JMLR: Workshop and Conference Proceedings 36:88–103, 2014 BIGMINE 2014

The Gamma Operator for Big Data Summarization on an Array DBMS

Carlos Ordonez ORDONEZ@CS.UH.EDU
Yiqun Zhang YZHANG@CS.UH.EDU
Wellington Cabrera WCABRERA@CS.UH.EDU
University of Houston, USA

Editors: Wei Fan, Albert Bifet, Qiang Yang and Philip Yu

Abstract

SciDB is a parallel array DBMS that provides multidimensional arrays, a query language and basic ACID properties. In this paper, we introduce a summarization matrix operator that computes sufficient statistics in one pass and in parallel on an array DBMS. Such sufficient statistics benefit a big family of statistical and machine learning models, including PCA, linear regression and variable selection. Experimental evaluation on a parallel cluster shows our matrix operator exhibits linear time complexity and linear speedup. Moreover, our operator is shown to be an order of magnitude faster than SciDB built-in operators, two orders of magnitude faster than SQL queries on a fast column DBMS and even faster than the R package when the data set fits in RAM. We show SciDB operators and the R package fail due to RAM limitations, whereas our operator does not. We also show PCA and linear regression computation is reduced to a few minutes for large data sets. On the other hand, a Gibbs sampler for variable selection can iterate much faster in the array DBMS than in R, exploiting the summarization matrix.

Keywords: Array, Matrix, Linear Models, Summarization, Parallel

1. Introduction

Row DBMSs remain the best technology for OLTP [Stonebraker et al. (2007)] and column DBMSs are becoming a competitor in OLAP (Business Intelligence) query processing [Stonebraker et al. (2005)] in large data warehouses [Stonebraker et al. (2010)]. However, both kinds of DBMS are slow and difficult to use for mathematical computations on large data sets due to the abundance of matrices. Currently, Hadoop, MapReduce and no-SQL systems (HadoopDB, Cassandra, Hive, MongoDB, Mahout project, and so on) are the most popular for big data analytics. Array DBMSs (SciDB [Stonebraker et al. (2013)], Rasdaman [Baumann et al. (2013)]) represent alternative novel systems, closer to relational DBMSs, to support matrix computations, which share many features with old DBMSs like I/O efficient algorithms, indexing, query processing, concurrency control and parallel processing, but which have significantly different storage, retrieval and query processing mechanisms.

In this work, we focus on the optimization of algorithms to compute three fundamental and complementary models. Specifically, we study the computation of principal component analysis (PCA), linear regression (LR) and variable selection (VS) on large data sets on an array DBMS (SciDB). These three multidimensional models involve many matrix equations, which is an excellent fit for an array DBMS. However, these three models require different

numerical and statistical methods for the their solutions: PCA is generally computed with a Singular Value Decomposition (SVD) on the correlation (covariance) matrix of the data set; LR is solved with least squares via a matrix factorization; VS requires visiting an exponential search space of variable combinations, whose most promising solution is currently found by MCMC methods, used in Bayesian statistics [Gelman et al. (2003)]. With that motivation in mind, we introduce a fast generic algorithm that divides the computation into two phases: (1) summarization, (2) model computation, including iterative behavior. Our main contribution is a powerful summarization operator in an array DBMS, which allows computing many linear models reading the data set once and in parallel. We should point out that our array operator has generality and remarkable efficiency, as it can be exploited in many statistics and machine learning models, but it cannot summarize the data set for every potential model (e.g. SVMs, decision trees). Experimental evaluation on large data sets shows our summarization operator is much faster than the R package (even if the data sets fits in RAM) and much faster than a column DBMS (one of the fastest DBMSs currently available) using SQL queries. Moreover, we show existing matrix operators in SciDB are slower or exhibit RAM limitations.

2. Definitions

This section introduces mathematical definitions and models studied in this work. PCA represents an unsupervised (descriptive) model where there is no target or predicted attribute. Linear regression is a fundamental supervised model in statistics, whose solution helps understanding and building many other linear models. Variable selection (VS), a supervised model, is among the computationally most difficult problems in statistics and machine learning, in which Markov Chain Monte Carlo (MCMC) methods [Hastie et al. (2001); Gelman et al. (2003)] are a promising solution.

