Data lake project at APHP

Telecom ParisTech supervisor:
Pascal Bianchi (full Professor)

Author:
Catherine Verdier (MSBGD2016 class)

AP-HP supervisor:
Dr Christel Daniel (Director of the Web Innovation and Data departement)

Project Team:
- *Nicolas Paris (WIND R&D Manager)*
- *Louis Mayaud (Research Director at Mensia Technologies)*
- *Stéphane Bréant (WIND Project Manager)*
- *Romain Pirracchio (Anesthesiologist and critical care physician at Georges Pompidou Hospital)*
- *Emmanuel Rougerie (System Administrator),*

List of abbreviations used in this document

ABPD	Arterial Blood Pressure (Diastolic)
ABPS	Arterial Blood Pressure (Systolic)
ACID	Atomicity, Consistency, Isolation, Durability
ANS	Autonomic Nervous System
APHP	Assistance Publique Hôpitaux de Paris
API	Application Programming Interface
ATI	Agence Technique Informatique
AUC	Area Under the Curve
BCI	Brain Computer Interface
BP	Blood Pressure
CPU	Central Process Unit
CSV	Comma Separated Value
EDS	Entrepôt de Données de Santé
ETL	Extract Transport Load
FDW	Foreign Data Wrapper
FFT	Fast Fourier Transform
FSM	Finite State Machine
GFS	Google File System
GPU	Graphics Process Unit
HDFS	Hadoop Distributed File System
HF	High Frequency
HR	Heart Rate
HRV	Heart Rate Variability
ICU	Intensive Care Units
IDE	Integrated Development Environment
JDBC	Java DataBase Connectivity
JSON	Javascript Object Notation
JVM	Java Virtual Machine
LF	Low Frequency
MapR-FS	MapR File System
MDM	Minimum Distance to Mean
MLlib	Machine Learning Library (Distributed ML algorithms available with Spark)
NoSQL	Not Only SQL
ODBC	Open DataBase Connectivity
ORC	Optimized Row Columnar
PCA	Principal Component Analysis
POC	Proof of Concept
PMSI	Programme de Médicalisation des Systèmes d'Information
RCFile	Record Columnar File
RDBMS	Relational DataBase Management System
RDD	Resilient Distributed Dataset (Spark concept)
REPL	Read Eval Print Loop
ROC	Receiver Operating Curve

RR	Respiration Rate
SAPS	Simple Acute Physiology Score
SLA	Service Level Agrement
TSD	Time Series Daemon (OpenTSDB Core component)
ULF	Ultra Low Frequency
VLF	Very Low Frequency
WIND	Web INnovation Data
XML	Extensible Markup Language

Abstract

APHP (Assistance Publique des Hôpitaux de Paris) is the biggest health organisation in Europe.

It includes 39 hospital units and its activities are varied: providing healthcare to patients, educating health workers and initiating or participating in Health Research programs.

More than 8 million people are receiving cares from its departments every year and during their stays, a large amount of data is collected.

In 2015, the organization is involved in 3430 clinical research projects and has initiated 999 of them.

APHP has started several projects for which, one of the aims is to implement new solutions to be able to produce more value from the records provided by its care units.

Among these projects, we can mention the following:

    I.   Orbis whole deployment;
Orbis is an Information System solution which main function is handling hospital patient folders.
The legacy solution is a product developed and distributed by AGFA-HealthCare.
A customized version (cross-hospitals) has been implemented for APHP[1].

    II.   Creation of a big healthcare data-warehouse;
The APHP EDS ("Entrepôt de Données de Santé") is a multi-sources data-warehouse built to give materials to APHP Research teams [2].

    III.   Deployment of the i2b2 application in all the APHP's Research units;
i2b2 (Informatics for Integrating Biology and the Bedside) is a scalable[1] framework which allows clinical researchers to collect easily datasets from a RDBMS modelled to store clinical records [3].

    IV.   Creation of a data lake;

    V.   Usage of the data lake with i2b2;

    VI.   Promotion of machine-learning techniques to produce more value from data.

Projects I to III are currently operational for a part of the APHP data. They are continuously improved with new hospital coverage and new functionalities required by users.

Concerning projects IV, V and VI, APHP objective is to install and feed an Hadoop 5-machines cluster possibly based on the MapR distribution.

The aim of this professional thesis is related with these 3 last projects and expected works should be defined and oriented in order to answer to the following questions:

- what are the right components and software architecture to store and query very large datasets?
- what are the right components and software architecture to catch properly and store waves provided by connected health devices?
- Is there any solution allowing to interact with the data lake from i2b2?
- What could be a good development environment to create custom batches, customs real-time acquisition processes, data-visualisation solutions?
- Can we build a machine-learning project with open-source/freeware tools and health data able to demonstrate the relevancy of the data-science techniques to generate new value from health data?

---

[1] "scalable" at i2b2 level refers to its last query generator system which allows to browse several i2b2 databases to produce a result [31]. This system is not as sophisticated as an Hadoop cluster which tends to appear as a single server to the user.

Table of Contents

Introduction

Healthcare, clinical research, as many humanities disciplines, are activities in which data collection and analysis become strategic concerns.

As many health organisations, APHP wants to keep its leadership in innovation.

In that way, the Great Paris Hospital Group has identified the command of big data technologies and the usage of data-science techniques as a new objective to achieve.

The works presented in this thesis relates key knowledges on concepts, frameworks and toolbox used, feasible technical solutions for future projects and a machine-learning exploration case as a "proof of concept" (POC).

This document does not intend to be exhaustive but could be seen as a serie of proposals to cover new functional needs at APHP.

Some of the experiences applied should be able to interoperate with existing projects. Hence, after a short introduction of people involved in this project, we will begin by an exposition of the operational data projects yet implemented.

Because one of the identified need is the build of a data lake based on the Hadoop MapR distribution, in a second step, we will give overviews on the Hadoop concepts and tools and also expose what are the specificities of the MapR distribution.

Then, we will discuss solutions for developing custom jobs and data analysis within the Hadoop platform.

The fifth part of this document will be dedicated to test and compare some technical architectures for :

- importing large datasets in the MapR cluster;
- querying data from the cluster;
- dealing with streams on the cluster.

The last paragraphs of the document will present a machine-learning POC.

Project team

The purpose of this section is to present people who has participated in this project, their function and the role they have played to make this work possible.

- **Nicolas Paris**, R & D Manager at WIND
  Organization and follow-up the project, material provider, extraction and supply data for the purposes;
- **Louis Mayaud**, Research Director at Mensia Technologies
  Data-science POC subject proposal, support and follow-up on POC implementation and results;
- **Stéphane Bréant**, Project Manager at WIND
  Provide data and functional information, specially about SAPSII data and implementation in Orbis;
- **Romain Pirracchio**, Anesthesiologist and critical care physician at Georges Pompidou Hospital (HEGP)
  Provide information about Intensive Care Units organization and working methods;
- **Emmanuel Rougerie**, System administrator at ATI
  Installation and administration of the MapR test platform in the APHP DataCenter;
- **Catherine Verdier**, Student at Télécom ParisTech (mastère spécialisé Big Data)
  Evaluation of the MapR platform, data-analysis, data-science, design, development

Existing data projects at APHP

For several years, APHP has initiated projects to get a more streamlined information system and be able to produce more value from available data.

In that regard, some ambitious roadmaps have begun:

1. Deploying the Orbis Information System in all hospitals by 2020 [1]
   A standard information system will be a simpler way to exploit data
2. Applying standard international repositories to describe clinical acts, biology acts, treatments … and so on … when it is possible
3. Building a datawarehouse (EDS[2]) with several health data sources to be able to supply relevant datasets to clinical researchers [2]
4. Providing the i2b2 web application to medical research teams allowing them the ability to build their dataset by themselves with the EDS data [3]

Projects 2, 3 and 4 have been entirely defined, managed and implemented by the recent WIND (Web Innovation Data) department currently directed by Christel Daniel. This entity wants to improve its skills in new data analytics solutions to enlarge the current projects coverage and define new guidelines and best practices for IT teams at APHP.

More precisely, APHP holds a large amount of unstructured data (images, wavelets records, clinical reports) which are not easily available for researchers. Some measurements performed at bedside could not be stored permanently within classical RDBMS[3].

For these reasons, the WIND service next initiative planned is the build of a data lake based on a Hadoop distribution.

Why Hadoop?
A brief history of the Hadoop success story

Hadoop version 1

Hadoop is the first open-source implementation of the MapReduce distributed system designed by Google to deal with large amounts of data (Google aim was to build the most complete Web pages inverted index) on commodity hardware [6].

The initial framework requirements (Hadoop version 1) were the following:

- Ensuring data persistence and availability (GFS/HDFS[4]);
  Moreover, GFS/HDFS allows a minimization of jobs failures.
- Optimize the network bandwidth in data processing with preferably transfer executable code rather than data (locality requirement);
- Building a scalable solution;
- Designing an easy way to develop distributed jobs (MapReduce programming model).

---

[2] EDS = Entrepôt de Données de Santé
[3] RDBMS = Relational DataBase Management System
[4] GFS = Google File System – HDFS = Hadoop Distributed File System

The ability of running on commodity hardware, the fault tolerance and the simplicity of the MapReduce paradigm were probably the reasons of the Hadoop success story: with a moderate investment, companies have built "data lakes" and imagine a lot of new applications which were not necessarily compliant with the state-of-the-art of the MapReduce programming model. Indeed: MapReduce (in its primary implementation) appears not relevant for iterative algorithms.

The weaknesses of the initial toolbox were:

- The HDFS namenode single point of failure;
  The HDFS namenode role is to manage and store the directory tree of the distributed file system.
  The replication of the HDFS namenode had to be managed manually.
- The strong coupling between HDFS and the MapReduce programming model (HDFS could be also useful for non MapReduce applications);
- A degraded scalability due to the centralized management of jobs (bottleneck).

Hadoop version 2

To overcome the drawbacks of the Hadoop v1 platform, several actors (already involved in the Hadoop design) have defined the architecture of the new generation of Hadoop frameworks.

The guiding idea is to introduce a key component to interact with the HDFS layer. This software block has been named YARN (Yet Another Resource Negociator) [4]. Yarn is the resource manager (RM) of the platform and each Hadoop application should be "on yard" implemented to deal with HDFS data [7]. In that v2 platform, MapReduce has been re-implemented as a yarn application.
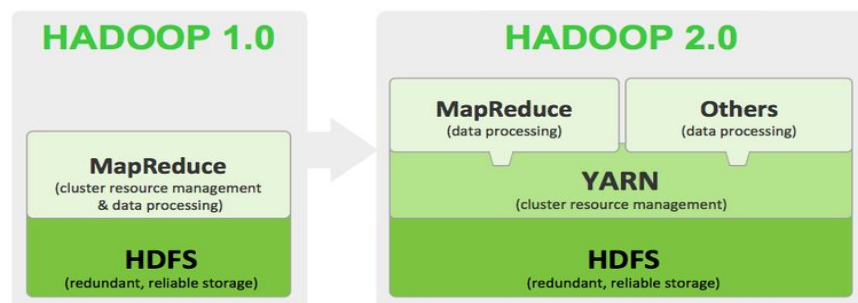
In Hadoop v2, the Yarn layer can be configured to replicate HDFS nodename data ensuring High Availability [9].

Implementing a MapReduce job as a Yarn application reduces the old job handling bottleneck and allows a higher scalability of the system: 4000 nodes maximum for Hadoop v1 versus 7000 nodes for Hadoop v2 regarding Yahoo! recommandations (see [4] § 4.1.1).

As a new requirement, we can also note the multi-tenant capability of the Hadoop v2 solution.

Since the availability of Yarn and the Hadoop v2 platform, a lot of distributed software have been developed or moved to deal with HDFS (HBase, Spark, Hive, Pig, Storm and so on …), and also many other applications are candidates to become Yarn compliant (e.g. [5] for OpenMPI). Hence, the Hadoop v2 is now a large galaxy[5] of components in which it is not so easy to navigate.

---

[5] 'Galaxy' seems to be a relevant term to define the Hadoop ecosystem because we can see there new products being born and dying very quickly. So, to build a solution, companies are often selecting proven architectures (supported by a large community) rather than new components (except for GAFA and other Web companies which are involved in the development of the new Yarn applications).

MapR toolbox quick overview

With the emergence of Hadoop v2, it is often necessary to install a set of Yarn applications to get a complete toolbox for big data analysis. In that way, it is possible to get a cluster supplied by a cloud computing company (Amazon Web Services, Microsoft Azure, Google cloud...). If data should be internally hosted (for security or legal purposes), it is easier to use one of the solutions proposed by the distributors.

We will not give a detailed comparison between the different Hadoop vendor solutions in this document. People interested in that topic could refer to [8], [10] and [11].

The MapR distribution includes some proprietary components which implement the mandatory Apache APIs:
- MapR-FS substituted to HDFS;
- Mapr-DB substituted to HBase;
- Mapr-Streams (scalable queue messages implementing the Apache Kafka API).

MapR-FS

The Hadoop Distributed File System, in its design presents several drawbacks ([14]):
- Bottleneck and Single Point Of Failure with the NameNode cluster (to understand the HDFS NodeName concept, you can have a look at [12]);
- An inability to update data due to its interaction with the underlying Operating System via the Java File System API (append-only file system).

The MapR-FS mitigate in that limitations ([13]) with:
- The suppression of the NodeName cluster: distributed file system information are stored in a distributed database[6] (specific file system containers);
- A direct interaction with NFS allowing random writes in the chunks[7] (a disk space dedicated to each cluster-node has to be supplied to MapR-FS at installation)[8].

MapR-FS ensures higher performances and scalability capabilities than HDFS.

It is available in the MapR Community Edition (freemium offer) without the High Availability functionality (file system information containers replicated in the cluster).

MapR-DB ([16])

MapR-DB is the database proprietary solution recommended by MapR. It is available in the Community Edition since 2014.

It implements the HBase API (version 1.1) and allows to store tables in HBase format (HFile v2) or MapR-FS format.

In further works and benchmarks operated at APHP, we will be able to compare the efficiency of these two different storage modes.

It is also possible to use MapR-DB as an Oriented Document Database (using one of the dedicated API : OJAI for Java or maprdb package for Python3).

MapR-DB combines the capabilities of HBase (columnar database) and MongoDB (document oriented database) in a unique component.

---

[6]The CFS (Cassandra File System) solution recently proposed by DataStax and quickly exposed in [14] is quite close to MapR-FS.
[7]A chunk is a HDFS block of large file
[8]The CEPHFS open-source system can offer analog properties but it is currently compliant with the Hadoop v1 solution only ([15])

MapR-Streams

In many streaming applications, a component ensuring the messages persistence should preferably be included, in order to avoid losing data. The most popular scalable solution used in high frequency streaming is Apache Kafka (initiated by LinkedIn see [17]).

MapR-Streams is the MapR scalable message queue: it implements the Apache Kafka 0.9 API and is able to store messages dealing directly with MapR-FS ([18]).

Other tools in the distribution

In addition to the proprietary components listed above, MapR Community Edition includes a serie of open-source Yarn applications such as Hive, Pig, Oozie, Spark, Drill and also several Web interfaces as Hue (Hadoop User Environment).

Assessment

MapR Hadoop custom solution is axed on low layers rationalisation and optimisation. The promise is more getting a performant and reliable platform than having dedicated interfaces to manage and monitor the cluster or performing data-visualizations and analysis. A part of the work carried out at APHP will attempt to check if a MapR cluster can hold the high volume of data not entirely stored yet in the current Information System.

Developing for the Hadoop platform

As described above, an Hadoop cluster is a large toolbox including many components.

