Programming Languages in Data Science: a Comparison from a Database Angle

Carlos Ordonez, Xiantian Zhou
University of Houston, USA

***Abstract*—In a typical Data Science project the analyst uses many programming languages to explore and build mathematical models on big data. Data sets come from diverse data sources including databases, logs, text and even images. A major challenge is managing and pre-processing so much data, with potentially inconsistent content, significant redundancy, in diverse format, with varying data quality. Database systems research has tackled such problems for a long time, but mostly on relational databases. With such motivation in mind, this paper compares strengths and weaknesses of popular languages used nowadays from a database system angle: Python, R and SQL. We focus on performance and functionality. We discuss the entire analytic pipeline, going from data integration, cleaning and pre-processing to model application and tuning. From a general perspective, we present a comprehensive survey of storage mechanisms, data processing algorithms, external algorithms, runtime memory management and essential optimizations. From a database systems angle, we consider programming abstraction, flexibility, processing speed, main memory limitations, consistency, and parallel processing.**

I. Introduction

The big data analytics revolution was born with data mining [2], which optimized machine learning statistics and pattern detection algorithms to analyze large data sets. Data mining research focus on scalability implied efficient processing beyond RAM limits, leaving parallel processing as a secondary requirement. Around the same time data warehousing [2] was born to analyze databases with demanding queries in so-called On-Line Analytical Processing [1]. During the last decade Data Warehouses evolved and went through a disruptive transformation to become Big Data Lakes, where data exploded in the three Vs: Volume, Velocity and Variety. The three Vs brought new challenges to data mining systems. So data mining and queries on Big Data lakes morphed into a catch-all term: Big Data Analytics [3]. This evolution brought faster approaches, tools and algorithms to load data, querying beyond SQL considering semantics, mixing text with tables, stream processing and computing machine learning models (broadly called AI). A more recent revolution brought two more Vs: Veracity in the presence of contradictory information and even Value, given so many development and tool options and the investment to exploit big data. Nevertheless, having so much information in a central repository enabled more sophisticated exploratory analysis, beyond multivariate statistics and queries. Over time people realized that managing so much diverse data required not only database technology, but also a more principled approach laying its foundation on one hand in mathematics (probability, machine learning, numerical optimization, statistics) and on the other hand, more abstract, highly analytic, programming (combining multiple languages, pre-processing data, integrating diverse data sources), giving birth to Data Science (DS). This new trend is not another fad: data science is now considered a competing discipline to computer science and even applied mathematics.

We survey programming languages widely used in Data Science: Python, R, and SQL. We contrast storage, data manipulation functions, processing and main analytical tasks across diverse systems including their intractive runtimes, the Hadoop [5] stack and parallel DBMSs [1]. We discuss their strengths and weaknesses, making clear there is no single winner and therefore they will have to coexist in the foreseable future.

II. System Infrastructure

A. Programming Language

The importance of the specific programming language used for analytic development cannot be overstated: it is the interface for the analyst, it provides different levels of abstraction, it offers different data manipulation mechanisms and it works best with some specific analytic task. There are two main families: interpreted and compiled. DS languages are interpreted, but the systems evaluating them have compiled languages behind, mostly C++ (and C), but also Java (Hadoop stack). Given the need to provide high performance the DS language offers extensibility mechanisms that allow plugging in functions and operators programmed in the compiled language, but which require expertise on the internals of the runtime system. The average analyst does not mess with the compiled language, but systems researchers and developers do. Finally, SQL is the established language to process queries in a database and to compute popular machine learning models and data mining techniques. In a similar manner to DS languages, it is feasible to extend the language with advanced anlytic capabilities with UDFs programmed in C++/C and a host language (Java, Python).

B. Data Set Storage Mechanisms

In this section identify the main storage mechanisms used today in most programming languages: arrays, data frames and tables. Notice each programming language offers such mechanisms in different flavors with varying features and limitations, but most of them are 2-dimensional. We should emphasize tensors (high dimensional numeric arrays) can be ”sliced” into matrices.