2.1. Data Set and Model

All models take a matrix X as input. Let $X = \{x_1, ..., x_n\}$ be the input data set with n points, where each point is in $\mathbf{R}^d$. That is, X is a $d \times n$ matrix, where x_i is a column d-vector (i.e., equivalent to a $d \times 1$ matrix) and $i = 1 \ldots n$ and $j = 1 \ldots d$ are used as matrix subscripts. In the case of LR and VS X is augmented with a $d+1$th dimension containing an output variable Y. We will use Θ, a set of matrices and associated statistics, to refer to the model in a generic manner.

2.2. Principal Components Analysis (PCA)

The objective of PCA is to reduce the noise and redundancy of dimensions by re-expressing the data set X on a new orthogonal basis, which is a linear combination of the original basis. In this case X has d potentially correlated dimensions but no Y. In general, PCA is computed on the covariance or the correlation matrix of the data set [Hastie et al. (2001)].

The PCA model Θ is U, a set of orthogonal vectors, called the principal components of the data set, ordered by their length (associated variance) in decreasing order and V, their squared eigenvalues. Such principal components are computed with a singular value decomposition (SVD) of the covariance V or correlation matrix ρ. In general, ρ is preferred

as it rescales all dimensions. Given ρ, its SVD is a symmetric matrix factorization expressed as $\rho = UVU^T$. In general, only a few components carry to most important information about variance of the data set and the remaining ones can be safely discarded to reduce d. When the data set has been normalized (i.e. with a z-score centering at the origin subtracting μ and rescaling by standard deviation σ_j) $\rho = XX^T$; we pay attention to the efficient computation of this matrix outer product. In this work we focus on getting U and V. The actual reduction of d to some lower dimensionality k (i.e., the k principal components) requires just a matrix multiplication, which is much simpler and faster than computing Θ.

2.3. Linear Regression (LR)

Let $X = \{x_1, \ldots, x_n\}$ be a set of n data points with d explanatory variables and $Y = \{y_1, \ldots, y_n\}$ be a set of values such that each x_i is associated to y_i. The linear regression model characterizes a linear relationship between the dependent variable and the d explanatory variables.

Using matrix notation, the data set is represented by matrices Y $(1 \times n)$ and X $(d \times n)$, and the linear model can be compactly defined as:

$$Y = \beta^T \boldsymbol{X} + \epsilon \tag{1}$$

where $\beta = [\beta_0, \ldots, \beta_d]$ is the column-vector of regression coefficients, ϵ represents Gaussian error and $\boldsymbol{X}$ is X augmented with an extra row of n 1s stored on dimension X_0. Notice X, in this model, is a $(d+1) \times n$ matrix, in order to make notation more intuitive. The vector β is usually estimated using the ordinary least squares method [Marin and Robert (2007)], whose solution is:

$$\hat{\beta} = (\boldsymbol{X}\boldsymbol{X}^T)^{-1}\boldsymbol{X}Y^T \tag{2}$$

2.4. Variable Selection (VS)

We now focus on one of the hardest analytics problems: variable selection in statistics, feature selection in machine learning. Since we have previously introduced linear regression, we will study variable selection in this model. However, our algorithms have the potential to be used in other models such as probit models [Hastie et al. (2001)], Bayesian classification [Ordonez and Pitchaimalai (2010)] and logistic regression [Hastie et al. (2001)], all of which have similar underlying matrix computations.

The input is X, like LR, but it is necessary to add model parameters. In general, a linear regression model is difficult to interpret when d is large, say more than 20 variables. The search for the best subsets of explanatory variables that are good predictors of Y [Guyon and Elisseef (2003)] is called variable selection. The assumption of this search is that the data set contains variables that are redundant, or that they have low predictive accuracy and thus they can be safely excluded from the model. When d is high, an exhaustive search on the 2^d subsets of variables is computationally intractable for even a moderately large d. Traditional greedy methods, for instance stepwise selection [Hastie et al. (2001)] can only find one subset of variables, likely far from the optimal, but the probabilistic Bayesian approach aims to identify several promising subsets of variables, which in theory can be

closer to the optimal. In this work, the set of selected variables will be represented by a d-dimensional vector $\gamma \in \{0,1\}^d$, such that $\gamma_j = 1$ if the variable j is selected and $\gamma_j = 0$ otherwise. We denote by Θ_γ the model that selects k out of the d variables, corresponding to the vector γ. The dot product $k = \gamma^T \cdot \gamma$ indicates how many variables were selected. Throughout this paper we will use γ as an index to project matrices on selected variables such as β_γ and $\boldsymbol{X}_\gamma$.

