Implementing a Linear Algebra Approach to Data Processing

Rogério Pontes[1], Miguel Matos[12], José Nuno Oliveira[1], and José Orlando Pereira[1]

[1] HASLab, INESC TEC & University of Minho, Braga, Portugal
[2] IST/INESC-ID, Lisboa, Portugal

Abstract Data analysis is among the main strategies of our time for enterprises to take advantage of the vast amounts of data their systems generate and store everyday. Thus the standard relational database model is challenged everyday to cope with quantitative operations over a traditionally qualitative, relational model.
A novel approach to the semantics of data is based on (typed) *linear* algebra (LA), rather than relational algebra, bridging the gap between data dimensions and data measures in a unified way. Also, this bears the promise of increased parallelism, as most operations in LA admit a 'divide & conquer' implementation.
This paper presents a first experiment in implementing such a typed linear algebra approach and testing its performance on a data distributed system. It presents solutions to some theoretical limitations and evaluates the overall performance.

Keywords: Formal methods, Linear Algebra, Big Data, Map Reduce, Hive

1 Introduction

In a world where data are generated faster than humans can analyze and comprehend, only decision support systems are capable of keeping up and providing analytics on time. Among these, Online Analytical Processing databases (OLAP) are used by data analysts to navigate across vast amounts of data and find business advantages or new opportunities.

Databases have long used *relational algebra* (RA) to model data storage, the operations that are carried out on data and the language used to interact with them. The so-called *relational model* is the formal underpinning of both OLTP[3] applications and OLAP[4] systems, in spite of serving two very distinct purposes and requirements. OLTP applications target small transactions and focus on the business frontend. OLAP systems focus on aggregating data to create information that is used by data analysts, so that these can decide which action might be best for their purposes. Harrah's Entertainmen improved their customer

[3] OLTP stands for "Online Transaction Processing".
[4] OLAP stands for "Online Analytical Processing".

practices and increased its revenue due to insights obtained from their customer center warehouses [1]. Another well-known example is Google's prediction of the influenza outbreaks in 2009. The prediction was made much quicker than the center of disease control (CDC) [2].

Similar results are achieved through *analytical queries* which, however, take a long time to complete, some taking hours or even days [3]. As data grow, this has a twofold impact. If, on one hand, queries take longer to complete, on the other hand more information can be extracted from the increased amount of data. As business success relies more and more on this kind of technology, it becomes increasingly important to have a fast and correct solution.

Macedo et al.[4] argue that relational algebra is adequate for giving semantics to the *qualitative* side of data, falling short where *quantitative* information is handled. They provide a novel approach based on *linear algebra* (LA) capable of not only expressing the semantics of OLAP system constructions such as data cube, roll up and cross tab, but also providing formal semantics for both the quantitative and the qualitative side of data. Their approach is columnar in the sense of representing columns in data relations by typed matrices, and is algebraic in the sense of relying only on matrix operations to encode queries.

One of the core promises of the typed linear algebra approach is the amount of parallel computation that can be performed, since matrix multiplication is a well-known 'divide & conquer' operation. On the negative side, the matrices involved are very large and sparse. If a proper storage format is not leveraged or if the data structure used does not take into account the operations required by a query, problems of performance and memory arise. Moreover, the theory requires an additional matrix product, named the *Khatri-Rao* product. This operation is essential to the algebra, capturing data joins in a simple and algebraic way.

Contribution. The main aim of this paper is to provide a distributed implementation of LA-based data processing. One of the challenges is that information needs to be consistent over a set of independent nodes. As such, the paper contributes to the LA-based approach to data analytics in several ways:

- Selection and improvement of an adequate sparse matrix format to handle the data and computation.
- Proposal of a *Khatri-Rao* product algorithm that can work on dense and sparse matrices.
- Proposal of a matrix encoding to keep the data consistent on a distributed setting.
- Evaluating the performance of query analysis tasks on a distributed setting with a typed linear algebra computation.