An array is a data structure provided by a programming language and it is arrangement of elements of the same data type. An array can represent a table as an array of records and a matrix as a 2-dimensional array of numbers. In most programming languages, like C++ or Java, the array is a data structure in main memory, although there exist systems (SciDB [7]) which allow manipulating arrays on disk. The data frame is another data structure in main memory, somewhat similar to an array. Data frame was introduced in the R language, allowing to assemble rows or columns of diverse data types, which has made its way into Python in the Pandas library. It has similarities to a relational table, but it is not a relational table. When the data type of all entries is a number data frames resemble a matrix, but they are not a matrix either. However, it easy to transform a data frame with numbers into a matrix and vice-versa. An SQL table is a set of rows (aka tuples), which is internally stored stored as arrays or lists in main memory or as blocks (small arrays) on disk. By adding a primary key (PK) constraint a flat table becomes a relational table [1]. An SQL table is a data structure on secondary storage (disk, solid state), allowing manipulation with SQL, without worrying abut row by row or block by block access, in contrast with DS languages like Python. A record is generally understood as physical storage of a tuple, followimg a specific column order and each value taking fixed space. Thus, by design and architecture, a relational table is significantly different from a data frame. Data frames and arrays are somewhat similar, but data frames are more general: they can track row and column names and they do not require contiguous storage in main memory.

C. File Access Mechanisms

In this section we discuss the main access mechanisms, depending on data set storage.

All systems can process text and binary files, but tend to favor one. DS languages generally work on text files, which can be easily transferred and exchanged with other analytic tools (spreadsheets, editors, word processors). DBMSs use binary files in specific formats, doing a format conversion when records are inserted by transactions or when they are loaded in batch from files. Hadoop system use a combination of both, ranging from plain files to load data to transforming data into efficient block formats as needed in a subsystem.

The I/O unit is an important performance consideration to process a large data set, which varies depending on the system: line by line or an entire file for a data frame. one line at a time for text files, one record or block of records for tables. DS systems and libraries vary widely on how they read, load and process data sets. Arrays vary according to their content; in 2-dimensional format of numbers they are read in one pass, as one block for a small array or multiple blocks for a large array. Large arrays are generally manipulated with block-based access on binary files, where each block is an array. Blocks for dense matrices are generally squared. Few systems attempt to provide subscript-based access on disk because it requires adding special functions and constructs in the programming language.

Data frames are generally loaded with one scan on the input text file, but there exist libraries that allow block by block access (called chunk to distinguish them from database blocks). Therefore, the access is sequential.

Each programming language and system incorporate different access operators, but here we provide a broad classification. In a DS language these are the most common operators in main memory: scan (iterator), sort, merge. Iterators come from object-oriented programming. The merge operator is similar to a relational join, but more flexible to manipulate dirty data. DBMSs feature database operators combining processing in main memory and disk: scan, sort, join; filters are processed with a full scan when there are no indexes, but they can be accelerated with indexes when possible. Sorting can be used at many stages to accelerate further processing.

III. Processing in Data Science: A Database Systems Perspective

A. A Flexible and Comprehensive Processing Architecture

It is difficult to synthesize all different DS pipelines into a single system architecture. We use a database architecture as a foundaion, generalizing it to diverse data sources and relaxing its strict structure and consistency requirements. Here we list the most common elements: diverse data sources, a data repository, an analytic computer or cluster. The tasks: loading, cleaning, integrating, analyzing, deploying. Data sources include: databases, logs, documents, perhaps with inconsistent and dirty content. The data repository can be a folder in a shared server or a data lake in parallel cluster. Cleaning and integrating big data is generally done combining Hadoop and DBMSs: it just depends where the bulk of information is. Some years ago the standard was to process big data in a local parallel cluster. Cloud computing [8] has changed the landscape, having data lakes in the cloud, which enables analysts to integrate and clean data in the cloud. In general a data set for analysis is much smaller than raw big data, which allow analysts to process data locally, in their own workstation or server.

B. Integrating Data Sources

Data science languages (Python, R) provide significant freedom to integrate data sets as needed. They have libraries to merge data sets by a common columns(s), behaving in a similar manner to a relational join. Python is more flexible than R to manipulate text (words, sentences). R provides better defined operators and functions to manipulate matrices [4]. In the absence of columns with overlapping content, data integration is difficult. In a Hadoop system there exist text manipulation libraries that can match records by content, which is especially challenging with strings. The best principled approaches come from the database world, where records can be matched by columns or by content, but they are generally restricted to tabular data (i.e. relational tables).