MCMC Methods: the Gibbs Sampler

We use the Bayesian formulation and Gibbs sampler introduced in [Marin and Robert (2007)] based on a Zellner G-prior, which has the outstanding feature of sharing several intermediate matrix computations with the other models defined above. Given a model Θ with K parameters, one of the parameters is sampled from its prior distribution, while the others $K-1$ parameters remain fixed. The set of variables selected at iteration i of the sequence of observations is denoted by a vector $\gamma^{[i]}$. The Gibbs sampler for variable selection generates the Markov chain: $\gamma^{[0]}, \ldots, \gamma^{[N]}$, in which it is expected the best variable subsets will appear more frequently. At each iteration I, the Gibbs sampler generates $\gamma^{[I]}$ based on the previous state of the Markov chain, $\gamma^{[I-1]}$. Since each coordinate $\gamma_j^{[I-1]}$ is a parameter of model Θ_γ, the Gibbs sampler visits all d entries of $\gamma^{[I]}$. In more detail, for every variable $\gamma_j^{[i]}$ the normalized probabilities $p(\gamma_j^{[i]} = 0)$ and $p(\gamma_j^{[i]} = 1)$ are calculated. Based on these probabilities either $\gamma_j^{[i]} = 0$ or $\gamma_j^{[i]} = 1$ is chosen by sampling and the jth position of vector $\gamma_j^{[i]}$ is updated accordingly. The Markov chain becomes stable after an initial number of iterations, known as the *burn-in* period B. Then the MCMC method iterates for a large number of iterations until the posterior probabilities of γ have been sufficiently explored (essentially a very demanding computation, which requires many trial and error runs).

3. A Fast Summarization Array Operator

We start by reviewing matrices containing sufficient statistics. Then we generalize them into a single matrix that can be exploited in several intermediate demanding computations for the linear models introduced in Section 2. Once we establish the importance of this summarization matrix, called Γ, we introduce an array operator and we carefully analyze how to optimize its computation.

3.1. Sufficient Statistics

A significant body of work, in the past in database mining and now in big data analytics, has focused on summarizing the data set. Several linear models, including PCA and linear regression, can be more efficiently computed using sufficient statistics, as proposed in [Ordonez (2010)]. We significantly generalize such work, looking at variable selection and MCMC methods under a unifying mathematical framework.

We first review the general definition of sufficient statistics. Then we integrate and generalize such matrices for the three models studied in this work. Sufficient statistics are defined as:

$$n = |X| \tag{3}$$

$$L = \sum_{i=1}^{n} x_i \tag{4}$$

$$Q = XX^T = \sum_{i=1}^{n} x_i \cdot x_i^T \tag{5}$$

$$\tag{6}$$

Intuitively, n counts points, L is a linear sum and Q is a quadratic sum, with x_i in outer product powers 1 and 2, respectively. As explained in Section 2.3 , the linear regression model uses an *augmented* matrix, represented by $\mathbf{X}$. We introduce a more general augmented matrix $\mathbf{Z}$, created by appending one additional row to $\mathbf{X}$, which contains the vector Y. Since X is $d \times n$, $\mathbf{Z}$ has $(d+2)$ rows and n columns. We will study how to compute the matrix outer product (related, but not the same as the Gram matrix $\mathbf{Z}^T\mathbf{Z}$):

$$\Gamma = \mathbf{Z}\mathbf{Z}^T \tag{7}$$

Matrix Γ contains a comprehensive and accurate summary of X, sufficient to compute all models previously defined. We explain below how this is done for each model.

$$\begin{aligned}
\Gamma &= \begin{bmatrix} \mathbf{Q} & \mathbf{X}Y^T \\ Y\mathbf{X}^T & YY^T \end{bmatrix} \\
&= \begin{bmatrix} n & L^T & \sum y_i \\ L & Q & XY^T \\ \sum y_i & YX^T & YY^T \end{bmatrix} \\
&= \begin{bmatrix} n & \sum x_i^T & \sum y_i \\ \sum x_i & \sum x_i x_i^T & \sum x_i y_i \\ \sum y_i & \sum y_i x_i^T & \sum y_i^2 \end{bmatrix}
\end{aligned}$$

Matrix Γ is comparatively much smaller than X for big data and symmetric. Such facts are summarized in the following properties:

Property 1: Given a large data set X where $d \ll n$, matrix Γ is $O(d^2) \ll O(dn)$ as $n \to \infty$.

