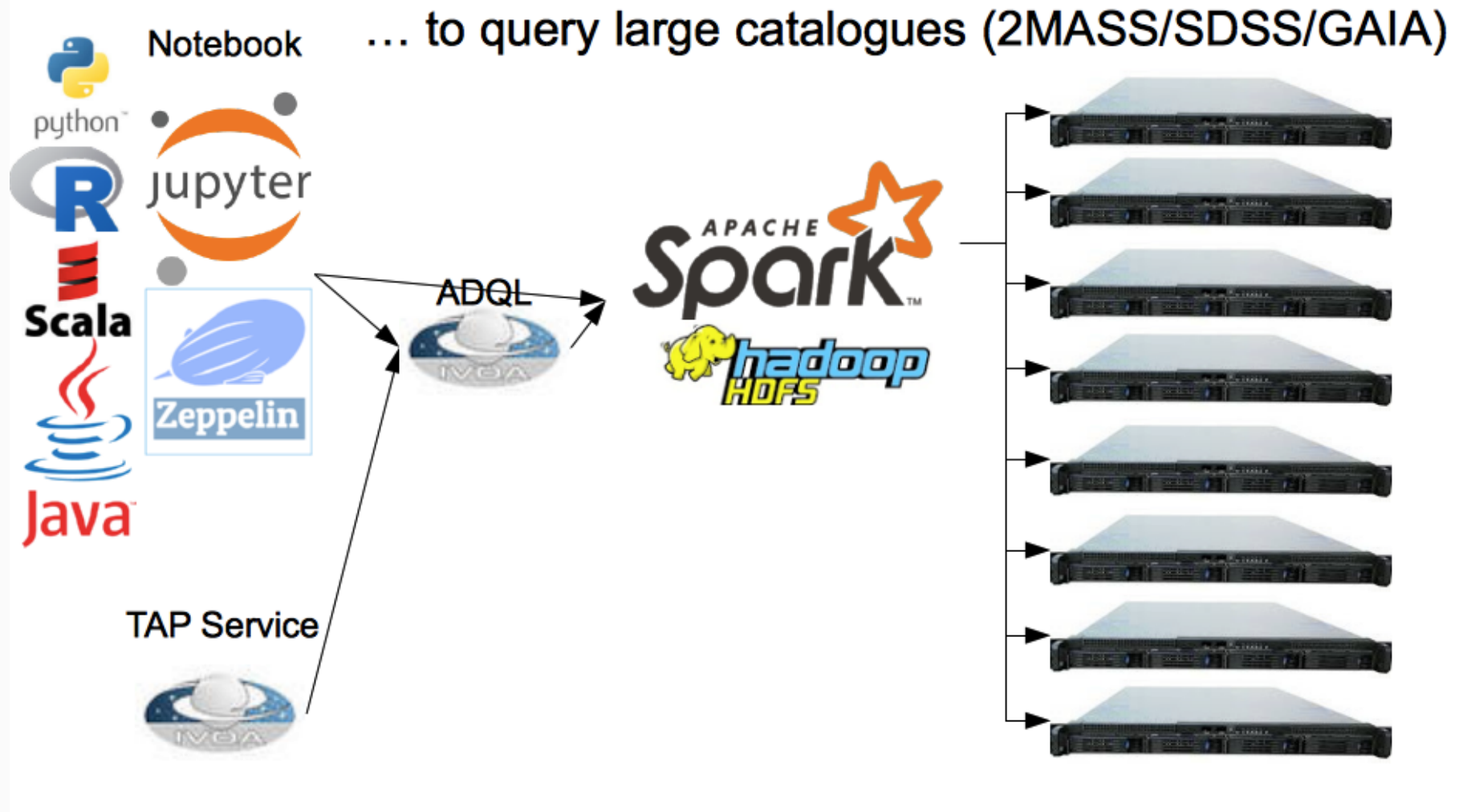
Credits: <http://healpix.jpl.nasa.gov/>

...and the CDS "cross-match" service (3)