For the APHP software teams, which has only a few experience with the platform, it could be an interesting material to dispose of a classification of the different components available: this is the purpose of the Appendix A.

Making our experiments, it has appeared that some components, such as MapReduce and Spark[9], require more accurate programming skills and that describing the ways and recommendations to implement jobs with them will be useful for the future works. So, we will first have a discussion about the development environments which can be used for each of these Yarn applications. People interested in the detailed steps to build some simple projects with the Eclipse IDE have to read the Appendix B.

Last, we will have a short discussion on pros and cons of the different notebooks which can be used on the platform. The Appendix C will describe our experience with Zeppelin.

Developing MapReduce jobs

Hadoop v1 and MapReduce Yarn application have been developed in Java and C.

Most of old Hadoop v1 application have been written with Java (few with C/C++).

They are still compliant with Hadoop v2 once recompiled with the new libraries.

It seems reasonable to continue using Java for developing custom MapReduce jobs.

---

[9]Yarn itself could have been included in this list. The Apache Hadoop documentation provides a guideline to create Yarn applications ([7]). We do not present this topic because developing Yarn applications is not so common and because we have not yet developed any Yarn project.

However, because Java is quite complicated and verbose, because many new scripting languages have recently emerged (python, R, Ruby,…), since release 1.2.1, the Hadoop framework supplies the "Hadoop streaming API" ([20]) which is the best way to create jobs from scripting languages (see benchmarks performed in [19]).

As shown in [19], Java stays more efficient to implement jobs, but what about Scala?

Scala has a special place among languages: it can be very concise and is compiled in bytecode as Java. So, it could be an interesting way to use it for future works. In that regard, Twitter has implemented the Scalding project designed to develop efficient MapReduce jobs with few lines of code ([21]).

The Appendix B will present a WordCount MapReduce job implemented with scalding and refers to a github repository holding a maven project with several tests we have experienced with this framework.

Developing Spark jobs

On the Spark side too, several languages can be used: Java, Scala, Python and R.

Spark itself has been implemented in Scala. Java and Scala objects can be directly used by SparkContext. However, for Python and R, objects have to be serialized (with py4j and R-JVM-bridge respectively), sent to the SparkContext and de-serialized. Hence, with the default configurations, Java and Scala will be more efficient to run Spark jobs.

Moreover, because Scala is easy to use with the Spark API, it will be faster for development than Java. The Spark package supply the Spark-shell tool (a REPL[10] engine for scala) which is useful to prototype jobs and as no equivalent for Java[11] ([22]).

We can note that Python code is not yet irrelevant with Spark: performances can be highly improved using the pypy project (see [24] and [25]) instead of the standard CPython interpreter ([23]).

A few words of open-source notebook servers solutions

Several notebooks solutions are available to run spark jobs and visualise results.

Some of them are dedicated to Scala (scala-notebook, spark-notebook).

Using Spark with R-Markdown from R-Studio (https://github.com/rstudio/sparklyr) seems to be possible.

Both Jupyter and Zeppelin offer the ability to use multiple kernels or interpreters (python, R, scala, …)

We decided to install and test Zeppelin, which allows scripting with several interpreters in the same notebook.

Making tests with it, we have noted that Python and R offer higher capabilities for visualization (once spark RDD[12]s or Dataframes has been converted to pandas Dataframes or R Dataframes).

Hence, for our Machine Learning POC, we have prefered jupyter notebook, because:

- it is more reliable;
- its results can stay embedded in the .ipynb file and can be easily shared with a git backend server such as GitHub.

The way to install and configure Zeppelin is exposed in the Appendix C.

---

[10] REPL = Read-Eval-Print-Loop

[11] REPLs are also available for Python (pyspark) and R (sparkR)

[12] RDD = Resilient Distributed Dataset (Spark object to manage distributed datasets)

Dealing with big dataset on MapR

In that section, we will discuss about the different formats which can be used to import, store and query a huge dataset on the MapR platform.

Each format tested will be studied in terms of :
- mutability[13];
- way and efficiency to perform import;
- way and performance to query data;
- ability to interoperate with the APHP applications.

Material

For each case, tests have been carried out with:
- A dedicated single machine cluster (6 cores, RAM 32Gb);
- A huge key/value CSV file generated (100,000,000 rows; key and value as strings; about 1.3Gb);
- The following queries (expressed in sql language) ⇒
  - **select count(1) from <Table> where key like 'AE%'**
  - **select * from <Table> where key like 'AE%' limit 1**
  - **select * from <Table> where key like '%AE%' limit 1** (full table/index browsing query);
- The tests have been applied through tools compliant with the used formats (Drill, Hive, HBase shell, Spark or MapReduce), depending on the context (import / query).

Storing big data

Since the first open-source Hadoop platform has emerged, many file formats have been developed in order to improve the MapReduce efficiency. The Hadoop framework is able to deal with a lot of data storing modes: text files, flat files, relational and Nosql databases (CSV, JSON, Avro, RCFile, ORC, Parquet[14], HBase, MongoDB...). Some of them, such as Avro, have been created for serialization needs and offer few mutability power.

In this section, we will present our experiments with:
- ORC;
- Parquet;
- Mapr-DB with HFilev2 format (HBase classical);
- Mapr-DB columns database with Mapr-FS format (Mapr proprietary format);
- Mapr-DB JSON documents database with Mapr-FS format.

ORC and Parquet

Mapr-DB has been described above. Now, we want to present ORC and Parquet files with which we have performed tests.

ORC and Parquet are the most successful file formats currently available in the Hadoop ecosystem. They are respectively highly related with Hive for the first one and Drill/Impala[15] for the second one and give high performances for big data analysis. They are both columnar formats, both able to encode and decode nested objects (as XML or JSON), they both use built-in rows, columns indices and data compression (to optimize I/O accesses in query processing).

---

[13] mutability is the ability of performing targeted updates on data

[14] CSV = Comma Separated Value / JSON = Javascript Object Notation / Avro = Avro / RCFile = Record Columnar File / ORC = Optimized Row Columnar / Parquet = Parquet

[15] Drill (MapR) and Impala (Cloudera) are two different open-source implementations of Google Big Query [27]

Th reference [27], describing the Google Big Query system is presented as the paper of reference describing the Parquet format: Parquet is inspired from the Google Big Query Dremel format. Dremel is a "in memory" format whereas Parquet is moreover persistent.

Dremel: from nested format to columnar format

Storing nested data in a columnar format is a very challenging purpose. We have decided to explain the model used by Parquet: Dremel, because we have used it intensively in APHP data exploration.

The encoding of records is highly dependant of their structures. Hence, when the encoding is performed, no change is possible on the data model level without a complete rebuild of the Parquet dataset.

In a first approach, we present the example used in the Google Dremel paper [27], with additional comments:

Nested Object definition	Record1	Record2

message **Document** {
required **int64 DocId**;
optional group **Links** {
 repeated **int64 Backward**;
 repeated **int64 Forward**;
 }
repeated group **Name** {
 repeated group **Language** {
  required **string Code**;
  optional **string Country**;
 }
 optional **string Url**;
 }
}
	DocId: 10
	Links
	 Forward: 20
	 Forward: 40
	 Forward: 60
	 (Backward: NULL)[16]
	Name
	 Language
	  Code: 'en-us'
	  Country: 'us'
	 Language
	  Code: 'en'
	  (Country: NULL)
	 Url: 'http://A'
	Name
	 (Language)
	  (Code:NULL)
	  (Country: NULL)
	 Url: 'http://B'
	Name
	 Language
	  Code: 'en-gb'
	  Country: 'gb'
	 (Url: NULL)
		DocId: 20
		Links
		 Backward: 10
		 Backward: 30
		 Forward: 80
		Name
		 (Language)
		  (Code: NULL)
		  (Country: NULL)
		 Url: 'http://C'

Dremel columns

r = In the path to the field, what is the last repeated field ? (**repetition Level**)
d = In the path to the field, how many defined fields ? (**definition level**)

DocId			Name.Url			Links.Forward			Links.Backward			Name.Language.Code			Name.Language.Country		
value	r	d	value	r	d	value	r	d	value	r	d	value	r	d	value	r	d
10	0	0	'http://A'	0	2[17]	20	0	2	NULL	0	1	'en-us'	0	2[18]	'us'	0	3
20	0	0	'http://B'	1[19]	2	40	1	2	10	0	2	'en'	2	2	NULL	2	2
			NULL	1	1[20]	60	1	2	30	1	2	NULL	1	1	NULL	1	1
			'http://C'	0	2	80	0	2				'en-gb'	0	2	'gb'	1	3
												NULL	0	1	NULL	0	1

---

[16] Undefined values have been added in red to help in the understanding
[17] Name is defined and Url is defined
[18] Only 2 because Code is a required field
[19] Last repeated field is Name
[20] Name is defined and Url is undefined

To optimize the encoding, NULL values are not explicitly stored: they can be identified when the value descriptor consists on only two fields (r and d), and the values are compressed.

To build the complete columnar table from a set of records, the algorithm uses a FSM[21] derived from the record structure. The FSM should be traversed recursively for each record ([27] p 9, fig 16) to compute the levels (repetition and definition i.e. r and d) of each field. An algorithm with two steps to construct an appropriate FSM is given in [27] p 10: first, scan the entire input dataset to compute the maximum depth of the model (fig 19), then shape the FSM using the maximum depth and the field list (fig 18).

Note that:
1. the FSM is simple for an input CSV file (not nested);
2. the column encoding could be more efficient with the knowledge of field data type (when the data-type is not given, all columns are encoded as strings)

Hive/ORC query Vectorization: relevance of columnar storages for big data analysis

As explained in [26] §6, columnar storages are more able to take advantages of CPU pipeline capabilities, provided that, in the inner loop processing a huge dataset, the used instructions stay basic. This can be achieved with Hive/ORC and the vectorization configuration. Using vectorization in ORC, implies to give up the lazy deserialization default mode (lazy deserialization allow less memory consumption but increases the inner-loop complexity). The same goes with a row format storage because the row-by-row processing generates often too much code ramifications.

With columnar models, a projection operation complexity is O(1): only keep columns of interest whereas performing a restriction on non key field (without index) requires a complete column lookup O(N) for each restriction.

Vectorization operates with block of data columns loaded as arrays in memory (batch processing). Hence, multiple restrictions can be parallelized (multi-threading) and moreover, when the hardware architecture includes vector processors (such as GPUs), the gpu vector instructions can be exploited.

Mutability power

The following table is an overview of the mutability power of each format used for our tests:

ORC	Parquet	MAPR-DB/HFilev2	MAPR-DB/MAPR-FS	MAPR-DB/JSON
- adding rows only - immutable data model - targeted updates with the 'ACID[22]' option	- adding rows only - immutable data model	- adding rows - adding columns in existing column family - index performances degradation with added rows	- adding rows - adding columns in existing column family - **no index performances degradation** with added rows	- adding rows - model entirely mutable

---

[21] FSM = Finite State Machine
[22] ACID = Atomicity, Consistency, Isolation, Durability

Importing a big dataset (see [Appendix D](#) for detailed user commands)

To import our large dataset (as presented in [§ "Material"](#)) on the cluster.

This is an extract of the CSV file:

```
ii1o,8yex
8m1m,fZ6r
0oR8,8135
2A01,0Us8
```

Several methods and storages have been experienced. The following table is a synthesis of our results:

method/storage	ORC	Parquet	MAPR-DB/HFilev2	MAPR-DB/MAPR-FS	MAPR-DB/JSON
MapReduce job	NA	NA	**time: ~53m** **IC=4**	**time: ~ 25m** **IC=4**	NA
Hive insert/select	**time: ~5m** **IC=2**	**time: ~5m** **IC=2**	NT	**time: ~ 30 m** **IC=2**	NA
Drill CTAS	NA	**time: ~1m** **IC=1**	NA	NA	NA
Spark job	NT	NT	NT	**time: > 4h** **IC=2**	NA
Java OJAI	NA	NA	NA	NA	**time: ~ 45 m** **IC = 3**

Legend:
NA = Not Applicable
NT = Not tested
CTAS = Create Table AS (the unique statement available to insert data with Drill)
IC = Implementation complexity (scale from 1 to 5)

Additional notes:
1. The MapReduce test has been performed with the importTsv job available with the HBase library; It can import directly rows or generate a bulk-load file. We did not success to import our huge bulk-load with the LoadIncrementFile HBase class whereas this operation works properly with smaller csv files. So our tests are based on a direct row import.
   The ImportTsv class code is available at: https://github.com/...
   A version able to deal with HBase composite keys exists at: https://github.com/boorad/CompositeKeyImportTsv
2. The spark job has been started from Zeppelin with 2Gb workers. The result appears not efficient.
3. Before processing Insert/Select from Hive, the text file should be mapped in the Hive metastore (this can be applied with Hue). The mapping has high restrictions formats.
4. Our Java OJAI code is available at:
5. Drill and Hive parquet files are not compliant on MapR5.1 (Parquet encoding versions seems different)
6. To fill a Maprdb table, it should be previously created. To fill it from hive, it should be mapped on Hive
7. Detailed used commands could be seen in the [Appendix D](#).

Querying a big dataset

In that section, we will present our benchmarks with the 3 queries listed in [§ "Material"](#).
Test have been performed with several tools supplied in the Mapr5.1 distribution when applicable.

Count query results

For this case, the query tested is:

select count(1) from <Table> where key like 'AE%'

tool/storage	ORC	Parquet	MAPR-DB/HFilev2	MAPR-DB/MAPR-FS	MAPR-DB/JSON
Hive	time: ~1m46s IC=2	time: ~2m27s IC=2	time: 18s IC=2	time: 18s IC=2	NA
Drill	NA	time: ~9s IC=1	time: ~1.5s IC=1	time: ~1s IC=1	NA
HBase shell	NA	NA	NA	NA	NA
Java OJAI	NA	NA	NA	NA	time: < 2s IC=3

Legend:
NA = Not Applicable
IC = Implementation complexity (scale from 1 to 5)

Additional notes:
All count queries are performed via MapReduce jobs in Hive

Selecting rows with key restriction

For this case, the query tested is:

select * from <Table> where key like 'AE%' limit 1

tool/storage	ORC	Parquet	MAPR-DB/HFilev2	MAPR-DB/MAPR-FS	MAPR-DB/JSON
Hive	time: ~0.1s IC=1	time:~1m3s IC=1	time: ~0.2s IC=2	time: ~0.2s IC=2	NA
Drill	NA	time: ~0.5s IC=1	time: ~0.8s IC=1	time: ~0.36s IC=2	NA
HBase shell	NA	NA	time: ~13s IC=1	time: ~0.2s IC=1	NA
Java OJAI	NA	NA	NA	NA	time: < 1s IC=3

Legend:
NA = Not Applicable
IC = Implementation complexity (scale from 1 to 5)

Additional notes:
To process restrictions with the hbase shell, we used PrefixFilter (see: http://www.hadooptpoint.com/filters-in-hbase-shell/)
Our query becomes: scan '<Table>', {FILTER => "(PrefixFilter ('AE'))", LIMIT => 1} (count with filter is possible with MapReduce jobs only)

Selecting rows with large key restriction

For this case, the query tested is:

select * from <Table> where key like '%AE%' limit 1

In that case, the index could not be used ⇒ the table or index will be entirely browsed

tool/storage	ORC	Parquet	MAPR-DB/HFilev2	MAPR-DB/MAPR-FS	MAPR-DB/JSON
Hive	time: ~0.1s IC=1	time: ~1m23s IC=1	time: ~0.2s IC=1	time: ~0.2s IC=1	NA
Drill	NA	time: ~0.3s IC=1	time: ~16s IC=1	time: ~4.3s IC=1	NA
HBase shell	NA	NA	NA?	NA?	NA
Java OJAI	NA	NA	NA	NA	time: > 5m IC=3

Legend:
NA = Not Applicable
IC = Implementation complexity (scale from 1 to 5)

Discussions

About formats

Importing big data files using tools as Hive or Drill could be an interesting way in terms of time performances, however, these tools are able to deal with a few input formats and we have to keep in mind that most of data could have very various structure even poor structure. So, be able to build a MapReduce program stays essential to deal with any kind of data. In that case, implementing jobs with Twitter scalding is probably the best solution: this tool manages most of Hadoop formats (except ORC) as well in input as in output and jobs can be implemented with it in a way very similar as Spark.