Property 2: $\Gamma = \Gamma^T$, resulting in $d + (d+2)^2/2 \approx d^2/2$, saving 1/2 of CPU operations.

Property 3: Γ can be equivalently computed as follows: $\Gamma = \mathbf{Z}\mathbf{Z}^T = \sum_{i=1}^{n} z_i \cdot z_i^T$.

Property 1 means that if Γ can fit in main memory it is feasible to maintain a summary of X in main memory. However, we emphasize we cannot assume X can fit in main memory. Therefore, the challenge is to efficiently compute Γ, minimizing the number of times X must be read from secondary storage. Property 2 is straightforward, but saves half of CPU work.

Table 1: Summary of matrices.

Matrix	Size	Description
X	$d \times n$	Data set: independent variables, dimensions
U	$d \times d$	Principal components
V	$d \times d$	Squared eigenvalues; diagonal
Y	$1 \times n$	Dependent variable
$\boldsymbol{X}$	$(d+1) \times n$	Augmented matrix with 1s
β	$(d+1) \times 1$	Regression coefficients including Y intercept
γ	$d \times 1$	Selected variables; binary vector
$\mathbf{Z}$	$(d+2) \times n$	Augmented X matrix with 1s and Y
Γ	$(d+2) \times (d+2)$	Generalized summarization matrix

Property 3 gives two equivalent expressions to get Γ: one as a matrix multiplication and a second one as a sum of vector products. Given the prominent importance of Γ we believe it should be available as an operator on any big data analytic system. Therefore, we will call our operator $\Gamma(X)$. Table 1 provides as summary of matrices, including the input data set, the output variable, model Θ and Γ.

3.2. Two Phase Algorithm

Based on Γ our algorithm to compute Θ works in two phases:

1. Compute Γ.
2. Exploit Γ in intermediate matrix computations.

This paper focuses on optimizing Phase 1, which is a well defined problem for any input data set X. Phase 2 requires incorporating Γ in the steps of numerical and statistical methods. By exploiting Γ it becomes possible to reduce the number of times X is read, and to reduce CPU computations in iterative methods. Identifying in which models intermediate matrix computations accept Γ is a deep research topic. We point out that this paper builds on top of previous research and we make clear to which kind of models our optimizations apply. That is, our choice of PCA, LR and VS has been carefully done based on the fact that they can exploit Γ.

3.3. Array DBMS: SciDB

An array DBMS enables the manipulation of multidimensional arrays of unlimited size, removing RAM limitations in a large matrix computation. The fundamental difference between SciDB and a row or column DBMS is storage by chunk. Such chunk requires a different kind of programming and optimization, compared to UDFs or stored procedures in SQL.

SciDB organizes array storage by chunks, as explained in [Stonebraker et al. (2013)]. A chunk is a large multidimensional block on secondary storage that stores a preferably large fraction of values for one attribute. The storage of an array in SciDB can be visualized like

a grid of rectangular chunks, where each chunk comprises a rectangular area of contiguous cells, as defined by the array dimensions.

The major question is how to program the computation of Γ in SciDB. The major choices are: exporting (with a bulk mechanism) the X matrix to a linear algebra package (LAPACK), in the query language provided by the DBMS (AQL, AFL), calling existing array operators or as a new array operator. Exporting data is evidently a bad idea, as it defeats the purpose of the DBMS, no matter how fast the computation can happen outside. The query language is a second option, but we will show it is inefficient as it requires a similar evaluation query plan as an SQL query with a costly relational join operation. Existing operators, will be shown to be inadequate as Γ is a is a matrix product that is particularly difficult to optimize. Last, we will show a new operator is required, providing orders of magnitude time improvement. These are the alternatives we will compare:

1. Query: programmed in AQL. Such query requires a join operation with arrays.
2. R operator: crossprod() function, available in SciDB-R to compute Γ
3. AFL function call: use gemm() and transpose() built-in operators to compute Γ
4. New operator: programmed in C++ in the SciDB extensibility interface.