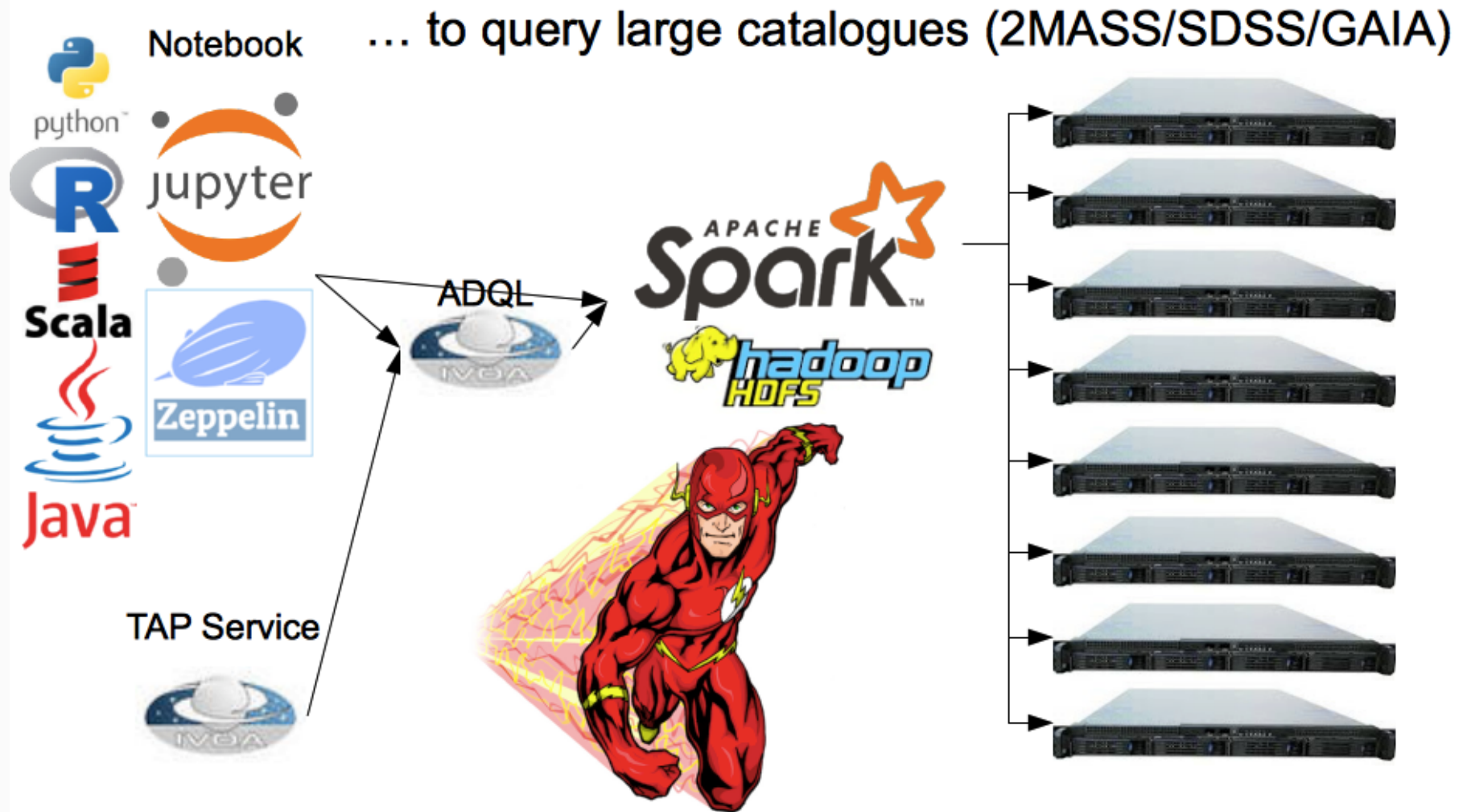
X-Match of 2MASS & SDSS DR9 (over 10,000 catalogues + own catalogue upload)

GAIA DR1 X SDSS DR9 (1 arcsec) in 17' (100.10⁶ matches, 34 GB)

□ We have a dream...



