2016 IEEE 16th International Conference on Data Mining Workshops

Database Integrated Analytics using R: Initial Experiences with SQL-Server + R

Josep Ll. Berral and Nicolas Poggi
Barcelona Supercomputing Center (BSC)
Universitat Politècnica de Catalunya (BarcelonaTech)
Barcelona, Spain

***Abstract*—Most data scientists use nowadays functional or semi-functional languages like SQL, Scala or R to treat data, obtained directly from databases. Such process requires to fetch data, process it, then store again, and such process tends to be done outside the DB, in often complex data-flows. Recently, database service providers have decided to integrate "R-as-a-Service" in their DB solutions. The analytics engine is called directly from the SQL query tree, and results are returned as part of the same query. Here we show a first taste of such technology by testing the portability of our ALOJA-ML analytics framework, coded in R, to Microsoft SQL-Server 2016, one of the SQL+R solutions released recently. In this work we discuss some data-flow schemes for porting a local DB + analytics engine architecture towards Big Data, focusing specially on the new DB Integrated Analytics approach, and commenting the first experiences in usability and performance obtained from such new services and capabilities.**

I. Introduction

Current data mining methodologies, techniques and algorithms are based in heavy data browsing, slicing and processing. For data scientists, also users of analytics, the capability of defining the data to be retrieved and the operations to be applied over this data in an easy way is essential. This is the reason why functional languages like SQL, Scala or R are so popular in such fields as, although these languages allow high level programming, they free the user from programming the infrastructure for accessing and browsing data.

The usual trend when processing data is to fetch the data from the source or storage (file system or relational database), bring it into a local environment (memory, distributed workers, ...), treat it, and then store back the results. In such schema functional language applications are used to retrieve and slice the data, while imperative language applications are used to process the data and manage the data-flow between systems. In most languages and frameworks, database connection protocols like ODBC or JDBC are available to enhance this data-flow, allowing applications to directly retrieve data from DBs. And although most SQL-based DB services allow user-written procedures and functions, these do not include a high variety of primitive functions or operators.

The arrival of the Big Data favored distributed frameworks like Apache Hadoop and Apache Spark, where the data is distributed "in the *Cloud*" and the data processing can also be distributed where the data is placed, then results are joined and aggregated. Such technologies have the advantage of distributed computing, but when the schema for accessing data and using it is still the same, just that the data distribution is transparent to the user. Still, the user is responsible of adapting any analytics to a *Map-Reduce* schema, and be responsible of the data infrastructure.

Recently, companies like Microsoft, IBM or Cisco, providers of *Analytics as a Service* platforms, put special effort into complementing their solutions by adding scripting mechanisms into their DB engines, allowing to embed analytics mechanisms into the same DB environment. All of them selected R [17] as a language and analytics engine, a free and open-source statistical oriented language and engine, embraced by the data mining community since long time ago. The current paradigm of "Fetch from DB, Process, Dump to DB" is shifted towards an "In-DB Processing" schema, so the operations to be done on the selected data are provided by the same DB procedures catalog. All the computation remains inside the DB service, so the daily user can proceed by simply querying the DB in a SQL style. New R-based procedures, built-in or user created by invoking R scripts and libraries, are executed as regular operations inside the query execution tree.

The idea of such integration is that, not only this processing will be more usable by querying the DB, where the data is managed and distributed, but also will reduce the overhead of the data pipe-line, as everything will remain inside a single data framework. Further, for continuous data processing, analytics procedures and functions can be directly called from triggers when data is continuously introduced or modified.

As a case of use of such approach, in this paper we present some experiences on porting the ALOJA framework [13] to the recently released SQL-Server 2016, incorporating this R-Service functionality. The ALOJA-ML framework is a collection of predictive analytics functions (machine learning and data mining), written in R, originally purposed for modeling and prediction High Performance Computing (HPC) benchmarking workloads as part of the ALOJA Project [16], deployed to be called after retrieving data from a MySQL database. Such project collects traces and profiling of Big Data framework technologies, and analyzing this data requires predictive analytics. These procedures are deployed in R, and communicating the framework with the database and R engine results in a complex architecture, with a fragile data-pipeline and susceptible to failures.