For each aimed purpose above, we have checked the high relevance and usability of popular file formats such as parquet or ORC. Moreover, they stay performant with aggregation queries, and any column restriction (key agnostic). They are useful to produce fast analysis. But today, NoSql columnar databases are still the best solution to keep a few of flexibility with data structure models.

About query tools and APH-HP i2b2 interoperability

Hive is a tool commonly available on Hadoop platforms. It can be remotely accessed from a Java client API.
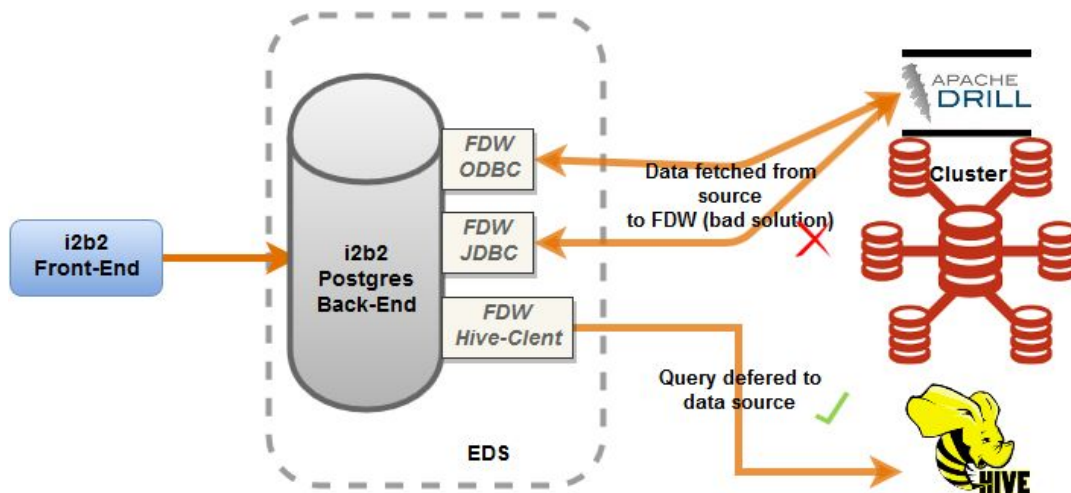
Drill is the MapR implementation of Google Big Query (distributed in-memory querying system) and offers additional functionalities as multiple source querying (flat files, RDBMS, NoSQL-columns, NoSQL-documents, Parquet, Avro) and also join capabilities with tables coming from heterogeneous sources[23]. Drill provides both JDBC and ODBC drivers.

---

[23] This functionality should be used carefully especially with large tables coming from external sources: in that case, the table is imported in the Drillbit (Drill cluster) and could cause an Out-Of-Memory exception.

APHP is expected for a solution allowing to query an Hadoop cluster from a Postgres Database (back-end of the i2b2 application) in order to be able to build i2b2 samples using data stored in a data lake. This can be carried out using a Postgres FDW (Foreign Data Wrapper - see: https://wiki.postgresql.org/wiki/Foreign_data_wrappers).

After tests, we have noted the inappropriate behaviour of the JDBC and ODBC FWDs for big data (all external data are downloaded to the Postgres memory which is not sustainable with big datasets). The Hive FDW seems useful because it delegates external sources queries to the cluster.

The following diagram illustrates one of the requirements which APH-HP would achieve with the data lake:

Big Data streams with MapR

Another important issue at APHP is to understand the state of the art regarding streaming architectures.

The aim is to be able to store large volume of data which could be provided by the measurement devices used by clinical teams every day.

This information cannot be exhaustively stored in the current information system (based on a RDBMS) because this is a too big amount of data.

An efficient streaming system should be designed regarding the following requirements:
- data loss minimization (fault tolerance);
- system availability

The message queue: a key component in streaming architectures

Streaming systems models involve two kind of actors: producers and consumers. To get the best compromise between fault tolerance and availability, modern systems are based on message streams and asynchronous communications ([29] chapter 2).
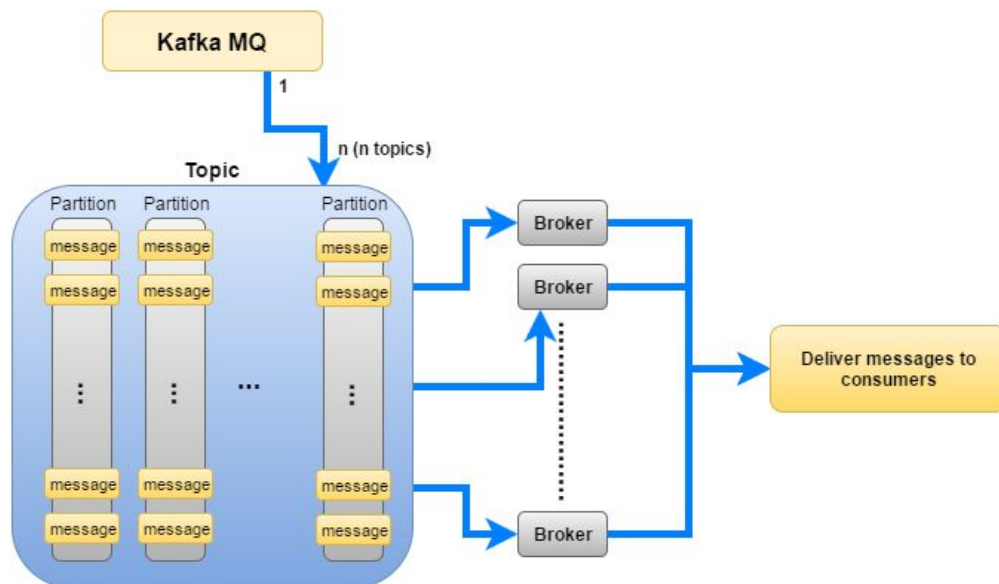
To allow asynchronous message exchanges, a message queue component is required to avoid data loss when a consumer fails or is unable to process stream fast enough. Moreover, a system can be resilient to a CPU crash if the message queue ensures message persistence.

Before the emergency of Kafka [17], traditional message queue components were poorly efficient when persistence was activated. Also, these ancient software were not always designed for scalability.

Kafka offers both message persistence (in an efficient way: persistence is systematic) and scalability. Message consumption is simplified: no information, no sequence number are stored in the messaging system (Kafka

brokers[24] are stateless). Hence, it is now the ideal component to allow asynchronous communication between producers and consumers. Its concepts are simple and it is easy to use with or without an Hadoop framework. With Kafka, message persistence is managed with a time-based SLA[25], when message delivery can be managed with three policies:

- Messages are never re-delivered but may be lost;
- Messages may be redelivered but never lost;
- Messages are delivered once and only once.

MapR streams vs. Kafka

Kafka is now intensively used in streaming architectures. Its model simplicity (one file per topic partition) holds several drawbacks [29] chapter 4:

- the performances decrease dramatically with a high number of topics (more than 100) due to complex IO operations (this can be a severe limitation in a context of multi-tenancy);
- Kafka does not supply an automatic mechanism for scalability (the entrance of a new machine in the cluster should be managed manually)

These limitations are related with the inner design of the software and could take several years before being resolved.

MapR streams is the MapR message queue solution to overcome the Kafka problems. It implements the Kafka producer and consumer APIs (to facilitate migration) and store topics and messages in the MapR-DB database. With this implementation, the number of topics is not limited and the scalability is entirely managed on the MapR cluster.

MapR streams solution is assumed to be relevant for multi-tenant platforms.

Thanks to the implementation of the Kafka APIs, it is easy to move from the open-source solution to MapR platform and conversely.

Micro-batch vs. Real Time Streaming Data Processing

Several years ago, when the first data lake platforms were built, only two modes were available to process data: batch and on-line modes (one event at a time).

---

[24] A broker is a Kafka component which delivers messages to consumers.
[25] SLA = Service Level Agrement

Batch mode processing was traditionally associated with MapReduce jobs, while on-line mode processing was deferred to external tools such as Storm or Flume for the most famous of them.

Many platforms in production, dealing with streaming data, are still implemented with Storm. This tool is known for its reliability and a large community of developers use it.
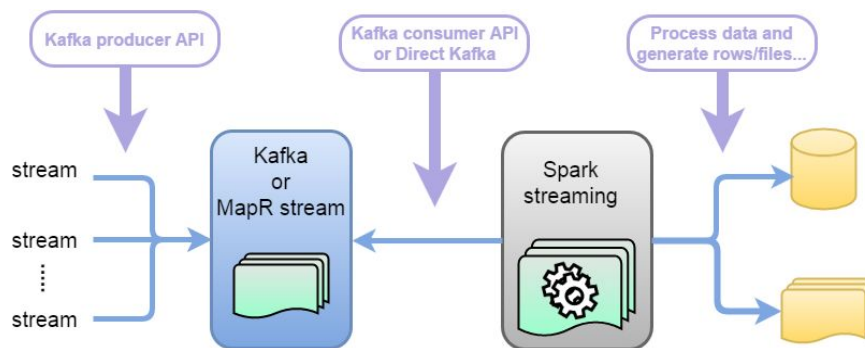
As Flume, Storm processes data one-at-a-time, while new emergent solutions as Spark-Streaming provide a model where data are processed by micro-batch: at a given frequency (every 1s, 1ms …). This frequency could be calibrated regarding to the requirements of the application. In case of failure, identically to the Kafka delivery modes, the consumption policy can be customized (loss of messages allowed/no loss message, doubles allowed/each message consume once and only once).

This model is not exactly a 'real-time' model, but it is generally more efficient when the volume of the stream is high.

The Storm community have developed Trident: a Storm plugin extension adding the micro-batch capability.

A couple of years ago, because Spark Streaming was not yet enough stable, MapR enhanced the OpenTSDB[26] (see: [30]) streaming solution with a micro-batch mode [28].

Spark streaming based architecture

This architecture can be directly implemented with the MapR Community Edition.

The Apache Kafka queue is provided (but not activated by the installer): the Message queue system recommended by MapR is MapR streams.

Spark streaming basic concepts are:

- the StreamingContext singleton object;
- the DStreams[27]: collections of RDDs available during a micro-batch processing.

Generally, to consume data provided by an external tool with Spark Streaming, we should use an API from a Receiver object. To deal with Kafka, Spark streaming offers a Direct Approach without Receiver: messages are consumed directly from the file system (using Kafka low level client API) and not throughout the Kafka brokers (Kafka high level API). It allows more efficiency and reliability.

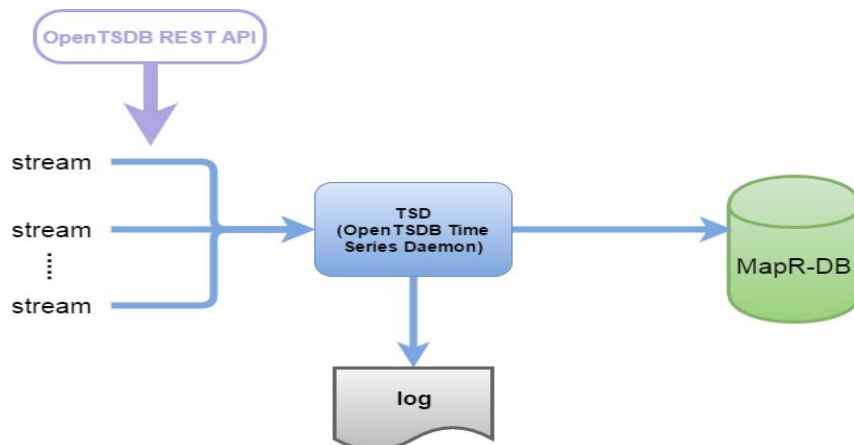Note that MapR streams can only be consumed using the direct approach.

Kafka broker component(s) can also be used to receive streams from remote producer (potentially remote devices). This is not possible with MaprStreams, so with the MapR message queue, the producer(s) should: or embed a server, or be able to pull the external devices regularly to get available stream.

---

[26] OpenTSDB is another scalable message queue solution: it is designed to store directly the stream in a database. It offers also a query language which allows to process streams. It is compliant with mysql, mongoDB, HBase, Cassandra database APIs.
[27] DStream = Discretized stream; DStreams methods are analog to RDDs methods and operate on each sub-RDD.

On the processing level, Spark streaming is able to generate a large variety of outputs: databases records, hdfs files, parquet, ORC files... It also allows on-line predictions capabilities with the ability of the MLlib[28] usage. It is also a relevant solution to implement a Kafka/MaprStreams producer pulling external streams.
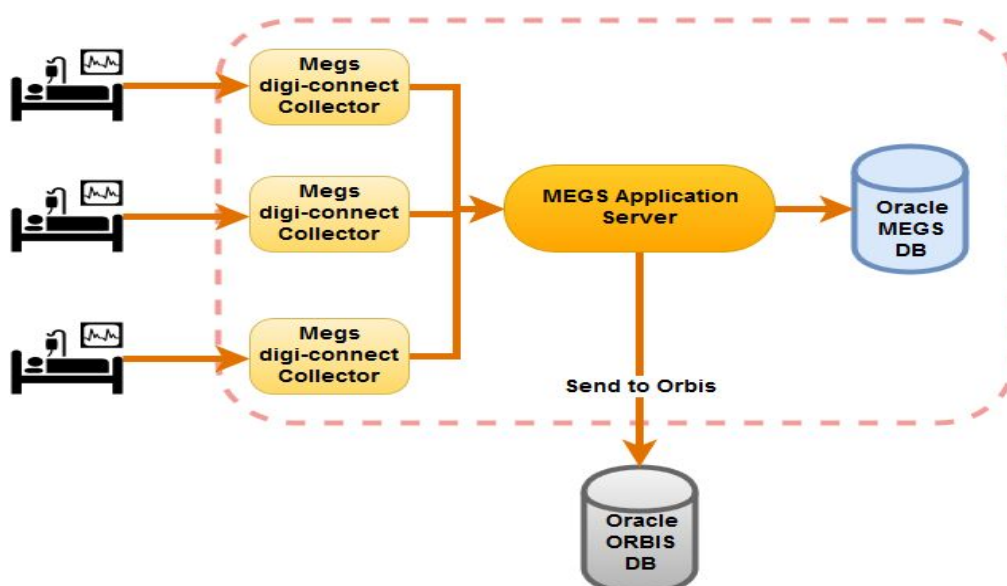
MapR OpenTSDB based architecture

The core component in OpenTSDB is the Time Series Daemon (TSD). External devices can push their streams to the TSD using its REST API. Streams are directly stored to the backend database in tables defined in the TDS configuration file.

The MapR custom TDS collects streams, compacts them as a bloomfilter (binary HBase compressed format) and store them in the database. In case of database failure, the TSD is able to rebuild bloomfilters from its logs.

The APHP streaming project

At APHP, Intensive Care Units collect data from devices related to beds. This information is then stored in a MEGS[29] application. The MEGS data are purged generally every 15 minutes (depending on the ICU's settings) which means that for each ICU a complete history of measures is available during this period. APHP would be able to record these data on the future data lake and provide it to researchers involved in programs exploring vital signs.