The query alternative is interesting as it establishes a connection with query optimization, a classical topic in database systems research. The R operator is the choice for the typical R user of SciDB. SciDB provides an R library that allows users to embed key matrix computations on SciDB. It should be pointed out it is difficult to identify which and how such computations should be mapped to SciDB. The function call is a compromise between an export and a main memory computation. Basically, the data set needs to reorganized to be passed to the efficient LAPACK library. Such conversion is necessary because LAPACK is a compiled Fortran library, which has a very simple, but different, memory management at run-time. Finally, our new operator requires developing a "chunk-aware" algorithm that we explain below.

3.4. The Gamma Operator on an Array DBMS

Previous work [Ordonez (2010)] on the computation of statistical models on a relational DBMS showed that the most efficient mechanism to compute sufficient statistics are aggregate UDFs, complemented by scalar UDFs for scoring. The main drawback is that they are tailored to relational DBMSs, which are known to be slow and cumbersome for matrix computations.

As pointed out by previous research [Ordonez (2010)], it is straightforward to compute n, L as they only require sums of a value (counting) and summing a d-vector. Recalling that Γ has n, L, Q as submatrices. the main difficulty is getting Q which requires cross-products among all dimension pairs. Therefore, computing Γ subsumes computing Q.

From a high performance angle, we will focus on optimizing Γ assuming X is a dense matrix. For high d X will likely be a sparse matrix, which in turn will require a sparse matrix multiplication for Γ. Since sparse matrices require completely different storage and access methods such operator is reserved for future research.

Observation: We make the following fundamental observation. Even though it is tempting to evaluate $\mathbf{Z}\mathbf{Z}^T$ as a matrix multiplication it would be a bad idea: we need to compute and materialize $\mathbf{Z}^T$ first, a costly transposition, storing it and then perform the actual matrix multiplication. Instead, we will study how to evaluate the summation $\sum_{i=1}^{n} z_i \cdot z_i^T$. Assuming z_i fits in main memory then it is reasonable to assume the vector outer product also fits in main memory. Therefore, it makes more sense to evaluate such sum in parallel.

Our operator requires that all data of z_i (x_i in consequence) fits in one chunk. In other words, z_i cannot be partitioned into several chunks in spanned storage. This requirement is important to accelerate I/O: larger chunks improve I/O because our Γ computation requires a full scan on the data set. Smaller chunks would trigger seeks, resulting in worse performance. Fitting many points x_i in one chunk is not a major limitation because SciDB favors fairly large chunk sizes, typically ranging above 8MB. Another major requirement and advantage is that the operator must work in parallel. It is a requirement that complicates programming because it is essential to design and develop the operator in such a way that X can be evenly partitioned across N processing nodes, with minimal or no need of synchronization overhead. The advantage is, of course, the ability to scale processing to more nodes as the data set size grows. Our Γ operator works fully in parallel with a partition of X into N database subsets $D_1 \cup D_2 \cup \cdots \cup D_N = X$, where we compute Γ_I on D_I for each node I. In SciDB terms, each worker I will compute Γ_I. When all workers are done the coordinator node will gather all results and compute a global $\Gamma = \Gamma_1 + \cdots + \Gamma_N$. This is essentially an obvious parallel computation [Stonebraker et al. (2010)], coming from the fact that we can push the actual multiplication of $z_i \cdot z_i$ into the array operator. The sequential algorithm to compute Γ is below whose generalization to a parallel one is straightforward. Given the additive properties of Γ the same algorithm is applied on each node $I = 1 \ldots N$ to get Γ_I, combining all partial results $\Gamma = \sum_I \Gamma_I$ in the coordinator node.