In this work we present how we could adapt our current data-processing approach to a more Big-Data oriented archi-

2375-9259/16 $31.00 © 2016 IEEE
DOI 10.1109/ICDMW.2016.155

tecture, and how we tested using the ALOJA data-set [12], as an example and as a review from the user's point of view.

After testing the Microsoft version of SQL+R services [10], we saw that the major complication, far away of deploying the service, is to build the SQL wrapping procedures for the R scripts to be executed. When reusing or porting already existing code or R applications, the wrapper just has to *source* (R function for "import") the original code and its libraries, and execute the corresponding functions (plus bridging the parameters and the return of this function). Current services are prepared to transform 2 dimensional R Data Frames into tables, as a result of such procedures. Aside of Microsoft services, we took a look into Cisco ParStream [6], displaying a very similar approach, differing on the way of instantiating the R scripts through R file calls instead of direct scripting. It remains for the future work to test and compare the performances among platforms, also to include some experiences with IBM PureData Services [7] and any other new platform providing such services.

This article is structured as follows: Section II presents the current state-of-art and recent approaches used when processing data from databases. Section III explains the current and new data-flow paradigm, and required changes. Section IV shows some code porting examples from our native R scripts to Microsoft SQL Server. Section V provides comments and details on some experiments on running the ALOJA framework in the new architecture. Finally, section VI summarizes this current work and presents the conclusions and future work.

II. State of the Art

Current efforts on systems processing Big Data are mostly focused on building and improving distributed systems. Platforms like Apache Hadoop [1] and Apache Spark [4], with all their "satellite" technologies, are on the rise on Big Data processing environments. Those platforms, although being originally designed towards Java or Python applications, the significant weight of the data mining community using R, Scala or using SQL interfaces, encouraged the platforms strategists over the years to include interfaces and methodologies for using such languages. Revolution Analytics published in 2011 RHadoop [9], a set of packages for R users to launch Hadoop tasks. Such packages included HDFS [14] and HBase [2] handlers, with Map-Reduce and data processing function libraries adapted for Hadoop. This way, R scripts could dispatch parallelizable functions (e.g. R "apply" functions) to be executed in distributed worker computing machines.

The Apache Spark platform, developed by the Berkeley's AMPlab [11] and Databricks [5] and released in 2014, focuses on four main applied data science topics: graph processing, machine learning, data streaming, and relational algebraic queries (SQL). For these, Spark is divided in four big packages: *GraphX*, *MLlib*, *SparkStream*, and *SparkSQL*. SparkSQL provides a library for treating data through SQL syntax or through relational algebraic functions. Also recently, a R scripting interface has been added to the initial Java, Python and Scala interfaces, through the SparkR package, providing the Spark based parallelism functions, Map-Reduce and HBase or Hive [3] handlers. This way, R users can connect to Spark deployments and process data frames in a Map-Reduce manner, the same way they could do with RHadoop or using other languages.

Being able to move the processing towards the database side becomes a challenge, but allows to integrate analytics into the same data management environment, letting the same framework that receives and stores data to process it, in the way it is configured (local, distributed...). For this purpose, companies providing database platforms and services put effort in adding data processing engines as integrated components to their solutions. Microsoft recently acquired Revolution Analytics and its R engine, re-branded as R-Server [8], and connected to the Microsoft SQL-Server 2016 release [10]. IBM also released their platform Pure Data Systems for Analytics [7], providing database services including the *vanilla* R engine from the *Comprehensive R Archive Network* [17]. Also Cisco recently acquired ParStream [6], a streaming database product incorporating user defined functions, programmable as shared object libraries in C++ or as external scripts in R.

Here we describe our first approach to integrate our R-based analytics engine into a SQL+R platform, the Microsoft SQL-Server 2016, primarily looking at the user experience, and discussing bout cases of use where one architecture would be preferred over others.