C. Input Data File Format

In general the input file is in either text or binary format, where the most common input file format are text files, with CSV format being a de facto standard. The CSV format allows representing matrices, relational tables and data frames with missing values and strings of varying length. Binary formats are more common in HPC and specific science systems (physics, chemistry, and so on)i and in the case of DBMSs they are proprietary. Streams come in the form of ever growing log files (commonly CSV), where each log record has a timestamp and records that can vary in structure and length. JSON, a new trend, is a more structured text format to import data, which can represent diverse objects, including documents and database records.

D. Copying Files vs Loading Data into Tables

In most DS languages there is no specific loading phase: any file can be analyzed directly by reading it and loading in the storage mechanisms introduce above in main memory. Therefore, copying plain files (e.g CSV files) to the DS server or workstation is all that is required. On the other hand, this phase represents a bottleneck in a database system and a more reasonable compromise in Hadoop systems. In a Hadoop system it can range from copying text files to formatting records in a specific storage format (key-value, Hive, Parquet). In a database system input records are generally encoded into a specific binary format, which allows efficient blocked random access and record indexing. A key aspect in a parallel system, is partitioning the input data set and distributing it, explained below.

E. Serial vs Parallel Processing

This is the norm in data science languages. It is easier to develop small programs without worrying about low-level details such as multicore CPUs, threads and network communication. In general, each analyst runs in their own space, without worrying about shared memory. In a large project, data integration and preparation is assumed to have been take care of before.

Parallel processing is the main strength of Big Data Hadoop systems and parallel DBMSs, which complement each other [6]. In most systems parallel processign requires three phases: (1) partitioniong data records and distributing them into chunks/blocks across nodes; (2) running processing code on each data partition; (3) gathering or assembling partial results for further computation or getting final results. Phase 1 may require sorting, automatically or manually and hashing records by a key. The main requirement is to process each partition independently, minimizing node or process coordination (an overhead). Given hardware evolution and cloud computing, there is a dilemma between going for scaleup (more cores, more RAM) or scale out (more machines, less RAM on each). In DS languages there is automatic parallelism in the object code which is optimized for modern multicore CPUs. Multicore CPUs, larger RAM and SSDs favor doing analysis in one beefy machine. On the other hand, big data, especially "Volume" support distributed processing to overcome the I/O bottleneck. Streams, being sequential data, lead to distributed filtering and summarization, but commonly centralized processing of complex models. Hadoop and DBMSs offer a parallel version of scan, sort and merge/join. However, Hadoop systems trail DBMSs on indexing and advanced join processing, but the gap keeps shrinking. When there are indexes, either in main memory or secondary storage filtering, joining and aggregations can be accelerated.

F. Enforcing Data Integrity and Consistency

In Data Science languages, like Python and R, there is a vague notion of ACID properties, which tends to be ignored, especially with text data (documents, web pages). When there are significant data additions or changes, the data preparation pipeline is re-executed and the analytical tasks repeated with a refreshed data set. Hadoop systems have been gradually strengthened with more ACID properties, especially to explore data sets interactively. It is noteworthy ACID properties have gradually subsumed eventual consistency. DBMSs provide the strongest ACID guarantees, but they are generally considered a second alternative after Hadoop big data systems to analyze large volumes of data with machine learning or graph algorithms. DBMSs forte is query processing. In general, most DBMSs propagate changes on queries recomputing materialized views. As there is more interest in getting near real-time (active) results, but maintaining consistency, the underlying tables are locked or maintained with MVCC.

G. Fault Tolerance

There are two main scenarios: fault tolerance to avoid data loss and fault tolerance during processing. This is a contrasting feature with data science languages, which do not provide fault tolerance either to avoid data loss or to avoid redoing work when there are runtime errors. In many cases, this is not seen as a major limitation because analytic data sets can be recreated from source data and because code bugs can be fixed. But they represent a time loss. On the other hand, Hadoop systems provide run-time fault tolerance during processing when one node or machine fail, with automatic data copy/recovery from a backup node or disk. This feature is at the heart of the Google file system (proprietary) and HDFS (open source). Parallel DBMSs provide fault tolerance to avoid losing data, going from secondary copies of each data block to the internal log, where all update operations are recorded, It is generally considered that continuously appending the recovery log is required for transactions, but an overhead for analytics.