□ We have a dream...





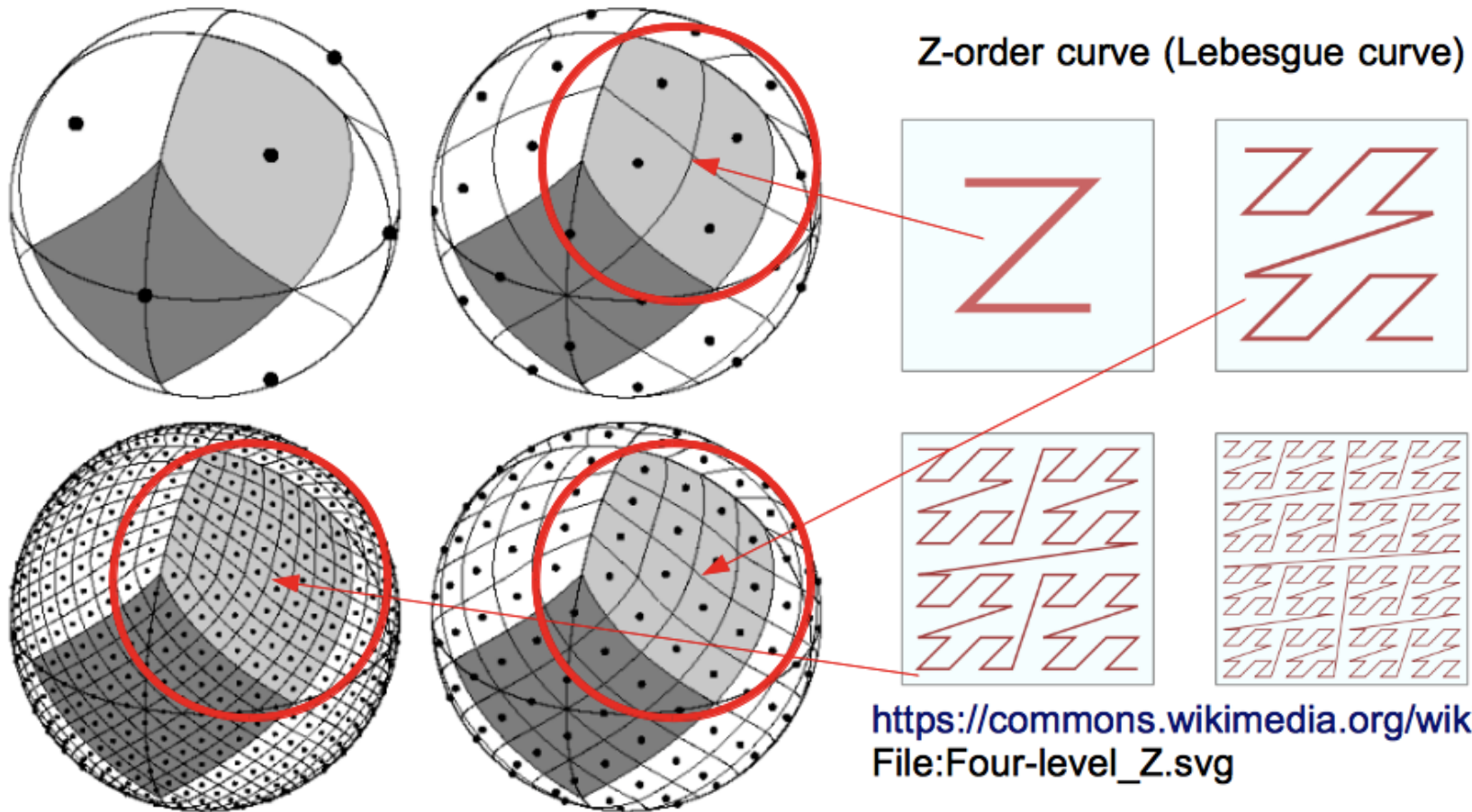
□ Challenge

How to distribute the data on nodes to achieve best possible performances?