III. Data-flow Architectures

Here we show three basic schemes of data processing, the local *ad-hoc* schema of pull-process-push data, the new distributed schemes for Hadoop and Spark, and the "In-DataBase" approach using the DB integrated analytics services. Point out that there is not an universal schema that works for each situation, and each one serves better or worse depending on the situation. As an example we put the case of the ALOJA-ML framework as an example of such architectures, how it currently operates, the problematic, and how it would be adapted to new approaches.

A. Local Environments

In architectures where the data and processing capacity is in the same location, having data-sets stored in local file systems or direct access DBs (local or remote databases where the user or the application can access to retrieve or put data). When an analytics application wants to process data, it can access the DB to fetch the required data, store locally and then pass to the analytics engine. Results are collected by the application and pushed again to the DB, if needed. This is a classical schema before having distributed systems with distributed databases, or for systems where the processing is not considered big enough to build a distributed computing-power environment, or when the process to be applied on data cannot be distributed.

For systems where the analytics application is just a user of the data, this schema might be the corresponding one, as the application fetches the data it is granted to view, then

do whatever it wants with it. Also for applications where computation does not require select big amounts of data, as the data required is a small fraction of the total Big Data, it is affordable to fetch the slice of data and process it locally. Also on systems where using libraries like *snowfall* [X], letting the user to tune the parallelism of R functions, can be deployed locally up to a point where this distribution requires heavier mechanisms like Hadoop or Spark.

There are mechanisms and protocols, like ODBC or JDBC (e.g. RODBC library for R), allowing applications to access directly to DBs and fetch data without passing through the file system, but through the application memory space. This is, in case that the analytics application has granted direct access to the DB and understands the returning format. Figure 1 shows both archetypical local schemes.

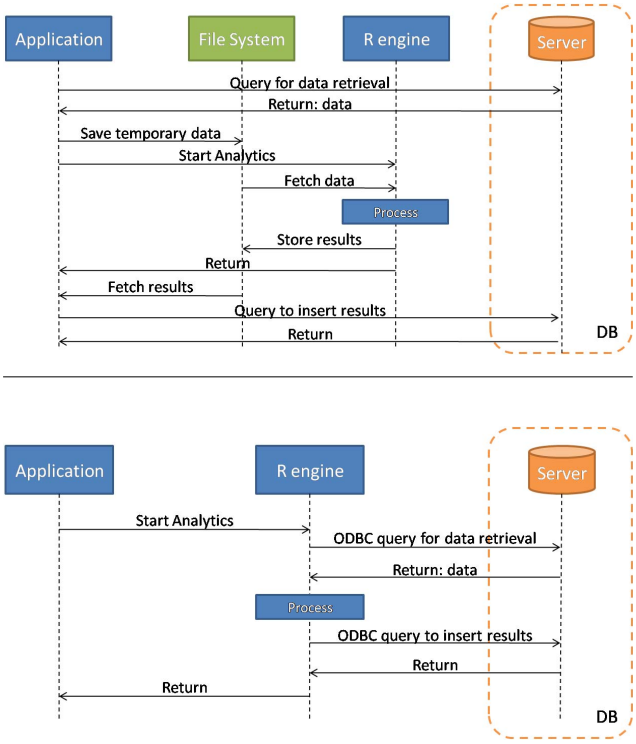

Fig. 1. Schema of local execution approaches, passing data through the File System or ODBC mechanisms

In the first case of Figure 1 we suppose an scenario where the engine has no direct access to DB, as everything that goes into analytics is triggered by the base application. Data is retrieved and pre-processed, then piped to the analytics engine through the file system (files, pipes, ...). This requires coordination between application and engine, in order to communicate the data properly. Such scenarios can happen when the DB, the application and the engine do not belong to the same organization, or when they belong to services from different providers. Also it can happen when security issues arise, as the analytics can be provided by a not-so-trusted provider, and data must be anonymized before exiting the DB.