2 Background

Let T be a relational data table in a relational database. Looking at T we find two kinds of attributes (columns of the table): either they are numeric, and their values can be subject to numeric operations, or they are symbolic. Attributes of the first kind are *quantitative* in nature — they are referred to as data *measures*; those of the second kind are *qualitative* and are known as *dimensions*.

This dimension/measure binomial leads to two kinds of matrices in the LA approach: *dimension* matrices, also called *projection functions*, and *measure* matrices. The latter are diagonal matrices with as many rows/columns as the number of data in the corresponding data column. The former are Boolean matrices whose cells addressed by (d, n) hold 1 iff the data value d can be found in record n of the table and 0 otherwise. These two kinds of matrix are exemplified by the middle and bottom tables of Figure 1, respectively.

As can be seen from these examples, both matrices tend to be sparse; a projection matrix with n lines and m rows will have $m(n-1)$ zeros; if n and m are the same, then there are $n^2 - n$ zeros (quadratic growth), which is what happens in every measure matrix.

To work with matrices of this kind one needs special matrix formats that minimize memory usage. Fortunately, this topic has been heavily studied in the literature [5]. From the many formats available, one seems to the best suited: *Compressed Sparse Column* (CSC). It uses three arrays to store the information of a matrix: (a) an array "Values" holding all non-zero values, sorted by columns; (b) another array "Rows" keeping track of the original row position of values; (c) a final array "Pointer" indicating where every column starts and ends.

Many algorithms have been proposed for matrix product, from the naive $O(n^3)$ algorithm to the Strassen algorithm, which is $O(n^{2.81})$ operations, or the one proposed by Coppersmith and Winograd which is $O(n^{2.38})$ [6]. By contrast, the Khatri-Rao product which is central to the typed linear algebra [4] approach has not driven much attention in the literature. To the best of our knowledge, section 3 presents the first version of the algorithm tuned for sparse matrices.

Seasons	Quantity
Spring	20
Summer	35
Spring	10
Autumn	50

$[\![T]\!]_{Quantity}$	0	1	2	3
1	20	0	0	0
2	0	35	0	0
3	0	0	10	0
4	0	0	0	50

$t_{Seasons}$	1	2	3	4
Spring	1	0	1	0
Summer	0	1	0	0
Autumn	0	0	0	1

Figure 1. Example of source data (top table), measure matrix (middle table) and dimension matrix (bottom table).

3 Matrix Generation and Computation

Improving the CSC format. Dimension and measure matrices, illustrated in the middle and bottom table of Figure 1, have a special property: each matrix column contains one non-zero element only. This makes it possible to improve the CSC format even further. On the measure matrices the improvement is to use a single array containing only the values, instead of using the standard three-array version of CSC format. By storing the values sequentially and by storing the initial column position of the matrix when divided in multiples parts, it is possible to know the correct position of each value.

We want to store and process the matrix data in a distributed system, where each independent node stores an horizontal partition of each table. As a single node of the system only has a partial view of the whole table and it is going to generate part of the global matrices, it must be able to generate unique identifiers for every attribute. To solve this issue without relying on a global entity that keeps track of the id of every attribute, we apply a 64base encoding[5] that creates a unique *id* for each attribute and can be directly mapped to the attribute. The application of 64base encoding with the proposed matrix storage format is a novel matrix encoding that can be used to generate, store and process typed matrices in a data-distributed setting.

Implementing the Khatri Rao matrix product. Given two matrices A and B, with dimensions $n \times m$ and $p \times m$, respectively, the result of the Khatri Rao product of A by B, denoted by $A \triangledown B$, is a matrix with dimensions $np \times m$. The computation of $A \triangledown B$ can be seen as an iteration over the columns of the argument matrices (this is why A and B must have the same number of columns), by multiplying every element of each column of matrix A by every element of the corresponding column of B. Essential to understanding the algorithm being presented is to know how to calculate the position of the results on the output matrix. If M is the current matrix A row, N is the current matrix B row then the resulting position is given by $p \times M + N$.