Data: $X = \{x_1, x_2, \ldots, x_n\}, Y = \{y_1, \ldots, y_n\}$
Output: Γ

$\mathbf{Z} \leftarrow [1, X, Y] = \{z_1, z_2, \ldots, z_n\}$
$\Gamma \leftarrow \mathbf{0}$
for $i = 1 \ldots n$ **do**
  for $a = 0 \ldots d+1$ **do**
    for $b = 0 \ldots d+1$ **do**
      $\Gamma_{ab} \leftarrow \Gamma_{ab} + z_{ia} * z_{ib}$
    end
  end
end

3.5. Algorithms Exploiting Gamma Operator

3.5.1. Principal Component Analysis (PCA)

As mentioned in Section 2.2, PCA can be calculated by computing SVD on the correlation matrix. To compute PCA we rewrite the correlation matrix equation based on the sufficient statistics in Γ: $\rho_{ab} = (nQ_{ab} - L_a L_b)/(\sqrt{nQ_{aa} - L_a^2}\sqrt{nQ_{bb} - L_b^2})$. The PCA algorithm two phases are:

1. Compute Γ, which contain n, L, Q
2. Compute ρ, solve SVD of ρ, select the k principal components.

3.5.2. Linear Regression (LR)

The LR can be solved by the least squares method: $\hat{\beta} = (XX^T)^{-1}XY^T$. As explained above, Γ contains XX^T and XY^T. Therefore, the LR algorithms becomes:

1. Compute Γ, which contains XX^T and XY^T.
2. Solve $\hat{\beta}$.

3.5.3. Variable Selection (VS)

Recall we perform variable selection with an MCMC method, potentially requiring thousands of iterations. We base the computation of $\pi(\gamma|X,Y)$ on the Zellner G-prior due to [Marin and Robert (2007)]. Zellner's G-prior relies on a conditional Gaussian prior for β and an improper (Jeffreys) prior for σ^2 [Marin and Robert (2007)], with parameters c and $\widetilde{\beta}$. The most common prior probability for γ is the uniform prior $\pi(\gamma|X) = 2^{-d}$ [George and McCulloch (1993); Marin and Robert (2007)]. Under Zellner's G-prior, the parameter c influences the number of variables selected by the model: large c values lead to parsimonious models (few variables), whereas small values of c promote saturated models (many variables) [Liang (2008)]. On the other hand, $\tilde{\beta}$ is a hyper-parameter to impose a Gaussian on β.

$$\beta|\sigma^2, X \sim \mathcal{N}_{k+1}\left(\tilde{\beta}, c\sigma^2\left(XX^T\right)^{-1}\right) \tag{8}$$

$$\sigma^2 \sim \pi\left(\sigma^2|X\right) \propto \sigma^{-2} \tag{9}$$

For notational convenience $\mathbf{X}_\gamma, \tilde{\beta}_\gamma, \mathbf{Q}_\gamma$ mean projecting $\mathbf{X}, \tilde{\beta}, \mathbf{Q}$ on selected variables. Recall $\mathbf{X} = [1, X]$. The full equation to get the posteriors of γ involves:

$$\pi(\gamma|X,Y) \propto (c+1)^{-\frac{k+1}{2}}\left(YY^T - \frac{c}{c+1}(Y\mathbf{X}_\gamma{}^T)(\mathbf{Q}_\gamma)^{-1}(\mathbf{X}_\gamma Y^T) - \right.$$
$$\left. -(c+1)^{-1}\tilde{\beta}_\gamma^T\mathbf{Q}_\gamma\tilde{\beta}_\gamma\right)^{-n/2} \tag{10}$$

The Zellner G-prior is computationally convenient for big data, because equations can be expressed in terms of Γ. We will use the sufficient statistics by equations 3, 4 and

Table 2: Data sets.

Data set	d	n	Description
KDDnet	100	10M	Network intrusion detection (KDD Cup)
Year	90	500k	Predict year of song release (UCI)

5 in order to avoid recomputing those matrix products at each iteration. Therefore, our optimized algorithm becomes

1. Compute Γ, which contains $\mathbf{X}_\gamma {\mathbf{X}_\gamma}^T$ and $\mathbf{X}_\gamma Y^T$.
2. Iterate the Gibbs sampler a sufficiently large number of iterations to explore $\pi(\gamma|X, Y)$

3.6. Time Complexity and Cost Analysis

Phase 1: Computing Γ is clearly $O(d^2 n)$; taking into account that Γ is symmetric the CPU cost is halved. Phase 2: Since we are computing various factorizations derived from Γ time is $\Omega(d^3)$, which for a hard dense matrix problem it may approach $O(d^4)$, when the number of iterations in the method is proportional to d. I/O cost is just reading X with the corresponding number of chunk I/Os.