Also, if the DB and analytics engine are provided by the same service provider, there will be means to communicate the storage with the engine in an ODBC or other way, allowing to fetch the data, process, and then return the results to the DB. As an example, the Microsoft Azure-ML services [15] allow the connection of its machine learning components with the Azure storage service, also the inclusion of R code as part of user-defined components. In such service, the machine learning operations do not happen on the storage service, but data is pulled into the engine and then processed.

At this moment, the ALOJA Project, consists of a web user interface (front-end and back-end), a MySQL database and the R engine for analytics, used this schema of data pipe-line. The web interface provided the user the requested data from the ALOJA database, and then displayed in the front-end. If analytics were required, the back-end dumped the data to be treated into the file system, and then started the R engine. Results were collected by the back-end and displayed (also stored in cache). As most of the information passed through user filters, this data pipe-lining was preferred over the direct connection of the scripts with the DB. Also this maintained the engine independent of the DB queries in constant development at the main framework.

Next steps for the project are planned towards incorporating a version of the required analytics into the DB, so the back-end of the platform can query any of the provided analytics, coded as generic for any kind of input data, as a single SQL query.

B. Distributed Environments

An alternative architecture for the project would be to upload the data into a Distributed Database (DDB), thinking on expanding the ALOJA database towards Big Data (we still have Terabytes of data to be unpacked into the DB and to be analyzed at this time). The storage of the data could be done using Hive or HBase technologies, also the analytics could be adapted towards Hadoop or Spark. For Hadoop, the package RHadoop could handle the analytics, while the SparkSQL + SparkR packages could do the same for Spark. Processing the analytics could be done on a distributed system with worker nodes, as most of the analytics in our platform can be parallelized. Further, Hadoop and Spark have machine learning libraries (Mahout and MLlib) that could be used as native instead of some functionalities of our ALOJA-ML framework. Figure 2 shows the execution schema of this approach.

Such approach would concentrate all the data retrieval and processing in a distributed system, not necessarily near the application but providing parallelism and agility on data browsing, slicing and processing. However, this requires a complex set-up for all the involved services and machines, and the adaption of the analytics towards parallelism in a different level than when using *snowfall* or other packages. SparkR provides a Distributed Data-Frame structure, with similar properties than regular R data-frames, but due to the distributed property, some classic operations are not available

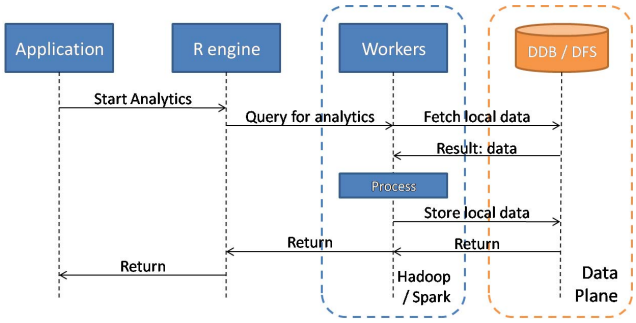

Fig. 2. Schema of distributed execution approach, with Distributed Databases or File Systems

or must be performed in a different way (e.g. column binding, aggregates like "mean", "count"...).

This option would be chosen at that point where data is large enough to be dealt with a single system, considering that DB managers do not provide means to distribute and retrieve data. Also, like in the previous approach, the data pipe-line passes through calling the analytics engine to produce the queries and invoke (in a Map-Reduce manner) the analytics when parallelizable, or collect locally data to apply the non-parallelizable analytics. The price to be paid would be the set-up of the platform, and the proper adjustment of the analytics towards a Map-Reduce schema.

C. Integrated Analytics

The second alternative to the local analytics schema is to incorporate those functions to the DB services. At this point, the DB can offer *Analytics as a Service*, as users can query for analytics directly to the DB in a SQL manner, and data will be directly retrieved, processed and stored by the same framework, avoiding the user to plan a full data pipe-line.

As the presented services and products seem to admit R scripts and programs directly, no adaption of the R code is required. The effort must be put in coding the wrapping procedures for being called from a SQL interface. Figure 3 shows the basic data-flow for this option.