The technical architecture of MEGS is presented below:

---

[28] MLlib = Machine Learning Library (Spark library)

[29] MEGS is an Agfa HealthCare application (Medizingeräteschnittstelle = Medicine device interface)

Thus, to record as a stream in the Hadoop cluster data provided by ICUs, following solutions can be considered:
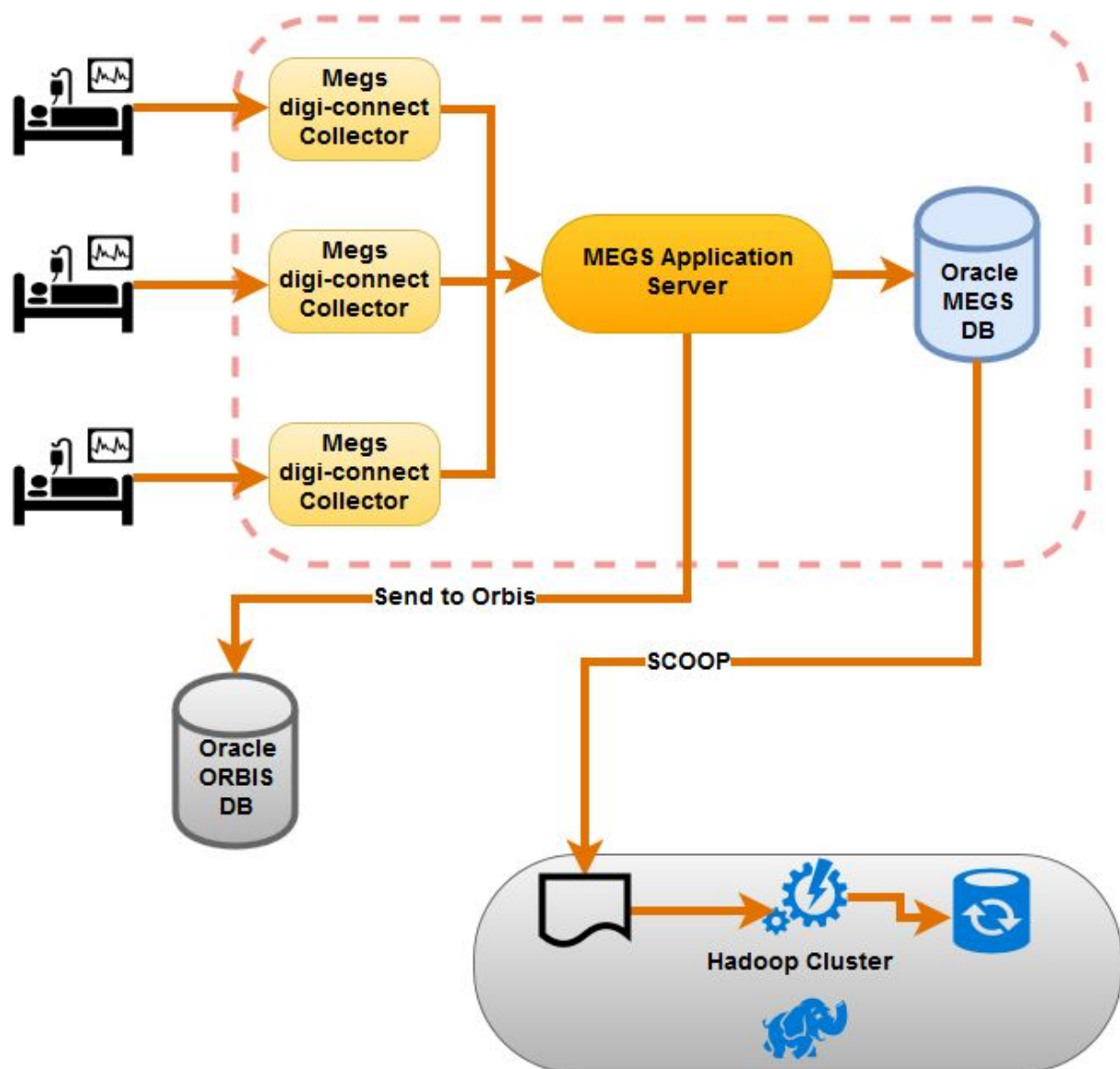
Collect data using apache scoop

Scoop is an Apache foundation open-source project which can be seen as a basic ETL[30] in the Hadoop ecosystem.
Any SGBDR source can be used as long as it can be accessed with a JDBC driver.
The target can be an HDFS file (CSV or SequenceFile) or Hive, HBase, Accumulo.
It can be set to act as a replicator (incremental processing).
If the data should be processed before be imported in a scalable database, a dedicated spark-streaming job could be implemented.
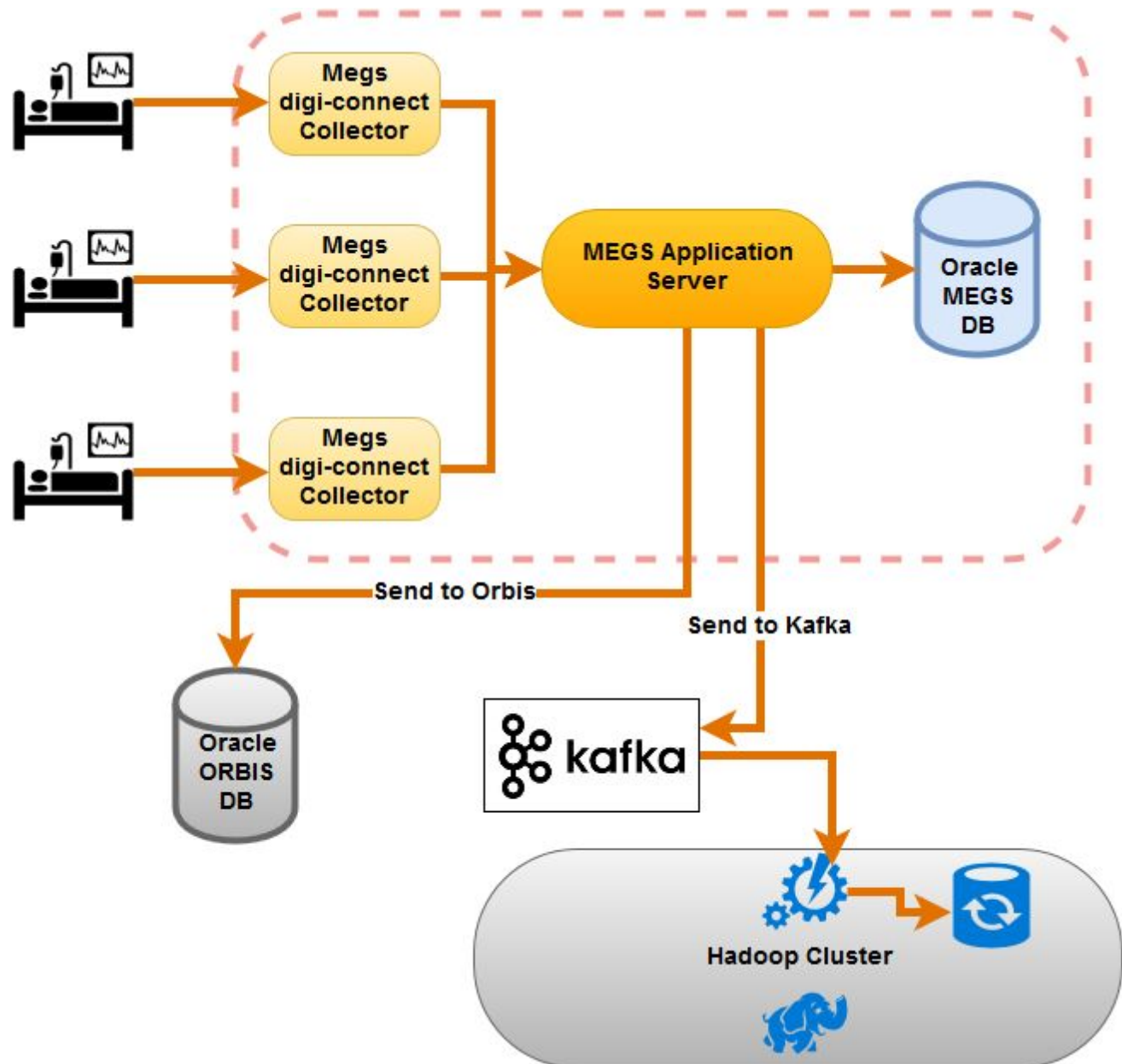

---

[30] ETL = Extract Transform Load

Collect data from a custom MEGS application service

Another option consists in implementing a new service on MEGS application level (as the "send to Orbis" service) which could feed a Kafka or a maprstreams queue.

Data could then be processed by a spark-streaming job as described below.
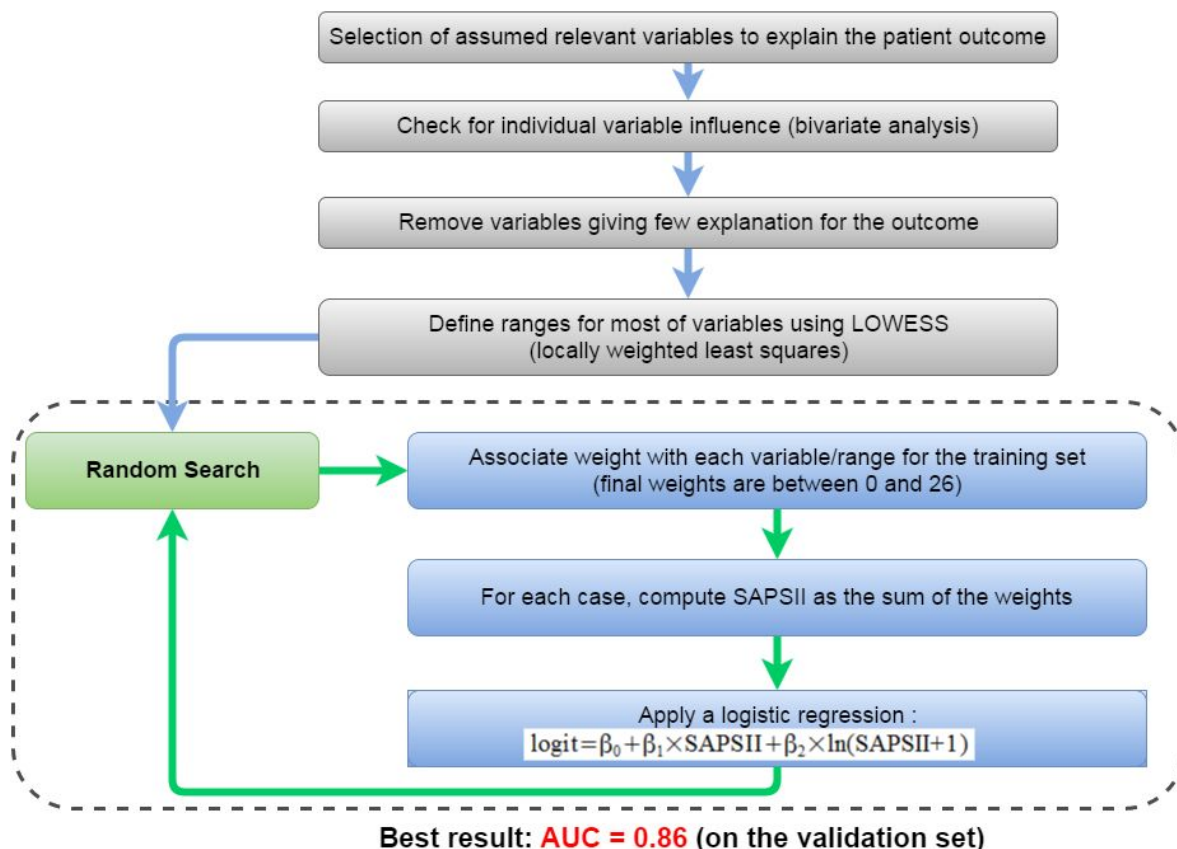
A machine-learning "Proof Of Concept"

Intensive Care Units (ICU) and SAPSII score

Several mortality scores exist and can be used by intensive care units (Apache[31] III, ApacheIV, SAPSII…).
At APHP, for each patient entrance in an ICU, the SAPS[32] II (Simplified Acute Physiology Score v2) score computation is mandatory.
The SAPSII score and related mortality probability [32] have been designed in 1993 with a dataset of 13152 patient cases provided by several European and North-American ICUs.
The methodology applied to build the SAPSII model is presented below:

Best result: **AUC = 0.86** (on the validation set)

The final weights for variables or couples (variable, threshold interval) related with the high score (AUC = 0.88 on the training set and 0.86 on the validation set) are between 0 and 26 (see [32] for details on final weights or following form: http://reaannecy.free.fr/igs2.htm) and the associated logit function is:

$$logit = -7.7631 + 0.0737 \times SAPSII + 0.9971 \times ln(SAPSII + 1)$$

The mortality probability can be computed from logit:

$$Pr(y = \frac{1}{logit}) = \frac{e^{logit}}{1 + e^{logit}}$$

---

[31] Apache = Acute Physiology and Chronic Health Evaluation
[32] SAPS = Simplified Acute Physiology Score

Tools and data sources used to explore APHP data

For our analysis, we have preferably used Drill and Parquet tables which is a comfortable tool to explore and join huge tables.

We have used csv files extracted from Orbis and our second data source is the Wind dataware-house (EDS).

We used python, pyodbc to query the Drill database, pandas and scikit-learn, xgboost, hyperopt, pyriemann for Data-science.

Dealing with SAPSII data at APHP

Thanks to the PMSI (Plan de Médicalisation des Systèmes d'Information), the SAPSII score should be computed for each patient entrance in an APHP ICU.

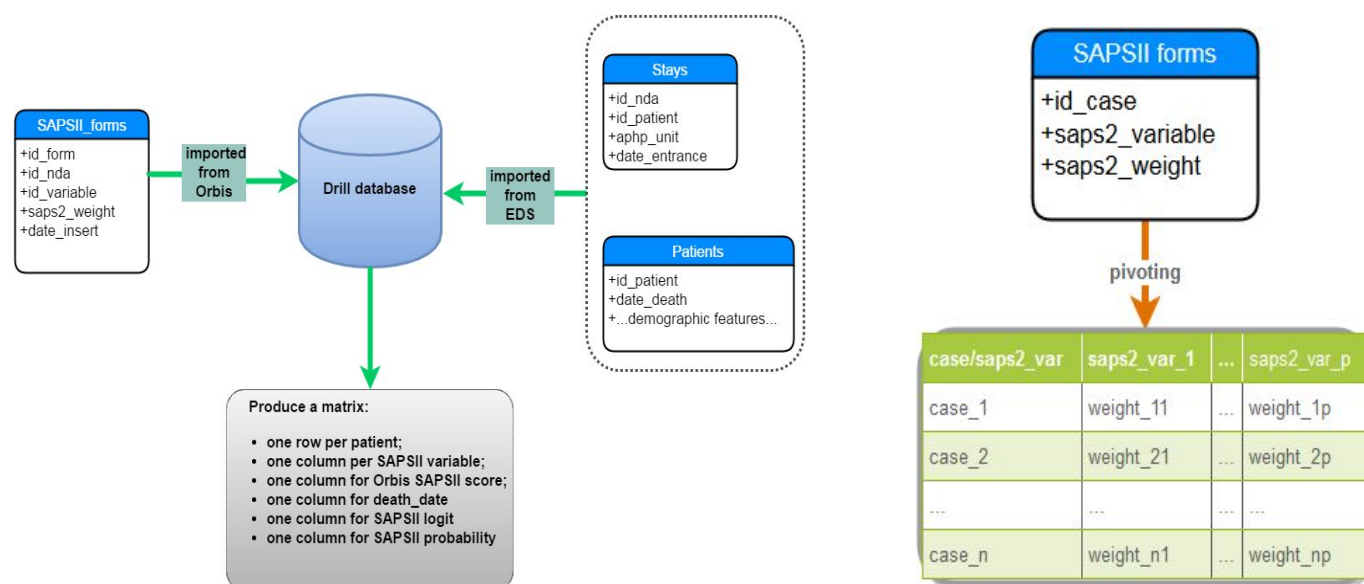The Orbis database includes more than 17000 SAPSII forms.