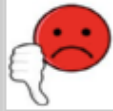


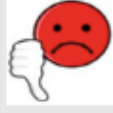






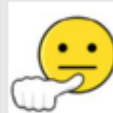
□ Sky partitioning scheme

Gorski et al. (2004)

HEALPix: 12 z-order curves on the sphere



□ Sky partitioning for Map/Reduce

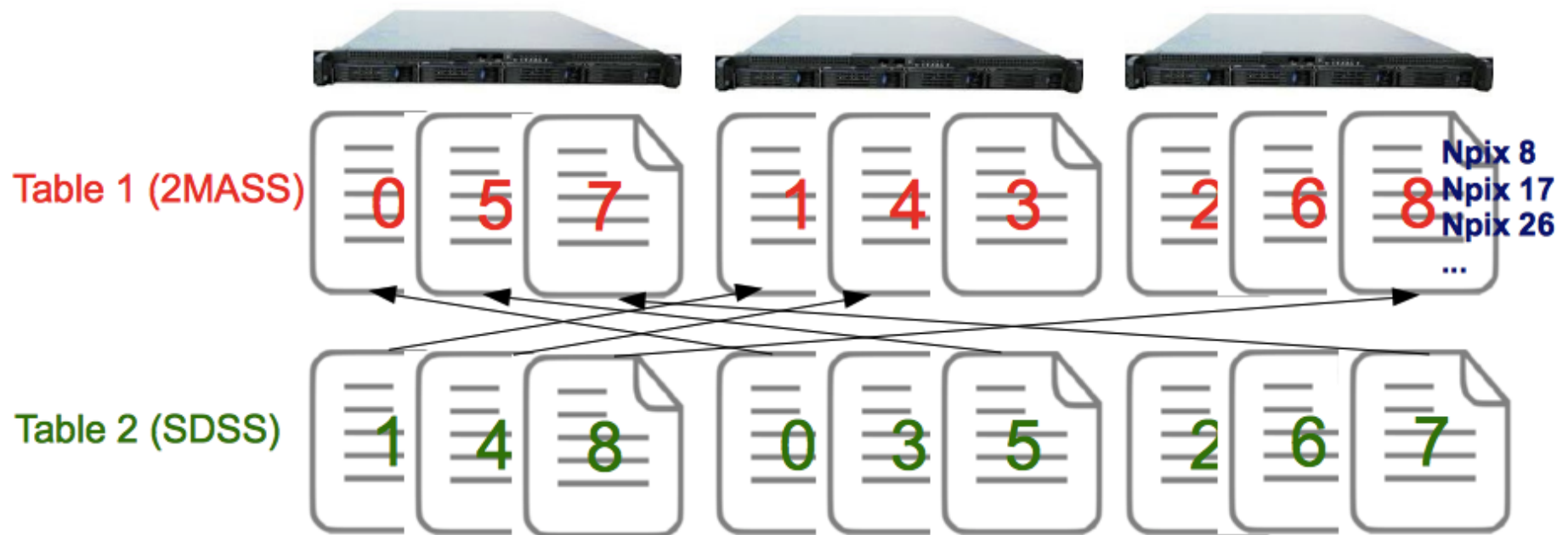
	Non positional queries, color-color diagrams, ...	Simple (large) Cone Search	Simple Xmatch (Known radius)
Default (random)			 Large data transfers
Range partitioner on HEALPix npix		 Unbalanced	 Large data transfers
Hash partitioner on large HEALPix npix	 Unbalanced	 Unbalanced	 Unbalanced++  Limited data transfers
Hash partitioner on small HEALPix npix (few xmatch radius)			 Limited data transfers

To be tested in practice!
(we consider here only one simple hardware configuration)

□ Data partitioning with Spark/HDFS

Toy example: 3 machines, 9 partitions

HahPartitioner on HEALPix: partition number i contains pixels $\text{npix} \% 9 == i$



- Same partitioning scheme (**co-partitioning**) :)
- Partitions on different nodes (**no co-location**): no way to tell spark to put same partition numbers on same machines!! :(

□ Data partitioning with Spark/HDFS

Solution:

- switch off HDFS
- move manually the partitions
- switch on HDFS



□ 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 (**hashpartitioning** => **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



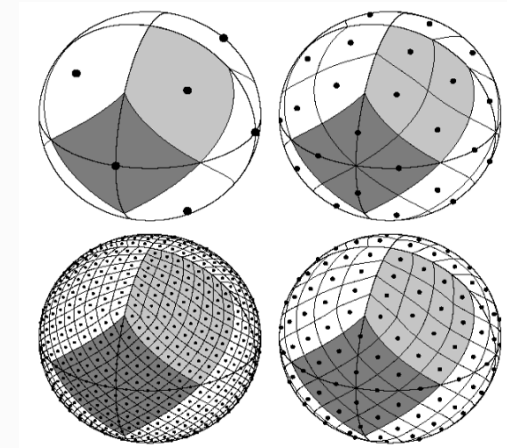
The joined elements are then filtered (distance between the 2 sources < a given threshold)



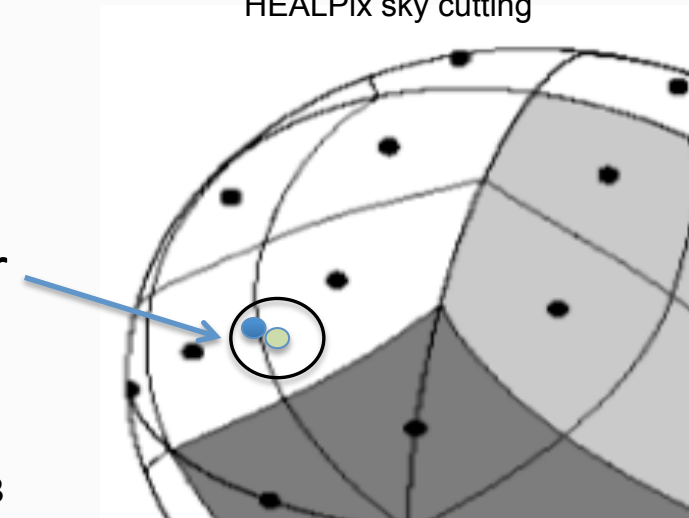
Final result stored in HDFS (in a text format for a later visualization and use)

□ Illustration

- A X-Match implementation in MapReduce, couples (Key = pixel number, Value)
- Side effects
 - Fuzzy join
 - Source duplication in the neighbour cells if needed



HEALPix sky cutting



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	1,4
Phase 2: join	31,0	21,0	13,0	11,0	9,9
mapToPair (sdss7.bin)	7,2	4,7	3,5	3,0	2,5
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.**