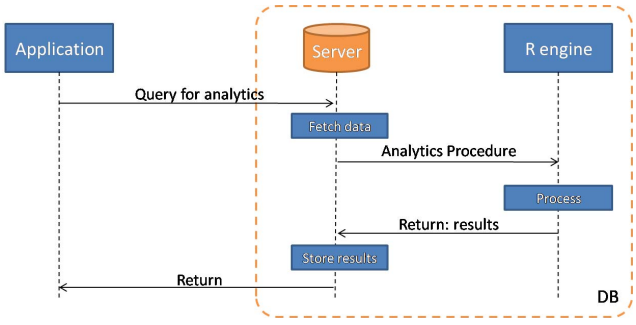

Fig. 3. Schema of "In-DB" execution approach

The advantages of such approach are that 1) analytics, data and data management are integrated in the same software package, and the only deployment and configuration is of the database; 2) the management of distributed data in Big Data deployments will provided by the same DB software (whether this is implemented as part of the service!), and the user doesn't need to implement Map-Reduce functions but data can be aggregated in the same SQL query; 3) simple DB users can invoke the analytics though a SQL query, also analytics can be programmed generically to be applied with any required input SQL query result; 4) DBs providing *triggers* can produce *Continuous Analytic Queries* each time data is introduced or modified.

This option still has issues to be managed, such as the capacity of optimization and parallelism of the embedded scripts. In the case of the Microsoft R Service, any R code can be inserted inside a procedure, without any apparent optimization to be applied as any script is directly sent to a external R sub-service managed by the principal DB service. Such R sub-service promises to apply multi-threading in Enterprise editions, as the classic R engine is single-thread, and multi-threading could be applied to vectorization functions like "apply", without having to load the previously mentioned *snowfall* (loadable as the script runs on a compatible R engine). Also, comparing this approach to the distributed computing system, if the service supports "partitioning" of data according to determined columns and values, SQL queries could be distributed in a cluster of machines (not as replication, but as distribution), and aggregated after that.

IV. Adapting and Embedding Code

According to the SQL-Server published documentation and API, the principal way to introduce external user-defined scripts is to wrap the R code inside a Procedure. The SQL-Server procedures admit the script, that like any "Rscript" can source external R files and load libraries (previously copied into the corresponding R library path set up by the Microsoft R Service), also admit the parameters to be bridged towards the script, and also admit SQL queries to be executed previous to start the script for filling input data-frames. The procedure also defines the return values, in the form of data-frames as tables, values or a tuple of values.

When the wrapping procedure is called, input SQL queries are executed and passed as input data-frame parameters, direct parameters are also passed to the script, then the script is executed in the R engine as it would do a "Rscript" or a script dumped into the R command line interface. The variables mapped as outputs are returned from the procedure into the invoking SQL query.

Figure 4 shows an example of calling ALOJA-ML functions in charge of learning a linear model from the ALOJA data-set, read from a file while indicating the input and output variables; also calling a function to predict a data-set using the previously created model. The training process creates a model and its hash ID, then the prediction process applies the model to all the testing data-set. In the current ALOJA data pipe-line, "ds" should be retrieved from the DB (to a file to be read, or directly to a variable if RODBC is used). Then

```
source("functions.r");
library(digest);

## Training Process
model <- aloja_linreg(ds = read.table("aloja6.csv"), vin = c("maps","iofilebuf"), vout = "exe_time");
id_hash <- digest(x = model, algo = "md5"); # ID for storing the model in DB

## Prediction Example
predictions <- aloja_predict_dataset(learned_model = model, ds = read.table("aloja6test.csv"));
```

Fig. 4. Example of modeling and prediction example using the ALOJA-ML libraries. Loading ALOJA-ML functions allow the code to execute "aloja_linreg" to model a linear regression, also "aloja_predict_dataset" to process a new data-set using a previously trained model.

the results ("model", "id_hash" and "predictions") should be reintroduced to the DB using again RODBC, or writing the results into a file and the model into a serialized R object file.