Unfortunately, the SAPSII variables values are not available and we can only work with weights and SAPSII variables identifiers.

The SAPSII forms explored are extracted from the Orbis database.

We have got a table with 16 rows per case (15 rows for variables and 1 row for related SAPSII score computed by Orbis).

For any case, Orbis triggers an alert once the SAPSII score is higher or equal to 15.

Once joined to Stays and Patient tables, it remains 14715 forms.

Transforming the original data to a matrix (pivoting table with pandas), we can note these elements:

- Potential doubles for several SAPSII forms (it can happen if the form has been filled twice or if a patient has come more than once in the ICU during his hospital stay). Because it occurs rarely (less than 250) and it is difficult to distinguish a double form from multiple ICU entries, we have kept all of them in the final matrix;
- For patients having several entrances in ICU for the same hospital stay and for which outcome is "death", we got several cases with outcome set as "death" only for the last;
- A mortality rate about: 13.34% (for the 14715 forms).

We have enriched the output matrix adding a column for the SAPSII logit and the SAPSII probability of mortality per case.

Regarding the APHP SAPSII score threshold of 15, we got an APHP probability threshold slightly less than 2%.

Hence, we got the following performance for SAPSII on the 14715 forms:

accuracy	26.8%	recall	99.5%
AUC	**0.85**	precision	15.4%

Note that the pertinence of the precision/accuracy measures are hard to appreciate because we cannot insure that the false positive cases finally escape for bad appreciation or because they received efficient cares.

Attempt to improve mortality prediction score with machine learning

Because the measures related with SAPSII variables are not available in the Orbis database, we built our dataset from the SAPSII weights stored in the SAPSII forms.
Each variable will be processed as a categorical variable.

SAPSII variables description

The following table gives an overview of SAPSII variables categorization and meaning:

Variable (Orbis referential code)	Description
Demographic variables	
AGE	Age of patient
Vital signs	
FREQUENCECARDIAQUE	Heart rate in bpm (beats/minute)
PRESSIONARTERIELLE	Systolic Arterial Blood Pressure in mm HG (millimeters of mercury)
TEMPERATURE	Body temperature in C° (celsius degrees)
GLASGOW	A score indicating how well the patient reacts to external stimuli
Oxygenation	
PAO2/FIO2	blood oxygenation over fraction of inspired oxygen (ventilated patients only, for others weight=0)
Renal function	
DIURESE	Urinary output in L/d (liters by day)
UREE	Blood urea in mmol/L or g/L or mg/dL
Chemistry	
BICARBONATEMIE	fraction of the anions in blood plasma
KALIEMIE	potassium concentration in the blood plasma
NATREMIE	sodium concentration in the blood plasma
BILIRUBINE	bilirubin concentration
Other	
GLOBULESBLANCS	White blood cells concentration
MALADIESCHRONIQUES	known chronical diseases (Metastatic cancer, Hematologic malignancy, AIDS)
TYPEADMISSION	type of admission (Scheduled surgical, Medical, Unscheduled surgical)

The measures considered are the baddest in the first 24 hours after ICU entrance.
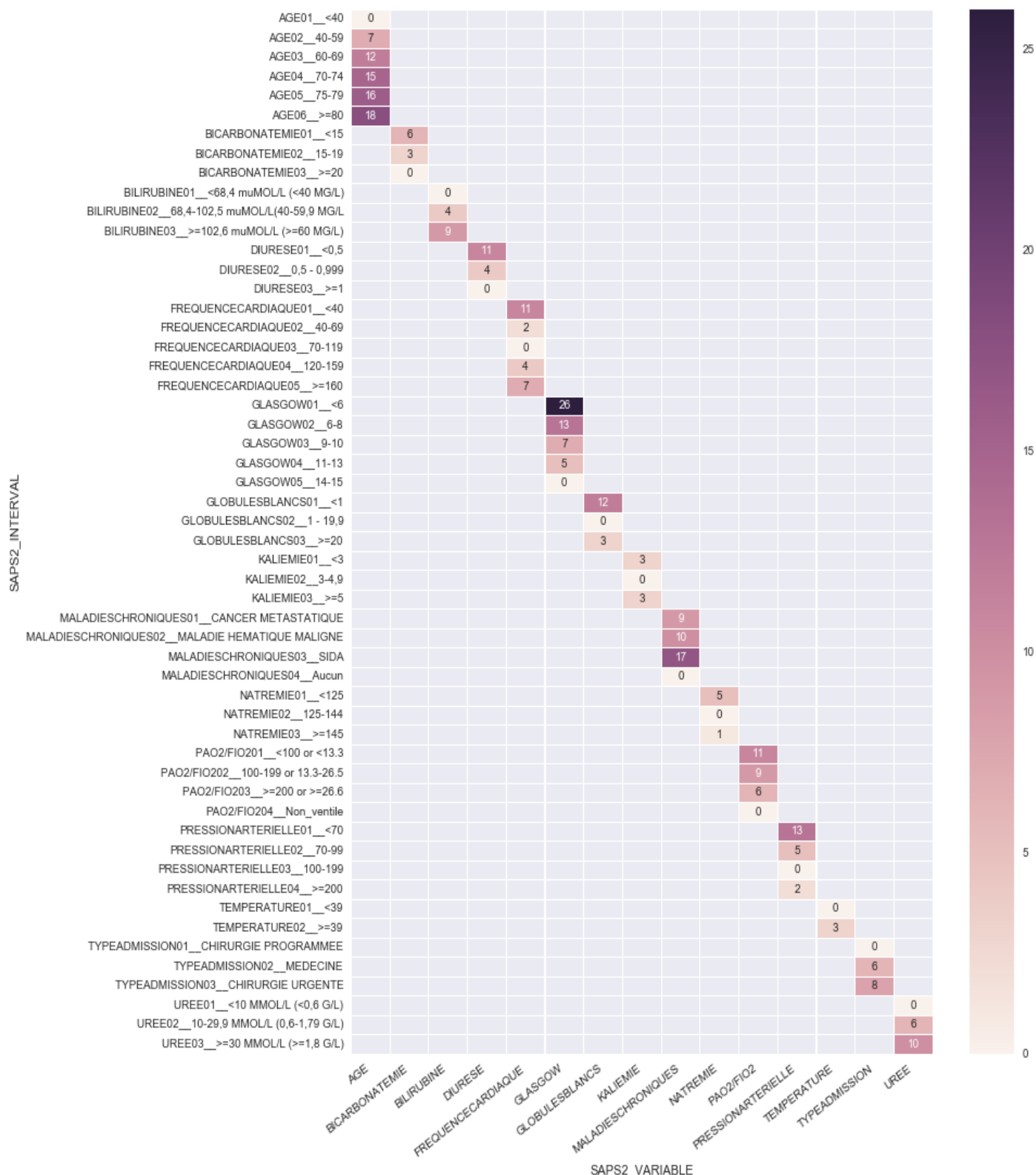
SAPSII variable intervals and weights

Note

the python code related to the figures of this section is available at :
https://github.com/catherineverdiergo/InternShip_python/blob/master/Thesis_special_plots.ipynb

The following diagram gives an overview of the clinical intervals associated with variables and related best weights got by the grid search process during the SAPSII design experiment:

Building a dataset for machine learning

Note
the python code related to the figures of this section is available at :
https://github.com/catherineverdiergo/InternShip_python/blob/master/SAPS2_Forms.ipynb

For most variables, only SAPSII weights, are stored in the Orbis database.
Hence, we have used every variable as categorical.
Weight have been transformed as categories using the sklearn.preprocessing.LabelEncoder class.

Methodological approach

From the 14715 rows available, we have extracted our validation set.
Because our dataset is unbalanced (mortality rate about 13%), we have built it with a similar balance of classes in order to get a representative validation set.
This can be easily performed with the sklearn.cross_validation.StratifiedKFold class.
The mortality rate of our validation set is about 15.4%; its size is 4906 (about ⅓ of the entire set).

On our training set (complementary set of the validation set), we have fit several binary classifiers:
- logistic regression;
- support vector machine;
- decision tree;
- random forest;
- gradient boosting (xgboost package).

To check the relevance of the models and limit overfitting, each of them have been cross-validated with 5 KFolds built with the sklearn.cross_validation.StratifiedKFold class to catch the unbalance of the classes.

The different models have been compared with the classical SAPSII in term of AUC (Area Under the Curve) with the validation set.
The AUC is the referenced score for SAPSII ([32]). It corresponds to the Area under the Receiver Operating Characteristic curve ([33]). It can be a good indicator to determine a threshold arbitrating between TPR (True Positive Rate) and FPR (False Positive Rate).
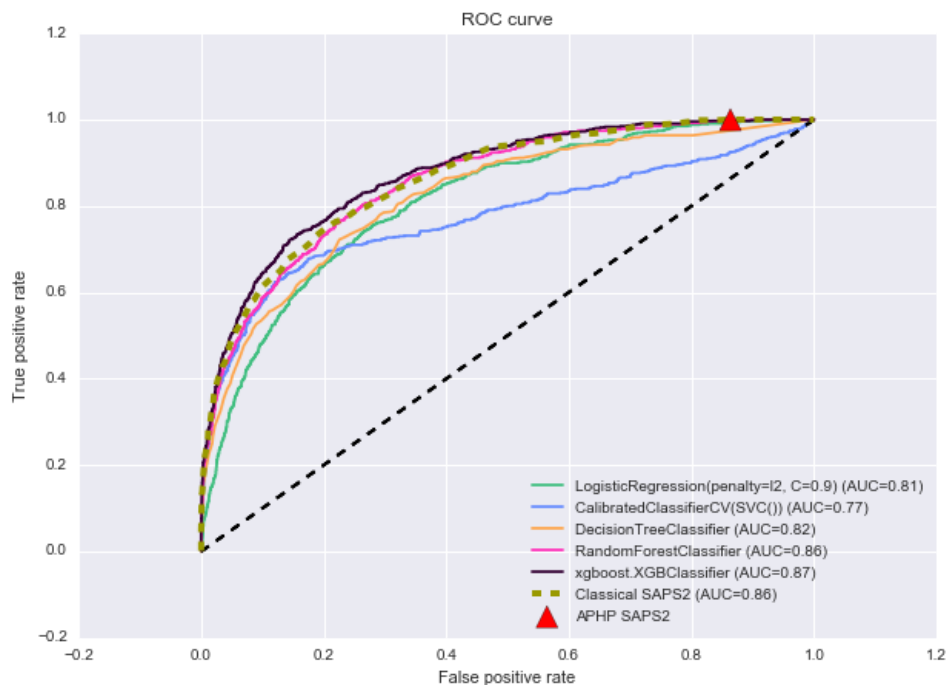
For our validation set, the classical SAPSII scores applied with the APHP threshold (SAPSII = 15) are:

accuracy	27.1%	recall	99.8%
AUC	**0.86**	precision	15.5%

Results

With the dataset built as described above, we got the following scores:

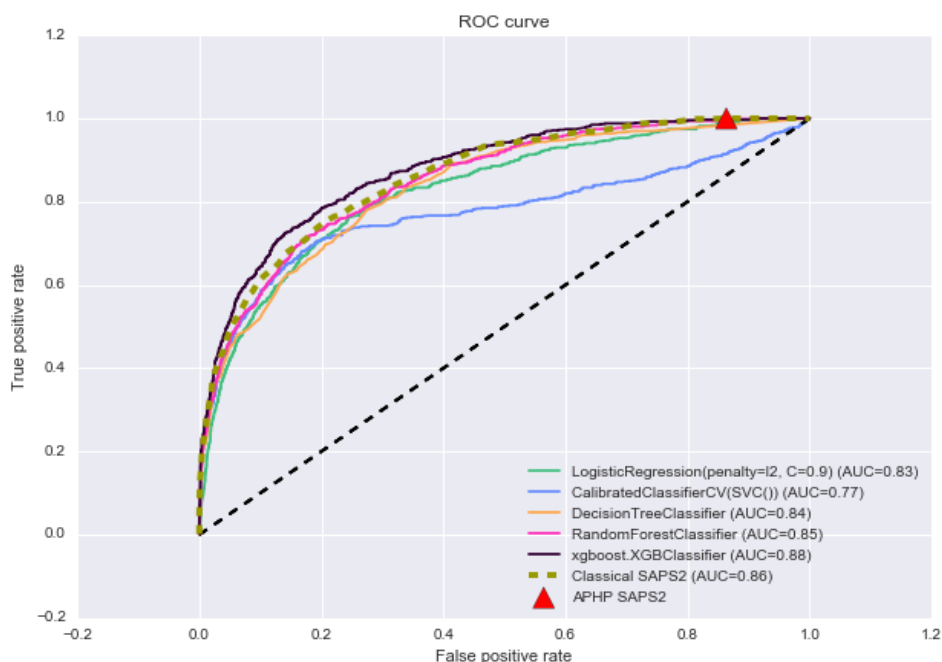| Classifier	AUC
Logistic regression	0.81
SVM	0.77
Decision tree	0.82
Random forest	0.86
Gradient boosting	0.87

Improvement with added features

To improve the performances of the models we have added some new features inherited from 'DIURESE', 'GLASGOW', 'PAO2/FIO2' and 'PRESSIONARTERIELLE'[33]: for each of these variables we have added by row a counter of related category occurrences in the training set, the square-root and the neperian logarithm of the categorical value.

With this second dataset, the scores have became:

Classifier	AUC
Logistic regression	0.83 ↗
SVM	0.77 →
Decision tree	0.84 ↗
Random forest	0.85 ↘
Gradient boosting	0.88 ↗

---

[33] for each feature x in ['DIURESE', 'GLASGOW', 'PAO2/FIO2' and 'PRESSIONARTERIELLE'] added features are: count(x), sqrt(x), ln(x+1)

Tuning best classifier

Note
the python code related to the figures of this section is available at :
https://github.com/catherineverdiergo/InternShip_python/blob/master/mortality_prediction.py

As shown above, the best AUC score we have got with our dataset has been provided by a Gradient Boosting classifier (xgboost package). The scores already presented have been rounded to nearest 0.01. Our best score is actually 0.875153. We will attempt to improve it.