Algorithm 1 KhatriRao product

```
Require: Marix $A(n \times m)$ , Matrix $B(p \times m)$
Ensure: Matrix $C(n * p \times m)$
  $C \leftarrow [n * p][m]$
  for $i = 0$ to $m - 1$ do
    for $j = 0$ to $n - 1$ do
      for $k = 0$ to $p - 1$ do
        $value \leftarrow A[j][i] * B[k][i]$
        $destLine \leftarrow p \times j + k$
        $C[destLine][i] \leftarrow value$
      end for
    end for
  end for
  return $C$
```

This algorithm has been added to the standard Linear Algebra library matrix-toolkits-java. This is an open source, high performance numerical library for matrix computation in Java. The algorithm has been adapted to work with the compressed matrix format presented in the previous section. In this format, when two projection functions are multiplied by a Khatri Rao product and since there is only one element per column the operation can be carried out in a lazy manner without having to create a dense matrix.

[5] Cf. 64 base encoding.

4 Evaluation

TPC-H[6] is an industry accepted OLAP benchmark. This section describes the experiments carried out to evaluate the implementation of our matrix encodings and operations using a modified version of TPC-H query 1. This evaluation was carried out with an Hadoop cluster with five servers, each with Ubuntu 14.04 64 bit, running on Intel Core i3-3240 @ 3.40 Ghz, 3K cache and 8GB of RAM. Data are generated from the TPC-H benchmark in the standard way. In this section we will introduce the derived query, its translation to a LA encoding and the overall set up of the tests carried out.

The query and its LA encoding. TPC-H query 1 was selected as first benchmark because it matches with several aspects of [4], namely: data are taken from a single raw data set (table), grouping involves two attributes only and the operation to be computed is a slice of a data cube. Thus data can be encoded as a vector filtered by the "where" clause.

Listing 1 Adapated TPC-H query1

```
SELECT Returnflag, Linestatus, Sum(Quantity)
FROM Lineitem
WHERE  Shipdate <= date ``1998-12-01'' - interval ``95'' day
GROUP BY Returnflag, Linestatus
```

Query 1 calls for three projection functions, one per attribute (*Returnflag*, *Linestatus*, *Shipdate*) and for a measure matrix for attribute *Quantity*. The 'group by' aggregation corresponds to the Khatri-Rao product of projection functions *Returnflag* and *Linestatus*. The result of this operation returns a matrix recording all possible combinations of values of such attributes:

$$(t_{Returnflag} \triangledown t_{Linestatus}) \cdot [\![T]\!]_{Quantity} \cdot filter \tag{1}$$

Second in the pipeline is the measure matrix which, composed with the Khatri-Rao outcome, yields the corresponding values for each combination. The final step,

$$filter = (!_{Shipdate \geqslant 1998-08-28 \wedge Shipdate \geqslant 1998-12-01})^{\circ}$$

is the multiplication by a column vector, which encodes the filtering of the data, which is denoted using the "!" notation of [4]. Altogether, this pipeline aggregates the values in the matrix rows and filters the results by a predicate on the *Shipdate* attribute.

[6] URL: http://www.tpc.org/tpch/default.asp.

Results. Pipeline (1) relies mostly on matrix multiplication, an operation which can be performed in a “divide and conquer” fashion, making the script a candidate for distributed data processing. This section gives the results of benchmarking this script on top of the Hadoop framework. This framework provides a fault tolerant distributed file system (HDFS) and a “Map-Reduce” application that allow us to run our query on top of it. Upon failures, the framework automatically resubmits failed jobs, hiding such complexity from the implementation. Additionally, resource scheduling in the cluster is of no interest in this experiment as it is a controlled environment that runs only the tasks that we assign them. The tasks are executed in batch mode and as such we don´t seek to assess the execution of the experiments with concurrent users.