Figures 5 and 6 show how this R code is wrapped as procedure, and examples of how these procedures are invoked. Using this schema, in the modeling procedure, "ds" would be passed to the procedure as a SQL query, "vin" and "vout" would be bridged parameters, and the "model" and "id_hash" would be returned as a two values (a serialized blob/string and a string) that can be saved into a models table. The return would be introduced into the models table. Also the prediction procedure would admit an "id_hash" for retrieving the model (using a SQL query inside the procedure), and would return a data-frame/table with the row IDs and the predictions.

We observed that, when preparing the embedded script, all sources, libraries and file paths must be prepared like a Rscript to be executed from a command line. The environment for the script must be set-up, as it will be executed each time in a new R session.

In the usual work-flow on ALOJA-ML tools, models (R serialized objects) are stored in the file system, and then uploaded in the database as binary blobs. Working directly from the DB server allow to directly encode serialized objects into available DB formats. Although SQL-Server includes a "large binary" data format, we found some problems when returning binary information from procedures (syntax not allowing the return of such data type in tuples), thus serialized objects can be converted to text-formats like *base64* to be stored as a "variable size character array".

V. Experiments and Experiences

A. Usability and Performance

For testing the approach we used the SQL-Server 2016 Basic version, with the R Service installed and with Windows Server 2012, from a default image available at the Azure repositories. Being the basic service, and not the enterprise, we assumed that R Services would not provide improvements on multi-threading, so additional packages for such functions are installed (*snowfall*). The data from the ALOJA data-set is imported through the CSV importing tools in the SQL-Server Manager framework, the "Visual-Studio"-like integrated development environment (IDE) for managing the services.

Data importing displayed some problems, concerning to data-type transformation issues, as some *numeric* columns couldn't be properly imported due to precision and very big values, and had to be treated as *varchar* (thus, not treatable as number). After making sure that the CSV data is properly imported, the IDE allowed to perform simple queries (SELECT, INSERT, UPDATE). At this point, tables for storing pair-value entries, containing the models with their specific ID_hash as key, is created. Procedures wrapping the basic available functions of ALOJA-ML are created, as specified in previous section IV. After executing some example calls, the system works as expected.

During the process of creating the wrapping procedures we found some issues, probably non-reported bugs or system internal limitations, like the fact that a procedure can return a *large binary* type (large *blob* in MySQL and similar solutions), also can return tuples of diverse kinds of data types, but it crashed with an internal error when trying to return a tuple of a *varchar* and a *large binary*. Workarounds were found by converting the serialized model object (*binary* type) into a *base64* encoding string (*varchar* type), to be stored with its ID_hash key. As none information about this issue was found in the documentation, at the day of registering these experiences, we expect that such issues will be solved by the development team in the future.

We initially did some tests using the modeling and prediction ALOJA-ML functions over the data, and comparing times with a local "vanilla" R setup, performance is almost identical. This indicating that with this "basic" version, the R Server (former R from Revolution Analytics) is still the same at this point.

Another test done was to run the *outliers classification* function of our framework. The function, explained in detail in its corresponding work [13], compares the original output variables with their predictions, and if the difference is k times greater than the expected standard deviation plus modeling error, and it doesn't have enough support from similar values on the rest of the data-set, such data is considered an outlier. This implies the constant reading of the data table for prediction and for comparisons between entries. The performance results were similar to the ones on a local execution environment, measuring only the time spent in the function. In a HDI-A2 instance (2 virtual core, only 1 used, 3.5GB memory),

```
%% Creation of the Training Procedure, wrapping the R call
CREATE PROCEDURE dbo.MLPredictTrain @inquery nvarchar(max), @varin nvarchar(max),
                                    @varout nvarchar(max) AS
BEGIN
EXECUTE sp_execute_external_script
@language = N'R',
@script = N'
    source("functions.r");
    library(digest); library(base64enc);
    model <- aloja_linreg(ds = InputDataSet, vin = unlist(strsplit(vin,",")), vout = vout);
    serial <- as.raw(serialize(model, NULL));
    OutputDataSet <- data.frame(model = base64encode(serial),
                                id_hash = digest(serial, algo = "md5"));
',
@input_data_1 = @inquery,
@input_data_1_name = N'InputDataSet',
@output_data_1_name = N'OutputDataSet',
@params = N'@vin nvarchar(max), @vout nvarchar(max)',
@vin = @varin,
@vout = @varout
WITH RESULT SETS (("model" nvarchar(max), "id_hash" nvarchar(50)));
END

%% Example of creating a model and storing into the DB
INSERT INTO aloja.dbo.trained_models (model, id_hash)
EXEC dbo.MLPredictTrain @inquery = "SELECT exe_time, maps, iofilebuf FROM aloja.dbo.aloja6",
                        @varin = "maps,iofilebuf", @varout = "exe_time"
```