It appears possible to couple scikit-learn GridSearchCV with a cluster using Spark and the spark-sklearn package from Databricks.
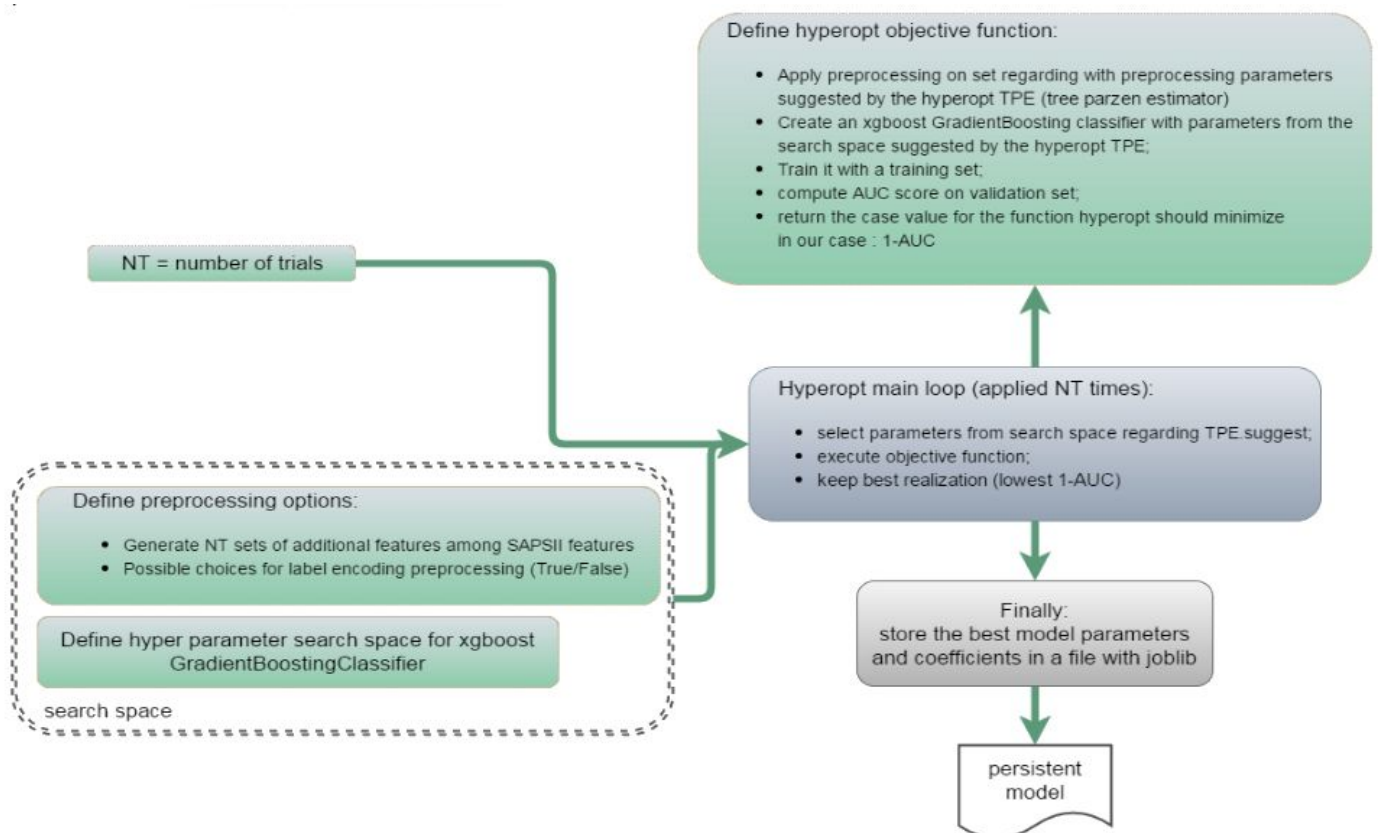The aim is to distribute GridSearch parameter combinations in a Spark cluster (instead distribution on the multiple CPUs with the n_jobs=-1 parameter on a stand alone machine).
Although this option appears feasible with the MapR distribution (as shown in the following MapR-demo project: https://github.com/mapr-demos/spark-sklearn-airbnb-predict), we did not apply it for the following reasons:
1. The recommended Spark version appears not clear (>=2.0 on https://github.com/databricks/spark-sklearn, wheras Spark1.6.1 is provided with the MapR5.1 distribution we are using);
2. Our cluster, installed for tests, was reduced to a single server, thus, the gain could not be higher than using directly the available CPUs (6) directly from a python process.

We have decided to test the Hyperopt python library. The Hyperopt package provides a cross-validated hyper parameter optimization algorithm based on a tree Parzen estimator. It is supposed to be more efficient than a GridSearchCV and easier to reproduce than a RandomizedSearchCV.
We have use it with the matrix generated from SAPSII forms extracted from Orbis, with variable preprocessing (Adding new features randomly / LabelEncoding SAPSII weights or keep original SAPSII weights) and a parameter space related to xgboost GradientBoosting.

Hyperopt main loop

Applying this process with several search spaces, we have saved a classifier providing an AUC about **0.881067** (versus 0.860668 for the classical SAPSII AUC on the same set), for the validation set considered in our hyperopt trials, with related preprocessing:

- **no label encoding (keeping original SAPSII weights);**
- **Adding count(x) and ln(x+1) for following features: ['FREQUENCECARDIAQUE', 'BILIRUBINE']**

Related classifier has been saved (parameters and coefficients) in:

https://github.com/catherineverdiergo/InternShip_python/blob/master/best_saps2_xgb_881066691088.pkl

Assessment

Our study on AP-HP SAPSII forms has shown that despite its age (1993), the SAPSII score remains a reliable indicator.

The variables, thresholds and weights defined in the original study by clinicians have maintained a high level of relevance.

As shown on the ROC curves graphics, it seems complicated to be able to define a new APHP threshold from which we got a recall close to 100%. However, the gradient boosting classifier, with the SAPSII forms data exported from Orbis, seems able to produce a more precise probability of mortality of ICU patients.

Persistant best classifier could be integrated in Orbis ICU application to get another opinion about the mortality risk for the ICU patients.

Working with ICU waves

At APHP, in two of the 39 hospitals, a ICU's dedicated module has been deployed. This plugin allows the recording of vital signs in the Orbis database.

In this second stage of the POC, we will explore this first bag of data and study the covariances between vital signs waves following an idea suggested by Louis Mayaud (Research Director at Mensia Technologies SA).

ICU waves exploration guideline

Louis Mayaud current research field is related with neurophysiology, but he has a strong knowledge in data science applied to ICUs mortality prediction ([34]).

The main idea suggested by Louis Mayaud is to explore vital signs waves covariances in a Riemannian space and check if they hold discriminant power to predict the outcome of an ICU patient.

Hence, we will attempt to deal with the following action plan with APHP waves data:

1. Collect measures provided by ICUs and related with first 24 hours of an entrance;
2. Keeping vitals signs (i.e.: HR (Heart Rate), RR (Respiration Rate); ABPD (Diastolic Arterial Blood Pressure) and ABPS (Systolic Arterial Blood Pressure) as variables of interest, check how many points we have regarding to the population (size of sample);
   Note: the number of points should apply for each variable of interest.
3. Keep only patients more than 15;
4. Remove aberrant values from each wave;
5. For the resultant sample, interpolate waves with a relevant period in order to make them all synchronous;
6. Attempt to make spectral analysis;
7. Build a set of matrices with waves (one per subject);
8. Compute the covariance matrices (with the pyriemann toolkit [36]);
9. Compute and plot the mean-covariances matrices (by class) to check the potential power of discrimination of our couple of model and data sample;
10. If the results appear relevant, attempt to classify data with: or the MDM (Minimum Distance to Mean) algorithm in the Riemann space, or with any other classifier using euclidean distance in the tangent space;
11. If step 9 provides results, attempt to retrieve related SAPSII forms in the set of SAPSII forms used above and enrich the model with features generated with waves (mean, variance, riemann distance to classes centroids got with the pyriemann Kmeans algorithm).
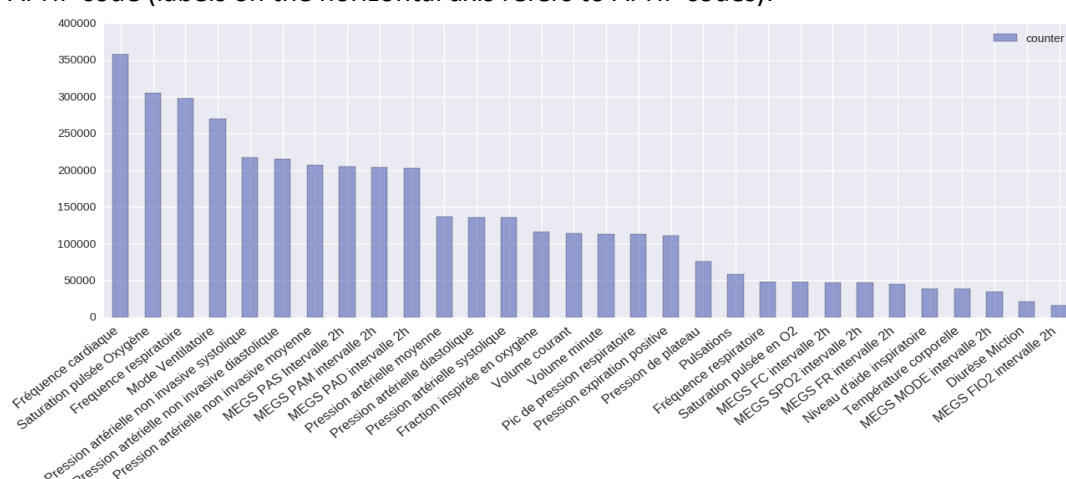
Filtering APHP set of measures

Note
the python code related to the figures of this section is available at :
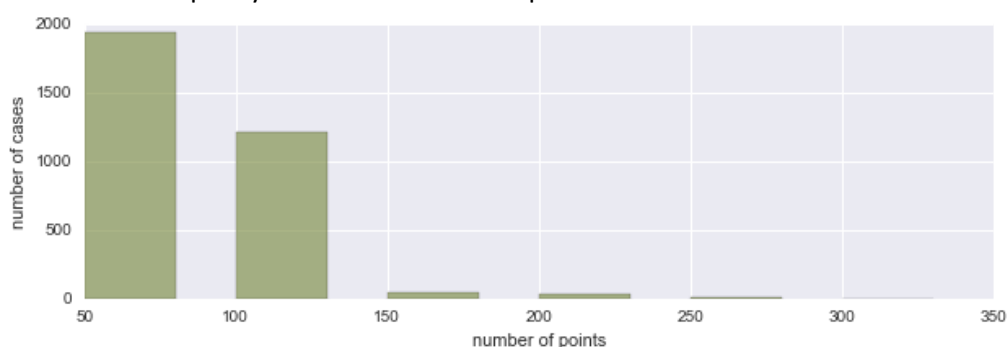https://github.com/catherineverdiergo/InternShip_python/blob/master/Measures-Cases%20filtetring.ipynb

Once extracted the measures available on the APHP Orbis information system, we got a set of more than 38 million of rows (csv file). Each measure is related to an APHP code. A vital sign measure could be linked to several codes: indeed, as an example, the arterial blood pressure may be captured with an invasive or a noninvasive way. We also note that data can be provided by several sources as 'Fréquence cardiaque' and 'MEGS FC intervalle 2h'. We have used the measure codes reference table from Orbis to identify our variables of interest.
Once measures filtered to keep only those done in the first 24h after entrance, we got the following number of measures by APHP code (labels on the horizontal axis refers to APHP codes):

Keeping only our variable of interest (HR, RR, ABPD and ABPS) for patients more than 15, we can show how many cases remain in our future sample by minimum number of points available for each cue:

If we want more than 100 points by wave, our sample size will be less too small.
Thus, we will include in our set subjects for whose we have 96 points or more by vital sign for the first 24h in ICU. That means we will have about one measure by 15 minutes. This period, for vital time series is probably too broad in order to process a relevant signal analysis ([35] § "Short- Versus Long-Lasting BP and HR Recordings").
Our Day-one data covers the period from 2012-09-05 to 2016-27-09.
The size of our set is 1446 and related mortality rate: 18.6%.
The mortality rate is higher than the one got with SAPSII forms dataset: indeed, keeping cases stayed more than 24h in ICU can refer to more serious cases.
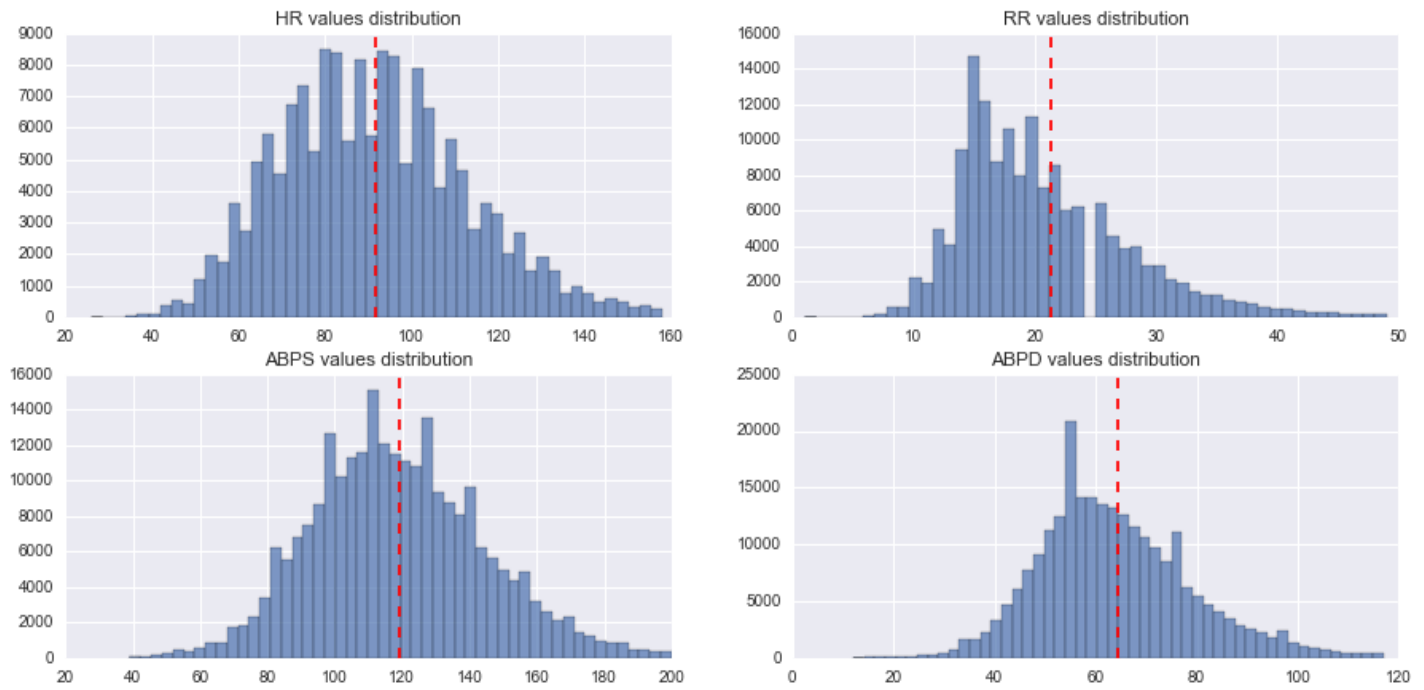The final number of measures we can explore is 813737.

We removed aberrant values as follow:
- remove negative or nul values;

- for each distribution, remove values distant from more than 3-std[34] to the mean.

Once this operation performed, the distributions of the wave variables look like:

Values distributions for vital signs available
(red dashed vertical lines are related to the means of the distributions)

During this operation, one of the patients has been subtracted from the set because his relevant number of measures became too low (less than 96) ⇒ after this process, our dataset size is 1445.

The age structure of the set by outcome is:

---

[34] std = standard deviation

Waves interpolation

Note
the python code related to the figures of this section is available at :
https://github.com/catherineverdiergo/InternShip_python/blob/master/Waves%20processing.ipynb

The goal of this process is to get the same number of points for all waves and all subjects included in our set.
We set the final number of points to 96, because this is the number of measures we have for most of cases.
A linear interpolation of the waves is performed with the pandas package.
96 points of measures per the first 24h in ICU means a frame rate of 15 minutes (900 s).

The figure below shows several example of original and interpolated waves:

outcome: dead   *outcome: alive*

At top, we can see the original waves as collected from the set extracted from Orbis.

A bottom, we have displayed the same waves after a linear interpolation.