We measure the job latency and resource usage. We compare our results with Hive [7], a Hadoop application that translates SQL to Map-Reduce jobs. Both applications divide a database table horizontally through the nodes and require an initial loading phase where the files containing the raw data are loaded and converted to the internal formats. In the experiments, Hive will be assessed with text file and optimized row format (ORC) without compression. Our approach does not use any compression either.

One machine hosts the HDFS name-node, the YARN resource manager and the Hive server. The remaining machines contain the HDFS data node and the YARN node manager. Each resource is given a 1GB JVM. In each machine 4GB are made available to YARN, which makes a cluster with a total amount of 16GB of RAM with 32 virtual cores. Each HDFS block has size 64MB.

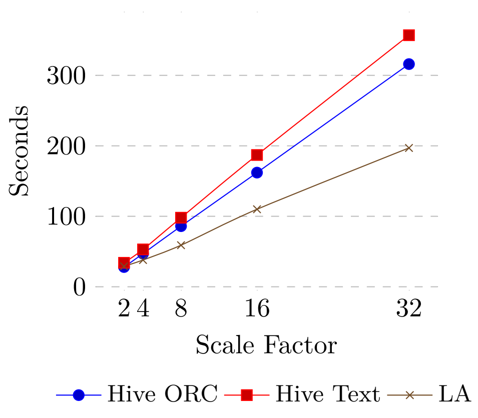

Figure 2. Job Latency

The experiment was executed over data generated by the TPC-H *LineItem* data table with the different scale factors that generate tables of increasing size. Scale factor 2 generates a table with an approximate size of 1.5GB while the scale factor 32 a table with size 23.5GB. The presented results are the average of a 10 run experiment. Figure 2 presents the average time it took to complete a job in the cluster. As can be observed, our approach has a significant improvement on the time it takes to compute the results. The improved latency comes mainly from the matrix encoding that not only allows to read the necessary dataset but also can be efficiently processed by the Khatri-Rao product. While relational algebra approaches require the processing to read every column of a dataset, our approach, similar to column oriented solutions, just processes the subset of the data crucial for the query [8].

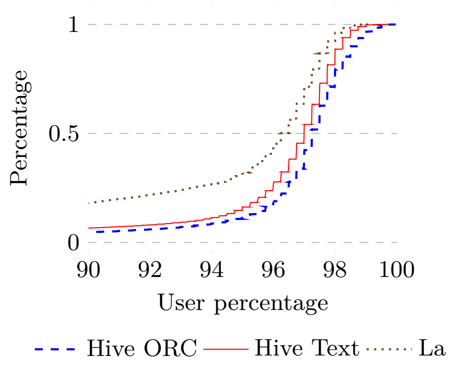

Figure 3. CPU Usage

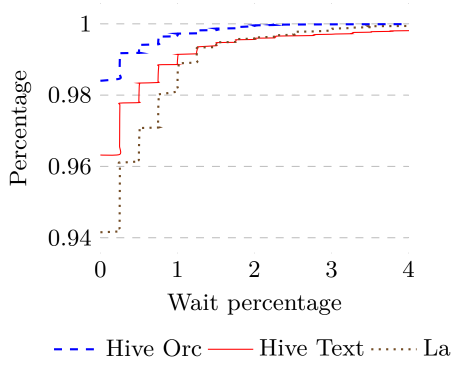

Figure 4. IO wait

The results presented in Figure 3 and 4 are a CDF (Cumulative Distribution Function) from all the experiments of the CPU usage using the *dstat* tool on the nodes that carry out the computation. From these plots we gather that the ORC format does a much better job at using the CPU as its usage percentage is between 96 and 100 while spending less time waiting for I/O operations. On the other hand, our approach has the CPU usage more distributed between 90 and 100 percent, which means that it spends more time waiting on I/O operations as can be seen in Figure 4. Even though not perceived in Figure 4, the ORC formats spends considerable less time in I/O.