Fig. 5. Version of the modeling call for ALOJA-ML functions in a SQL-Server procedure. The procedure generates the data-set for "aloja_linreg" from a parametrized query, also bridges the rest of parameters into the script. It also indicates the format of the output, being a value, a tuple or a table (data frame).

```
%% Creation of the Predicting Procedure, wrapping the R call
CREATE PROCEDURE dbo.MLPredict @inquery nvarchar(max), @id_hash nvarchar(max) AS
BEGIN
DECLARE @modelt nvarchar(max) = (SELECT TOP 1 model FROM aloja.dbo.trained_models
                                 WHERE id_hash = @id_hash);
EXECUTE sp_execute_external_script
@language = N'R',
@script = N'
    source("functions.r");
    library(base64enc);
    results <- aloja_predict_dataset(learned_model = unserialize(as.raw(base64decode(model))),
                                     ds = InputDataSet);
    OutputDataSet <- data.frame(results);
',
@input_data_1 = @inquery,
@input_data_1_name = N'InputDataSet',
@output_data_1_name = N'OutputDataSet',
@params = N'@model nvarchar(max)',
@model = @modelt;
END

%% Example of predicting a dataset from a SQL query with a previously trained model in DB
EXEC aloja.dbo.MLPredict @inquery = 'SELECT exe_time, maps, iofilebuf FROM aloja.dbo.aloja6test',
                         @id_hash = 'aa0279e9d32a2858ade992ab1de8f82e';
```

Fig. 6. Version of the Prediction call for ALOJA-ML functions in a SQL-Server procedure. Like the training procedure in figure 5, the procedure primarily retrieves the data to be processed from a SQL query, and passes it with the rest of parameters into the script. Here the result is directly a table (data frame).

it took 1h:9m:56s to process the 33147 rows, selecting just 3 features. Then, as a way to improve the performance, due to the limitations of the single-thread R Server, we loaded *snowfall*, and invoked it from the ALOJA-ML "outlier_dataset" function, on a HDI-A8 instance (8 virtual core, all used, 14GB memory). The data-set was processed in 11m:4s, barely 1/7

of the previous time, considering the overhead of sharing data among R processes created by *snowfall*, demonstrating that despite not being a multi-threaded set-up, using the traditional resources available on R it is possible to scale R procedures.

B. Discussion

One of the concerns on the usage of such service is, despite and because of the capability of multi-processing using built-in or loaded libraries, the management of the pool of R processes. R is not just a scripting language to be embedded on a procedure, but it is a high-level language that allows from creating system calls to parallelizing work among networked working nodes. Given the complexity that a R user created function can achieve, in those cases that such procedure is heavily requested, the R server should be able to be detached from the SQL-server and able in dedicated HPC deployments.

The same way *snowfall* can be deployed for multi-threading (also for cluster-computing), clever hacks can be created by loading *RHadoop* or *SparkR* inside a procedure, connecting the script with a distributed processing system. As the SQL-server bridges tables and query results as R data frames, such data frames can be converted to Hadoop's Resilient Distributed Data-sets or Spark's Distributed Data Frames, uploaded to a HDFS, processed, then returned to the database. This could bring to a new architecture of SQL-Server (or equivalent solutions) to connect to distributed processing environments, as slave HPC workers for the database. Also an improvement could be that the same DB-server, instead of producing input table/data frames already returned Distributed Data Frames, being the data base distributed into working nodes (in a partitioning way, not a replication way). All in all, the fact that the embedded R code is directly passed to a nearly-independent R engine allows to do whatever a data scientist can do with a typical R session.