IV. Analytics

A. Mathematical Objects

We introduce definitions for the main mathematical objects used in data science. We use matrix, graph and relational table as the main and most general mathematical objects. Graphs can represent cubes and itemsets as a particular case.

The most important one is the matrix, which may be square or rectangular. A matrix is composed of a list of vectors.

A tuple is an n-ary list of values of possibly diverse types, accessed by attribute name. A specific tuple in a relational table is accessed by a key. Finally, bags and sets contain any kind of object without any pre-defined structure, where elements are unioque in sets and there may be repetitions in bags.

It is important to emphasize these mathematical objects are different. A matrix is different from a relational table. In a matrix elements are accessed by subscript, whereas in a relational table by key and attribute name. In an analagous manner, a vector is different from a tuple. Relational tables are sets of uniform objects (tuples), but sets in general are not. Then bags represent generic containers for anything, including words, files, iamges and so on. We will argue different systems tend to be better for one kind of object, requring significant effort the other two.

A graph $G = (V, E)$ consists of a set of vertices V and a set of edges E connecting them. Graph size is given by $n = |V|$ and $m = |E|$, where $m = O(n)$ for sparse graphs and $m = O(n^2)$ for dense graphs, with a close correspondence to sparse and dense matrices. A graph can be manipulated as a matrix or as a list of edges (or adjacent vertices).

B. Data Exploration

A data set is commonly explored with descriptive statistics and histograms. These statistical mechanisms and techniques give an idea about data distribution. When a hypothesis comes up it is common to run some statistical test whose goal is to reject or accept a user-defined hypothesis. In general, these analyses are fast and they scale reasonably well on large data sets in a data science language, especially when the data sets fits in RAM. Together with statistics Data Science systems provide visualization aids with plots, mesh grids and histograms, both in 2D and 3D. When interacting with a DBMS, analysts explore the data set with queries, or tool that generate queries. In general queries are written in SQL and they combine filters, joins and aggregations. In cube processing and exploration, one query leads to another more focused query. However, the analyst commonly exports slightly pre-processed data sets outside the DBMS due to the ease of use of DS languages (leaving performance as a secondary aspect). Lately, data science languages provide almost equivalent routines to explore the data set with equivalent mechanisms to queries. Out of many systems out there, the Pandas library represents a primitve, but powerful, query mechanism whose flexibility is better than SQL, but lacking important features of query processing.

C. Graph Analytics

The most complex exploration mechanism is graphs, which are flexible to represent any set of interconnected objects. Nowadays, they are particularly useful to represent social networks and interconnected computers and devices on the Internet. Their generality allows answering many exploratory questions, which are practically impossible to get with descriptive statistics. Even though graphs represent a mathematical model they can be considered descriptive models rather than predictive. Nevertheless, many algorithms on graphs are computationally challenging, with many of them involving NP-complete problems or exponential time and space complexity. Well-known problems include paths, reachability, centrality, diameter and clique detection, most of which remain open with big data. Graph engines (Spark GraphX, Neo4j) lead, followed by parallel DBMSs (Vertica, Teradata, Tigergraph) to analyze large graphs. DS language libraries do not scale well with large graphs, especially when they do fit in main memory.

D. Computing Machine Learning Models

Machine learning and statistic are the most prominent mathematical models. We broadly classify models as descriptive and predictive. Descriptive (unsupervised) models are a generalization of descriptive statistics, like the mean and variance, in one dimension. For predictive (supervised) models there is an output variable, which makes the learning problem significantly harder. If the variable is continueous we commonly refer to it as Y, whereas if it a discrete variable we call it G. Nowadays machine learning has subsumed statistics to build predictive models This is in good part due to deep neural networks, which can work on unlabeled data, on text and on images. Moreover, deep neural networks have the capability of automatically deriving features (variables), simplifying data pre-processing. Generally speaking these computation involve iterative algorithms involving matrix computations. The time complexity goes from $O(dn)$ to $O(2^d n)$ and $O(dn^2)$ in practice. But it can go up to $O(2^d n)$ for variable/feature selection or Bayesian variable networks. The number of iterations is generally considered an orthogonal aspect to $O()$, being a challenge for clustering algorithms like K-means and EM. Data science language libraries excel in the variety of models they and the ability to stack them. Models range from simple predictive models like NB and PCA to deep neural networks. It is fair to say Python dominates the landscape in neural networks (Tensorflow, Keras Scikit-learn) and R in advanced statistical models. Hadoop systems offer specialized libraries to compute models like Spark MLlib and Mahout. The trend has been to build wrappers in a DS language, where Python is now the dominant DS language Spark, leaving Scala for expert users. DBMSs offer some algorithms programmed via SQL queries and UDFs, fast cursor interfaces, or internal conversion from relational to matrix format, but they are difficult to use, and they cannot be easily combined and they require importing external data.