Standards of variability - Spectral analysis

Since the 70s, research, specially in cardiology, have established a relationship between the heart rate variability (HRV) and the autonomic nervous system (ANS).

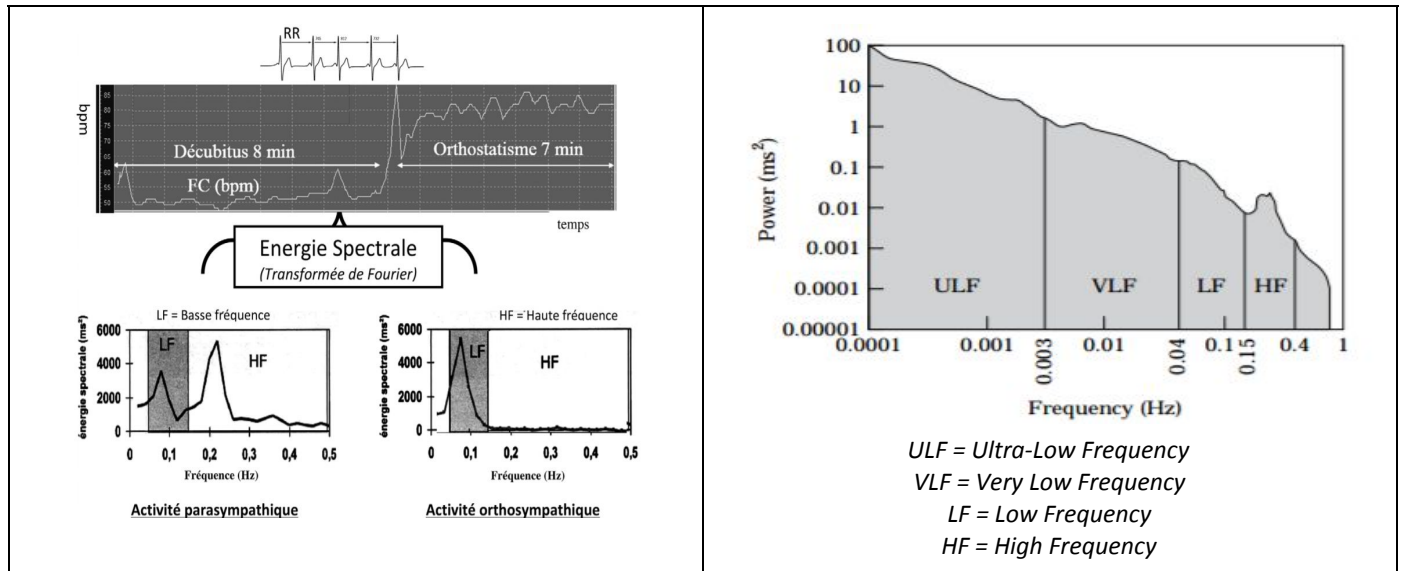Recognized as a standard since 1996, the Task Force of the European Society of Pacing and Electrophysiology has defined what are the frequency ranges (for a heart rate spectrum) representative of either sympathetic either parasympathetic[35] activities.

ULF = Ultra-Low Frequency
VLF = Very Low Frequency
LF = Low Frequency
HF = High Frequency

Looking at the graphic above, to be able to catch sympathetic and parasympathetic activities in the heart rate wave, we need capture frequencies from 0.04 Hz to about 1 Hz.

This cannot be possible with the broad period between 2 measures in the APHP waves (~ 900 s).

With such a period, regarding the Shannon-Nyquist theorem, the highest frequency which can be caught is:

$$Nyquist_{frq} = \frac{1}{2} \times \frac{1}{period} = \frac{1}{2} \times \frac{1}{900} \approx 0.6 \; mHz$$

It seems not possible to apply filters on waves to split them on sympathetic/parasympathetic components.

An example of spectrum (FFT normalized and frequency scale stopped at the Nyquist frequency) got from the APHP samples is:

00084806812015-12-15 -- Outcome: alive

---

[35] The sympathetic activity of the ANS is high when a subject has to react to external stimuli (stress, perceived threat) whereas the parasympathetic activity is related with the organism self-regulation (digest, physiological needs). These two activities are always complementary balanced.

To be able to filter ICU waves with appropriate frequency ranges, it would be preferable to record measure each:

$$\frac{1}{2 \times 0.4} = 1.25s$$
.

To be able to apply an efficient high-pass filter on cues (filtering ULF and VLF), the recommended period for records should be about:

$$\frac{1}{2 \times 0.04} = 12.5s$$

Covariance matrices definition and properties

We note X_i^S the vector describing the ith wave for the subject S.
We note COV^S the covariance matrix for the subject S.

and for discrete random variables, we can use the unbiased empirical covariances:

$$COV_{ij}^S = COV_{ji}^S = \frac{1}{np-1}\langle(X_i^S - \mu_i^S)|(X_j^S - \mu_j^S)\rangle$$ and $COV_{ii}^S = Var(X_i^S)$ (empirical variance actually)

$$where: \begin{cases} \langle.|.\rangle & inner\ product; \\ np & number\ of\ points\ for\ waves; \\ \mu_k^S & empirical\ mean\ of\ measures\ for\ wave\ k \end{cases}$$

COV^S is a symmetric positive definite matrix (i.e. its eigenvalues are all non-negative).

If nw is the number of waves considered, COV^S is entirely defined by $\frac{nw \times (nw+1)}{2}$ coefficients.

Moreover, the Cauchy-Schwarz inequality ensures $|\langle X_i^S | X_j^S \rangle| \leq ||X_i^S||_2^S \cdot ||X_j^S||_2^S \ \forall i,j \in \{1,..,nw\}$

and also: $|COV_{ij}^S| \leq \sqrt{Var(X_i^S).Var(X_j^S)}$ ([37] p 227, 228)

Thus, the COV^S matrices can be represented by points of $\mathbb{R}^{\frac{nw(nw+1)}{2}}$ constrained in a subset of the space (hyper cone of SPD matrices).

Evaluate the discriminant power of the covariances with simplified models

Clinical researchers (e.g. [38]) have already demonstrate the role of vital signs variability to determine the risk of death and the outcome for a patient (as example, the bad heart rate during the first 24h in ICU is one of the features involved in the SAPSII score).
We have elaborated two models to see how high the waves covariances have effect.

On the previous built dataset of waves (1445 subjects), following transformations have been applied:
1. measures scaling with the sklearn.preprocessing.MinMaxScaler class;
2. computing of the covariance matrices with the pyriemann toolkit thanks to the fit_transform method of the pyriemann.estimation.Covariances class.

Then, two new datasets (size 1445) have been made:
- the first-one with the variances of waves as features (nw=4 features)
  for each subject, the vector of features is: **Tr(COV ^{s})** (the trace of the covariance matrix);
- The second with variances and covariances ($\frac{nw(nw+1)}{2}$ features $\Rightarrow$ 10 features)

These two datasets are then splitted in train and valid set including identical cases for both (respecting a relevant balance of classes in the same way that we did above with SAPSII forms data).
To finish, we apply a 5-cross-validated logistic regression on outcome prediction to check if the covariances can can improve discrimination.
Our achievements give us:

Dataset 1: waves variances only		Dataset 2: waves variances and covariances	
accuracy	81%	accuracy	81%
AUC	0.6	AUC	0.603

Adding covariances improves very slightly the AUC, but the gain, appears quite poor.

The Riemannian distance in the SPD space ([37] p 230)

The space of SPD matrices is a differentiable Riemannian manifold designed by $\mathcal{M}$.
On the $\mathcal{M}$ manifold, the geodesic is the curve with minimum length joining 2 points Ω and Φ (related to 2 SPD matrices). For the points considered, the geodesic (a set of points of the manifold) is defined by:

$$\gamma_R(\beta, \Omega \to \Phi) = \Omega^{\frac{1}{2}}(\Omega^{-\frac{1}{2}}\Phi\Omega^{-\frac{1}{2}})^{\beta}\Omega^{\frac{1}{2}} \quad \beta \in [0,1]$$

with β : step size, $\Omega^{\frac{1}{2}}.\Omega^{\frac{1}{2}} = \Omega$ and $\Omega^{-\frac{1}{2}}.\Omega^{-\frac{1}{2}} = \Omega^{-1}$ (Ω and Φ as SPD matrices).

Then the Riemannian distance is defined as the length of γ_R as:

$$\delta_R(\Omega \leftrightarrow \Phi) = ||ln(\Phi^{-1}\Omega)||_F = \sqrt{Tr(ln^2(\Lambda))}$$

with $Tr(\Lambda) = eigenvalues[\Phi^{-1}\Omega; \ \Omega^{-1}\Phi; \ \Phi^{-\frac{1}{2}}\Omega\Phi^{-\frac{1}{2}}; \ \Omega^{-\frac{1}{2}}\Phi\Omega^{-\frac{1}{2}}]$

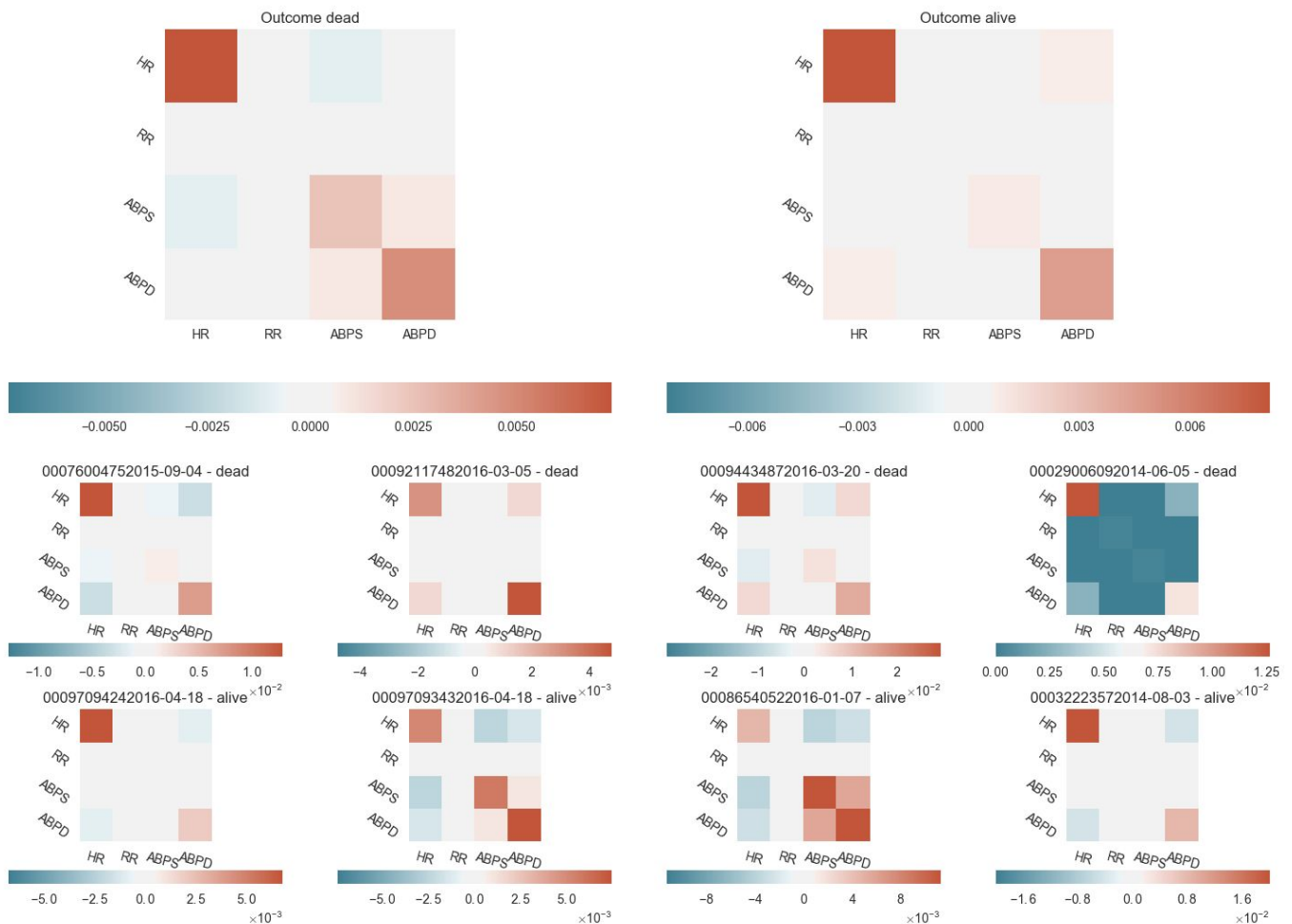and $||\cdot||_F$ the Frobenious norm.

The Riemannian distance holds interesting properties (see [37] p 231).

Mean covariance matrix - geometric mean of points on the Riemannian manifold

Using the Riemannian distance as defined above, it could be useful to be able to define a mean of a set of points on the manifold. Pennec et al., 2004; Manton, 2004 ([37] p 236) have proposed an algorithm to compute it regarding expected properties for this notion, which is available in the pyriemann toolkit.

We have computed the mean covariance matrices by class, to check if we can see any relevant trend in covariance values to separate the two possible outcomes. In that way, covariance matrices have been computed with regularization (lwf=Ledoit and Wolf regularization or oas=Oracle Approximating Shrinkage), because for several cases in our set, empirical covariance matrices are not full rank and the eigenvalues cannot be computed.

Mean covariances (Oracle Approximating regularization)

In the figure above, the mean covariance matrices by outcome are displayed up. Several examples for different outcomes are displayed down (dead outcomes for the first row, alive outcomes for the second row).
On the mean covariances level, we can see a trend of opposite variances between HR and ABPS, but the trend is very weak.

The Minimum Distance to Mean classifier (MDM)

Note
For this section, our results, are not included in the document, because they are not relevant enough.

Using the covariance means in the Riemannian space, Marco Congelo and al. have designed a very simple multiclass supervised algorithm called Minimum Distance to Mean or MDM (see [37] p 201). The initial context of this work was BCI (Brain Computer Interface) and the aim was the identification of a body simple action from the brain activity recorded by several electrodes.

The principle of the algorithm is the following:

Training stage:
- For each set of synchronized cues available in the training set, compute the covariance matrix;
- For each class, compute the mean covariance matrix (or point of $\mathcal{M}$), denoted $\overline{COV}_c$ for class c.

Validation stage:
- For each set of synchronized cues available in the validation set, compute the covariance matrix denoted COV_i for the ith example;
- Choose the predicted class regarding the lowest riemannian distance from the considered point to the mean points computed during the training stage: $\underset{c}{argmin}\ \delta_R(\overline{COV}_c \leftrightarrow COV_i)$

For such classifier, the training stage could be fast, either when the number of waves is high: indeed, the riemannian distance is congruence-invariant ([37], (282) p 231), and hence stays the same after a transformation such as PCA.

Working with covariance matrices, it is also possible to try any classifier in the euclidian tangent space.
In the tangent space, classical classifiers can improve scores, provided that the considered points are located in a small part of the Riemannian manifold (if the points are widely scattered, the tangent space is not an euclidian space).

Conclusion

On the MapR distribution evaluation level

With our experiments and benchmarks, the WIND department has now an overall picture of the tools available on an Hadoop platform. It is satisfied of the performances of the framework to query large set of data.

We have noted that an Hadoop cluster supplies a lot of relevant tools and formats to deal, filter, clean and normalize raw data (specially when they are big) and transform it in exploitable datasets for data-science purposes.

However, the MapR distribution appears not enough relevant for the interoperability purpose with i2b2.
Hence, WIND will probably start an evaluation of the Horthonworks solution with Hive/ORC on Tez, instead of Parquet on Drill, to check if it could achieve all its requirements.