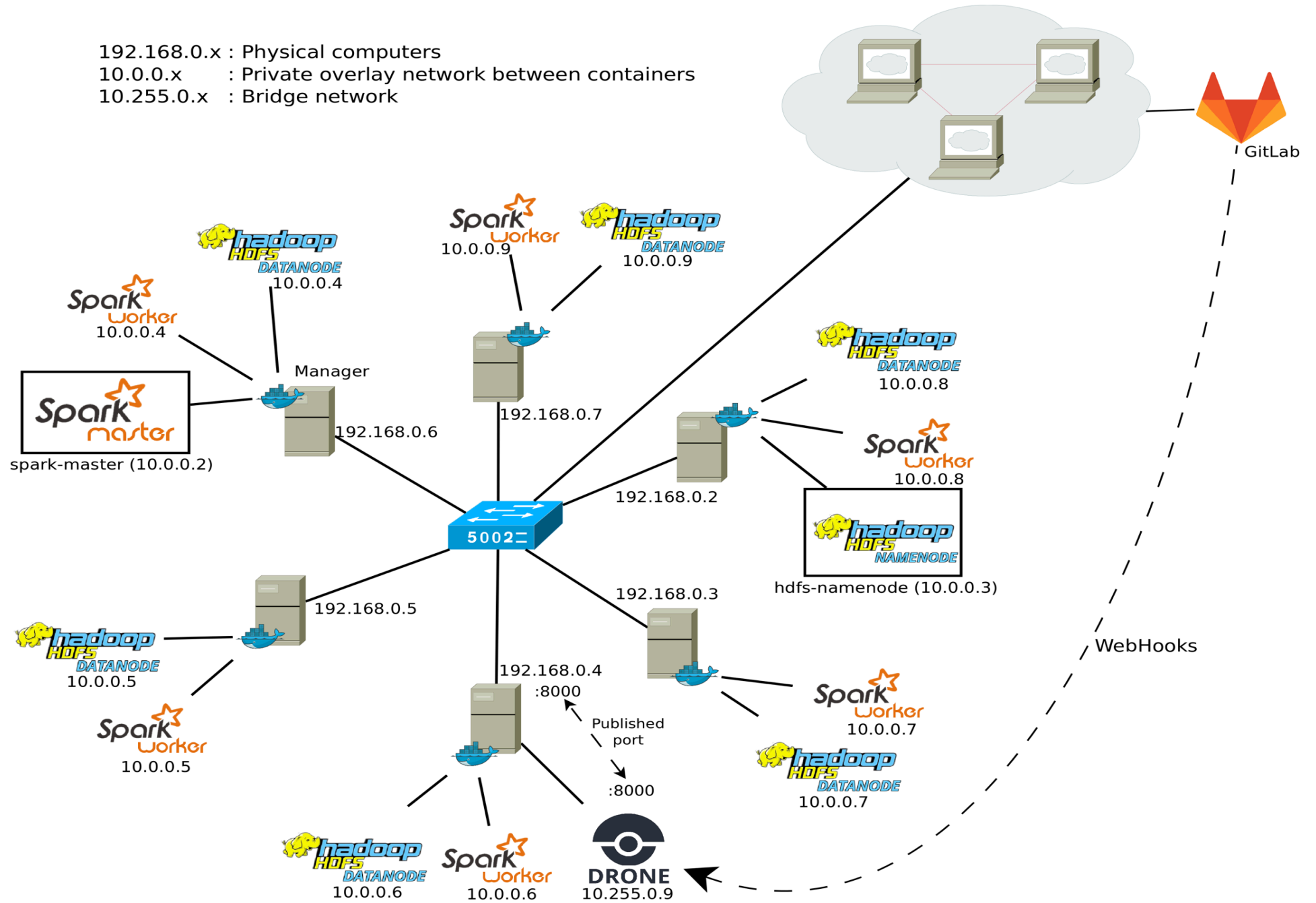
□ On-going work

- Spark provides a **simple** interface BUT you probably need to **understand internals** to **optimize** non-trivial problems.
- You are stuck into the Map/Reduce model (more general, but hard to optimize).
- **Not suited** for small configurations ? (2MASS/SDSS crashes on 6 desktop machines, needs deepest investigations).

□ On-going work (2)

- 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).

192.168.0.x : Physical computers
 10.0.0.x : Private overlay network between containers
 10.255.0.x : Bridge network



□ How it works ?

- Docker is installed on all computers and configured in **Swarm mode**
- **One node** is defined as the **manager** (where we can create services)
- Overlay network to let the containers (running on the different nodes) communicate with each other
- To create the different services we need to have an image to create each container (**Spark**, **Hadoop** and **Drone** on the previous slide)
- Finally we can create all the services on the **Docker manager**
- The spark master and the **HDFS** namenode services have a **replication of 1** (their container will be running on only one node of the **Swarm**)
- The **HDFS** datanode and Spark worker services run in a global mode (their container will be running on all nodes of the **Swarm**)
- The Spark workers and HDFS datanodes can **communicate** with their **master server** by using their names (spark-master and hdfs-namenode) which will be resolved by the **embedded DNS** server
- The **Drone** service, used for automatic builds, has a published port to get access to the Web GUI. This port is mapped on all the nodes and this service can be accessed from the outside with the address of any node despite that the container is only running on one node (e.g. 192.168.0.4:8000 and 192.168.0.5:8000 will lead to the same page).

□ 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 mirrors 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**”.

□ 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/>