VI. Summary and Conclusions

The incorporation of R, the semi-functional programming statistical language, as an embedded analytics service into databases will suppose an improvement on the ease and usability of analytics over any kind of data, from regular to big amounts (Big Data). The capability of data scientists to introduce their analytics functions as a procedure in databases, avoiding complex data-flows from the DB to analytics engines, allow users and experts a quick tool for treating data in-situ and continuously.

This study discussed some different architectures for data processing, involving fetching data from DBs, distributing data and processing power, and embedding the data process into the DB. All of this using the ALOJA-ML framework as reference, a framework written in R dedicated to model, predict and classify data from Hadoop executions, stored as the ALOJA data-set. The shown examples and cases of use correspond to the port of the current ALOJA architecture towards SQL-Server 2016, integrating R Services.

After testing the analytics functions after the porting into an SQL database, we observed that the major effort for this porting are in the wrapping SQL structures for incorporating the R calls into the DB, without modifying the original R code. As performance results similar than R standalone distributions, the advantages come from the input data retrieving and storing.

In future work we plan to test the system more in-depth, and also compare different SQL+R solutions, as companies offering DB products have started putting efforts into integrating R engines in their DB platforms. As this study focused more in an initial hands-on with this new technology, future studies will focus more on comparing performance, also against other architectures for processing Big Data.

Acknowledgments

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 639595).

References

[1] Apache Hadoop. http://hadoop.apache.org (Aug 2016).
[2] Apache HBase. https://hbase.apache.org/ (Aug 2016).
[3] Apache Hive. https://hive.apache.org/ (Aug 2016).
[4] Apache Spark. https://spark.apache.org/ (Aug 2016).
[5] Databricks inc. https://databricks.com/ (Aug 2016).
[6] ParStream. Cisco corporation. http://www.cisco.com/c/en/us/products/analytics-automation-software/parstream/index.html (Aug 2016).
[7] PureData Systems for Analytics. IBM corporation. https://www-01.ibm.com/software/data/puredata/analytics/ (Aug 2016).
[8] R-Server. Microsoft corporation. https://www.microsoft.com/en-us/cloud-platform/r-server (Aug 2016).
[9] RHadoop. Revolution Analytics. https://github.com/RevolutionAnalytics/RHadoop/wiki (Aug 2016).
[10] SQL-Server 2016. Microsoft corporation. https://www.microsoft.com/en-us/cloud-platform/sql-server (Aug 2016).
[11] UC Berkeley, AMPlab. https://amplab.cs.berkeley.edu/ (Aug 2016).
[12] Barcelona Supercomputing Center. ALOJA home page. http://aloja.bsc.es/ (Aug 2016).
[13] J. L. Berral, N. Poggi, D. Carrera, A. Call, R. Reinauer, and D. Green. ALOJA-ML: A framework for automating characterization and knowledge discovery in hadoop deployments. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015*, pages 1701–1710, 2015.
[14] D. Borthakur. *The Hadoop Distributed File System: Architecture and Design. http://hadoop.apache.org/docs/r0.18.0/hdfs_design.pdf*. The Apache Software Foundation, 2007.
[15] Microsoft Corporation. Azure 4 Research. http://research.microsoft.com/en-us/projects/azure/default.aspx (Jan 2016).
[16] N. Poggi, J. L. Berral, D. Carrera, A. Call, F. Gagliardi, R. Reinauer, N. Vujic, D. Green, and J. A. Blakeley. From performance profiling to predictive analytics while evaluating hadoop cost-efficiency in ALOJA. In *2015 IEEE International Conference on Big Data, Big Data 2015, Santa Clara, CA, USA, October 29 - November 1, 2015*, pages 1220–1229, 2015.
[17] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2014.