On the Data-science POC level

Working with the SAPSII ICUs forms, we have succeeded to improve the referenced AUC score to compute the probability of mortality with one of the new machine learning algorithm and it appears relevant to introduce these techniques in future data exploration works at APHP.

The project related to ICU waves exploration did not produce any interesting result because despite a high volume of data extracted from Orbis, the frame rate between points in the cues was too large enough.
However, we have identified the right frequency to store ICU wave points in order to explore them more precisely: this is an interesting result to design the proposal to store MEGS transient data in the Hadoop data lake. Once this project will be operational, APHP will be able to use the work carried out during this internship to potentially produce more relevant results.
Then, as the individual BCI models, we can imagine in the future some specific patient warning models based on continuous vital waves monitoring.

References

[1]   DÉPLOIEMENT GÉNÉRALISÉ DU SYSTÈME D'INFORMATION PATIENT ORBIS

[2]   Présentation de l'entrepôt de données de santé APHP

[3]   i2b2 official web site

[4]   Vinod Kumar Vavilapalli & al. "Apache Hadoop YARN: Yet Another Resource Negotiator" (2013)

[5]   Douglas Eadline : The Yarn Invitation

[6]   Jeffrey Dean and Sanjay Ghemawat : MapReduce: Simplified Data Processing on Large Clusters (2004)

[7]   Hadoop v2 documentation : developing Yarn applications

[8]   Octo technologies : Hadoop feuille de route (12/2015)

[9]   Hadoop v2 documentation: High Availability

[10]   Christophe Paragaud (Ippon Technologies) : Dans la jungle des distributions Hadoop (2013/2014)

[11]   The Forrester Wave™: Big Data Hadoop Distributions, Q1 2016
        by Mike Gualtieri and Noel Yuhanna January 19, 2016.

[12]   The Hadoop Distributed File System
        Konstantin Shvachko, Hairong Kuang, Sanjay Radia, Robert Chansler (May 2010)

[13]   MapR-FS vs HDFS
        Dale Kim (June 2016)

[14]   Because Hadoop isn't perfect : 8 ways to replace HDFS
        Derrick Harris (July 2012)

[15]   CEPFS documentation : USING HADOOP WITH CEPHFS

[16]   MapR-DB : The MapR Converged Data Platform

[17]   Kafka: a Distributed Messaging System for Log Processing
         Jay Kreps, Neha Narkhede, Jun Rao (LinkedIn) 2012

[18]   MapR Streams : The MapR Converged Data Platform

[19]   Uri Laserson : A guide to python Frameworks for Hadoop (January 2013)

[20]   Hadoop documentation : Hadoop Streaming

[21]   The Twitter Scalding project on GitHub

[22]   Scala vs Python for Spark (February 2015)

[23]   Python vs. Scala vs. Spark (January 2015)

[24]   Pypy site and Pypy reference papers

[25]   Benchmarks: Spark with CPython vs Spark with pypy

[26]   Major Technical Advancements in Apache Hive
         Yin Huai1 & al. (April 2014)

[27]   Dremel: Interactive Analysis of Web-Scale Datasets
         Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton,
         Theo Vassilakis Google, Inc. (September 2010)

[28]   Time Series Databases: New Ways to Store and Access Data
         Ted Dunning and Ellen Friedman (October 2014) - Ed O'Reilly

[29]   New Designs Using Apache Kafka and MapR Streams
         Ted Dunning and Ellen Friedman (May 2016) - Ed O'Reilly

[30]   OpenTSDB site

[31]   Using Common Table Expressions to Build a Scalable Boolean Query Generator for CDW
         Daniel R. Harris, Darren W. Henderson, Ramakanth Kavuluru, Arnold J. Stromberg, and Todd R.
         Johnson
         IEEE JOURNAL OF BIOMEDICAL AND HEALTH INFORMATICS, VOL. AAA, NO. BBB, DECEMBER CCC
         CDW = Clinical DataWarehouse

[32]   A new Simplified Acute Physiology Score (SAPS II) based on a European/North American
multicenter
          study.
         Le Gall J., Lemeshow S., Saulnier F. (JAMA Dec 1993)

[33]   ROC Graphs: Notes and Practical Considerations for Data Mining Researchers
         Tom Fawcett (HP Laboratories - January 2003)

[34]   Prediction of mortality in septic patients with hypotension. DPhil. University of Oxford.
        Louis Mayaud (2014)

[35]   Spectral Analysis of Blood Pressure and Heart Rate Variability in Evaluating Cardiovascular
        Regulation
        A Critical Appraisal
        Gianfranco Parati, J. Philip Saul, Marco Di Rienzo and Giuseppe Mancia (1995)

[36]   pyRiemann: Biosignals classification with Riemannian Geometry

[37]   EEG Source Analysis
        Marco Congedo (2013)

[38]   Variabilité de la fréquence cardiaque : un marqueur de risque cardiométabolique en santé publique
        Jean MARSAC (Bull. Acad. Natle Méd., 2013, 197, no 1, 175-186, séance du 22 janvier 2013)

Appendix A : MapR development tools classification

This appendix intends to give an overview of the layers or the components available in the MapR Community Edition version 5.1. Some of them have not been used or tested during the internship session because there are several bugs in this freemium distribution and for some cases, we have not find any workaround.

Tool	Description
Main web applications	
MapR admin tool	A web application quite poor to manage the cluster MapR specific Useful to start/stop components and monitor the resources Available on HTTPS (port 8443)
HUE	Hadoop User Environment A unified application to pilot several Hadoop applications, browse MaprFS, track and plan MapReduce jobs. Rather friendly. Based on the Django framework. Available on HTTP (port 8888)
JobTracker UI	To follow-up MapReduce jobs (redundant as HUE) Available on HTTP (port 8088)
Spark UI	To follow-up Spark jobs Available on HTTP (port 8080)
Drill UI	Drill administration and query tool Available on HTTPS (port 8047)
Scripting tools	
Hive	A high level tool to query 'sql like' MaprFS data Transforms queries in MapReduce jobs or uses underlying DBMS indices for several queries HiveServer2 listen on port 10000 by default (has been changed to 10001 because conflict with another application) Can be used from HUE (change the Hive2 port in the HUE configuration too) Can be used in RAM with Spark
Pig	A scripting tool to transform and import data into maprFS or Hadoop databases Implements users scripts as MapReduce jobs. Scripts are transcripted as actions into a DAG[36]. Can be used from HUE
Drill	A MapR open-source implementation of Google BigQuery Allows to build complex queries from multiple sources in distributed RAM
Oozie	A tool to build (pipeline) and schedule tasks Could be pilot from HUE with a powerful UI Does not work in the Mapr 5.1 CE (issue with maprfs)
Tools for experienced developers	
MapReduce	Custom MapReduce jobs can be implemented with Java, python, R, Scala, C, C++ …
Spark	Spark is a very rich tool executing jobs in distributed blocs of RAM Spark is implemented in Scala It allows to : • Manage data with a MapReduce programming model (actions to perform previously stored in a DAG[10]) • Manage mini-batch streaming data • Implementation of machine learning processes (MLlib)
Yarn	A communication layer with (HDFS/MapRFS) and application implementing a specific Java interface

---

[36] DAG = Direct Acyclic Graph

Appendix B
MapR development environment for eclipse

The Eclipse IDE can be used essentially for Java or Scala development.

In this appendix, the installation and usage of the standard Eclipse IDE for Java are supposed to be known and available. The creation of Eclipse projects as well as Maven projects must be mastered.

Creating eclipse projects for the MapR cluster
To be able to create programs applicable for the MapR cluster, the best way is to perform the steps exposed in the following MapR guideline: https://www.mapr.com/blog/basic-notes-on-configuring-eclipse-as-a-hadoop-development-environment-for-mapr   Once this installation is completed, you will be able: - to send MapR commands to the cluster from your computer (with MapR client);   - to create and build Java programs (all required Java libraries are available with the MAPR_HADOOP User library created in Eclipse);   - be able to deploy your package on the cluster from your computer[37]

If you prefer Scala
To be able to develop and compile with Scala in an Eclipse IDE, install the Scala IDE for Eclipse from: http://scala-ide.org/docs/current-user-doc/gettingstarted/index.html   Take care on the scala version to compile your packages for the cluster: the scala IDE refers to the scala2.11 compiler and the scala libraries available on the Hadoop cluster (with the Spark application) is scala2.10.   So you should change the scala compiler in your project (project/properties/scala compiler)

Allow the cluster to run MapReduce jobs in scala

Scala2.10 libraries are available by default for any spark job deployed. However, there are not in the classpath for the hadoop command. Hence, if you create a MapReduce job in scala, check if the following libraries: - scala-compiler.jar;   - scala-library.jar;   - scala-reflect.jar   are in the /opt/mapr/hadoop/hadoop-2.7.0/share/hadoop/common/lib on the cluster
 If they are not apply the following commands on the cluster:
 	
 `$ cd /opt/mapr/hadoop/hadoop-2.7.0/share/hadoop/common/lib`
 `$ ln -s /opt/mapr/spark/spark-1.6.1/scala/lib/scala-compiler.jar`
 `$ ln -s /opt/mapr/spark/spark-1.6.1/scala/lib/scala-library.jar`
 `$ ln -s /opt/mapr/spark/spark-1.6.1/scala/lib/scala-reflect.jar`

Maven MapR projects
To test two different streaming architectures, we have customized a MapR demo built with maven: https://github.com/mapr-demos/mapr-streams-sparkstreaming-hbase   With the pom.xml of the project, all libs supplied in the MapR client are downloaded by Maven from the MapR Maven repository (http://repository.mapr.com/maven/).

---

[37] This functionality has been tested from a Windows7 machine. We were able to deploy properly from the command line but we do not success to perform it from Eclipse.

MapReduce with scalding to avoid the "boilerplate code"

Coding MapReduce jobs in Java implies to implement special interface (Mapper and Reducer).
For two different jobs, we often get a verbose code with a lot of quite identical lines, specially when input or output files have identical structures. This aspect of programming is named "boilerplate code" in computer programming (see: https://en.wikipedia.org/wiki/Boilerplate_code).
To reduce the boilerplate code in MapReduce code and get a code more readable, Twitter has implemented the scalding toolbox.
During the internship at APHP, we have tested scalding to implement several jobs.
Our project can be seen at: https://github.com/catherineverdiergo/scalding-examples

As an example, here is a WordCount job implementation with scalding:

```
import com.twitter.scalding.{Job, Args, TextLine, Tsv}

class WordCountJob(args : Args) extends Job(args)  {
  TextLine( args("input") )
  .flatMap('line -> 'word) { line : String =>
  line.toLowerCase.split("[<>/\\s\"\'`(),;:\\.=-]+")
  .filter { _.length() > 3 }
  .filter { _.matches("[a-zA-Z]+") }
  }
  .groupBy('word) { _.size }
  .write( Tsv( args("output") ) )
}
```

The input is a text file.
The output is a Tsv file.
Each line is transformed as word list (word separation with a regular expression) ⇒ Map stage
The word list is filtered (remove words having less than 3 characters) ⇒ Map stage
The word list is filtered a second time (get only words with alphabetic characters) ⇒ Map stage
Then the number of occurrences of each word is produced with a 'group by' ⇒ Reduce stage

Note that the 2 filter calls can probably be factorized in a single filter.

This code can be compared with Java WordCount examples available at:
https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html

Scalding output formats

Scalding is supposed able to produce a lot of output formats.
With the current version, we did not find any example to produce Parquet outputs

Scalding running modes / scalding REPL

Scalding can be executed in local mode as in distributed mode (throughout an HDFS cluster).
The frameworks embeds a REPL tool, which can be useful to prototype jobs (see:
https://gist.github.com/johnynek/a47699caa62f4f38a3e2)

Appendix C : Using the Zeppelin notebook server

Zeppelin is a notebook server with which several interpreters[38] are available at the installation.

We do not success to communicate with the MapR cluster remotely from a Windows7 installation, but the connection has been established locally or from a remote Unix machine.

Zeppelin is a new application and is strength is its ability to launch spark-submit jobs implemented in Scala, Python or R in a notebook cell. It can also run hive, hbase, spark-sql scripts … and so on… It is also possible to run local Python or R scripts.

Jupyter notebook has analogous capabilities but Zeppelin is easier to configure because its configuration could be entirely managed from the Web UI.
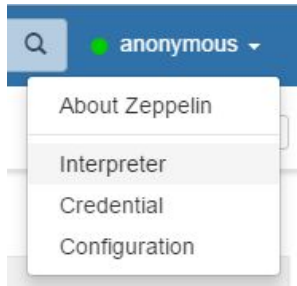
Binary distribution:

http://zeppelin.apache.org/download.html (download and unzip)

List of interpreters available with version 0.6.0:

---

[38] A Zeppelin interpreter is similar to a jupyter kernel

Changes performed with the default configuration:

Change default HTTP listen port	in file: $ZEPPELIN_DIR/conf/zeppelin-site.xml
To be able to use hbase shell on MapR	in file: $ZEPPELIN_DIR/conf/zeppelin-env.sh add: export HADOOP_HOME=/opt/mapr/hadoop/hadoop-2.7.0 export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop export HBASE_HOME=/opt/mapr/hbase/hbase-1.1.1 HBASE_CONF_DIR=/opt/mapr/hbase/hbase-1.1.1/lib/*:/opt/mapr/hbase/hbase-1.1.1/conf
To execute hive commands	Use the %sql interpreter Configure the interpreter: ● Select the 'Interpreter' item in the top-right menu: ● Lookup for the %spark interpreter, edit it and add the following property: ● Save and restart the spark interpreter
Start / Stop Zeppelin	● $ZEPPELIN_DIR/bin/zeppelin-daemon.sh start ● $ZEPPELIN_DIR/bin/zeppelin-daemon.sh start

Appendix D : User commands used to import data

Operation	Tool used	Command
create a MapRDB table (HFile format)	hbase shell	create '<Table_name>', '<cf_1[39]>',... ,'<f_n>
create a MapRDB table (MaprFS format)	hbase shell	create '<maprfs_path>', '<cf_1>',... ,'<cf_n>
import csv file in Hbase	command shell	hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.columns="<target schema>" -Dimporttsv.separator="<sep>" <Table_name|maprfs_path> <tsv_hdfs_path> <target schema> should map ROW_KEY and <cf_i:qualf_j[40]> with the columns of the tsv file **Example : HBASE_ROW_KEY,cf_1:value** the first column of the tsv file is the ROW_KEY the second column is stored in the column family 'cf_1' and the qualifier 'value'
import csv file : generate bulkload	command shell	hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.columns="<target schema>" -Dimporttsv.separator="<sep>" -Dimporttsv.bulk.output=<hdfs_target_bulkload> <Table_name|maprfs_path>
Change format with MaprDB (HFile <==> MapRFS)	command shell	use: hbase org.apache.hadoop.hbase.mapreduce.CopyTable
Map a csv file for Hive	Hue	use: Hive metastore / import
Map an HBase table in Hive	hive	Example for our key/value table CREATE EXTERNAL TABLE <Hive_Table>(key string, value string) ROW FORMAT SERDE 'org.apache.hadoop.hive.hbase.HBaseSerDe' STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler' WITH SERDEPROPERTIES ("hbase.columns.mapping" = ":key,cf1:col1") TBLPROPERTIES ("hbase.table.name" = "<Table_name|maprfs_path>");
Drill configuration	Drill UI (easier way)	http://<hostname>:8047/storage
Start Drill REPL	command shell	$DRILL_PATH/bin/sqlline -u jdbc:drill:schema=<schema>;zk=<zookeeper_quorum>

---

[39] cf_i = column family number i
[40] qualif_j = qualifier number j