Given DBMS rigid archicture, and learning curve, analysts prefer DS languages over SQL, followed by Hadoop with data volume forces the analyst to use such tool. In other words, DBMSs have lost a lot of ground as an analytical platform, being used today mainly for decision support queries and in isolated cases to compute machine learning models. In practice, most models can be computed on samples with acceptable accuracy or they can be computed on pre-processed data sets where d and n have been significantly reduced,

compared to raw (redudant) data. Building ML models on big data remains an open problem, but it is a moving target, gradually going deeper into DS languages like Python.

V. Current Trends and Research Issues

In Data Science Python and R are now the established programming languages. Python and R are not the most efficient programming languages, nor the ones that guarantee data consistency and integrity, but they are flexible, easy to learn and their libraries are getting bigger and faster. Among both languages, Python adoption is growing faster, but R's vast statistical libraries (CRAN) will make it a competitor for a long time. Neverheless, SQL s required to query databases and extract and pre-process data sets coming from relational databases. In the corporate world transactions remain an important data source, but non-transactional data and non-database data are growing faster. Data warehousing is now an old concept, which has been replaced by so-called data lakes. Cube processing and ad-hoc queries remain relevant in specialized data warehouses, but such techniques have been gradually subsumed by exploratory statistical analysis in big data. Hadoop big data systems are still required, given their scalability and flexibility to store and analyze big data, but they face a competition from more powerful workstations and containers and virtual machines in cloud computing (i.e. the scaleup vs scaleout dilemma). Both Hadoop systems and DBMSs interoperate with Python and R, at different levels depending on the specific analytic system. CSV text files and JSON are de-facto standard file formats to transfer big data, leaving proprietary formats behind.

Research and technology in Data Science opens many research issues, adapting and extending the state of the art in database systems, programming languages, data mining and parallel computing. Data Scientists and Data Engineers require programming languages and associated tools that are reasonably fast, intuitive and flexible (i.e. not necessarily the faster, but flexible to develop analytics). More research is needed to develop seamless mechanisms to convert data formats into each other, and perhaps avoid it. Many researchers and DS tool developers are moving away from the "fastest system" mentality, given hardware advances and the cloud, and instead they are rethinking algorithms and techniques to reduce development time and avoid mistakes. New analytic algorithms and techniques must be developed such that they can exchange data easily and they can interoperate, with more relaxed structure assumptions compared to traditional DBMSs.

References

[1] H. Garcia-Molina, J.D. Ullman, and J. Widom. *Database Systems: The Complete Book*. Prentice Hall, 2nd edition, 2008.

[2] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, San Francisco, 2nd edition, 2006.

[3] C. Ordonez and J. García-García. Managing big data analytics workflows with a database system. In *IEEE/ACM CCGrid*, pages 649–655, 2016.

[4] G. Ostrouchov, W.-C. Chen, D. Schmidt, and P. Patel. Programming with big data in r, 2012.

[5] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *IEEE 26th Symposium on Mass Storage Systems and Technologies, MSST*, pages 1–10. IEEE Computer Society, 2010.

[6] M. Stonebraker, D. Abadi, D.J. DeWitt, S. Madden, E. Paulson, A. Pavlo, and A. Rasin. MapReduce and parallel DBMSs: friends or foes? *Commun. ACM*, 53(1):64–71, 2010.

[7] M. Stonebraker, P. Brown, D. Zhang, and J. Becla. SciDB: A Database Management System for Applications with Complex Analytics. *Computing in Science and Engineering*, 15(3):54–62, 2013.

[8] Y. Zhang, C. Ordonez, and L. Johnsson. A cloud system for machine learning exploiting a parallel array DBMS. In *Proc. DEXA Workshops (BDMICS)*, pages 22–26, 2017.