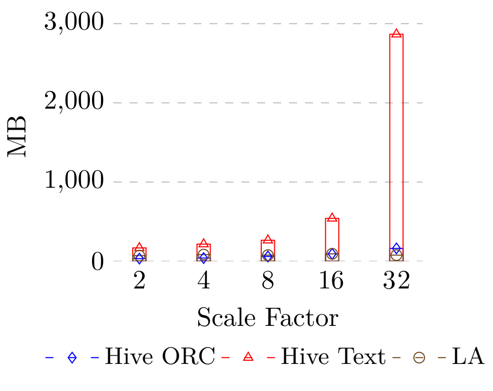

Figure 5. Data read from disk and over the network.

Hadoop reads a block of a file locally if available or reads it through the network otherwise. So we decided to aggregate the values from both channels to see which approach needs to read the least amount of bytes. From Figure 4 it becomes clear that the textual format used in Hive is the least efficient while a distinct pattern can not be found in the other approaches. Nonetheless, the Hive ORC on average seems to use less data on smaller sizes while our approach seems to use less data as scale factor increases. These results relate nicely with the latency time, explaining why our approach terminates much faster (it needs to read less information).

5 Conclusions and Future Work

This paper presents a first implementation and test of a typed linear algebra (LA) approach to data processing in a distributed environment. These preliminary results indicate that, compared to a standard Hive implementation, we have an interesting solution to further explore and test, as we witness an increase of 60% at most in the latency of the jobs and use about 45% less data on the best case. On the other hand Hive has a relative better CPU usage.

Recent developments [9] show similar advantages of the typed LA approach to data processing in parallel environments, while a strategy for translating SQL analytical queries to LA scripts is defined. We plan to automate this process, which will make our experiments much easier to carry out for the other TPC-H queries. This could also be applied to translating MDX queries.[10] Last but not least, and as anticipated in [9], linear algebra enables formally correct transformation of LA scripts, making it possible to compare different LA implementations of the same query for performance.

In spite of such positive results, definite conclusions can only be drawn once a comprehensive set of TPC-H queries is benchmarked. The main contribution of this short paper is to give a preliminary evaluation of the performance of LA scripts generated from SQL analytical queries running on a data distributed environment. This is a promising area of research that we intend to develop further in the future.

References

1. H. J. Watson and B. Wixom, "The Current State of Business Intelligence," *IEEE Computer*, pp. 96–99, 2007.
2. J. Ginsberg, M. H. Mohebbi, R. S. Patel, L. Brammer, M. S. Smolinski, and L. Brilliant, "Detecting influenza epidemics using search engine query data.," pp. 1012–1014, 2009.
3. E. F. Codd, S. B. Codd, and C. T. Salley, "Providing OLAP to User-Analysts: An IT Mandate," *Ann ArborMichigan*, p. 24, 1993.
4. H. Macedo and J. N. Oliveira, "A linear algebra approach to OLAP," *Formal Aspects of Computing*, pp. 1–25, 2014.
5. M. Silva, "Sparse matrix storage revisited," pp. 230–235, 2005.
6. D. Coppersmith, "Rectangular Matrix Multiplication Revisited," *J. Complexity*, pp. 42–49, 1997.
7. "Hive."
8. A. Floratou, J. M. Patel, E. J. Shekita, and S. Tata, "Column-Oriented Storage Techniques for MapReduce," *PVLDB*, pp. 419–429, 2011.
9. J. Oliveira, "Towards a linear algebra semantics for query languages," June 2016.
10. S. Bergamaschi, M. Interlandidi, M. Longo, L. Po, and M. Vincini, "A meta-language for MDX queries in elog business solution," in *2012 IEEE 28th Int. Conf. on Data Engineering*, pp. 1417–1428, 2012.