.

4. Experimental Evaluation

This section presents an experimental evaluation with well-known data sets used in data mining. Since our operator and the 2-phase algorithm do not change the accuracy of matrices we do not measure accuracy. We should emphasize that our summarization operator was used to produce the Γ matrix to be consumed by the R package. That is, the actual model Θ computation happened in R. Therefore, the model computed by R and the model computed by SciDB in tandem with R are the same. We first present time benchmarking experiments comparing our operator with other systems, including R. Then we study scalability and speedup on a large cluster in the cloud.

4.1. Experimental Setup

We used two data sets: the network intrusion data set and the year prediction data set, summarized in Table 2, obtained from the KDD 99 Cup and UCI machine learning repository, respectively. We sampled and replicated both data sets to get varying n (data set size) and d (dimensionality). The original KDDnet data set had n=5M and $d = 34$ and many zeroes, resulting in singular matrices. Its columns were replicated 3X and rows 2X to get $d = 100, n =$ 10M to analyze time performance. Then rows were sampled or replicated in log-10 scale. On the other hand, Year had few zeroes, resulting in non-singular (numerically good) matrices. In this case Year rows were replicated to get larger n in log-2 scale. Scalability on d, being computationally more demanding and numerically more problematic, is not analyzed in this paper.

Table 3: Software and hardware.

Item	Local	Cloud
Array DBMS	SciDB 14.3	SciDB 14.3
OS	Linux Ubuntu	Linux Ubuntu
Processor	Intel Quadcore CPU 2.153 GHz	Intel Xeon E5-2680v2, 8vCPUs
RAM	4 GB	15 GB
Storage	HD 3 TB	SSD 1 TB

Table 4: Summary of cluster configurations (default in bold face).

Cluster	Nodes	Instances	Cores	RAM	Disk
Local 1/1	1	1	4	4 GB	3 TB
Local 1/2	1	2	4	4 GB	3 TB
Local 1/4	1	4	4	4 GB	3 TB
Local 2/4	2	4	8	8 GB	6 TB
Cloud 1/4	1	4	8	15 GB	1 TB
Cloud 2/8	2	8	16	30 GB	1 TB

From a software angle, we used the SciDB DBMS as it supports the development of new operators, it provides a big library of functions and it is fully parallel. For comparison purposes, we used the Vertica DBMS, one of the fastest column DBMSs today to evaluate SQL queries on large databases. For statistical processing the R package is the standard with its well known RAM limitation. Finally, all our systems used LAPACK underneath for difficult linear algebra computations (e.g. SVD, matrix inversion). We tested on three hardware configurations: local server 1 node (default), local cluster with 2 nodes and a cluster in the Amazon cloud with 2 powerful nodes. Table 3 summarizes the local server/cluster, whereas Table 4 shows the clusters.

For PCA and LR we used the R default values to get accurate results. For VS we used a small number of iterations (1000) since this algorithm is CPU bound (after being optimized) and we were interested in understanding performance, rather than a reliable Bayesian model which requires many trial/error runs and tens of thousands of iterations (i.e., exploring the posterior probability and evaluating convergence of the Markov Chain).

We ran each system three times and we report the average. When a system crashes, mainly due to RAM limitations, we report "fail". When the computation could not finish in 30 minutes we report "stop". In general we show execution times in seconds.

4.2. Comparing Summarization Operator: array DBMS, R and column DBMS

We start by comparing R with our SciDB program. For PCA and LR we use the princomp() and lm() standard R function calls. For VS in R we use the R script available from [Marin and Robert (2007)]. The original script was so slow that it could not run in under 1 hour even for n=1k. Therefore, we were forced to incorporate our optimization based on Γ. The SciDB operator computes Γ, which is passed to an R script further optimized by us. Table 5 compares total time to get Θ in R and the DBMS+R. R scales in a decent manner, but

Table 5: Computing model Θ; data set KDDnet $d = 100$; 1 node local 1/2; time in secs.

n	PCA		LR		VS	
	R	Γ operator + R	R	Γ operator + R	R	Γ operator + R
10k	0.7	0.8	0.7	0.9	13.1	13.0
100k	5.7	2.5	6.3	2.6	34.4	14.8
1M	61.2	16.8	60.5	16.9	113.0	29.1
10M	fail	194.9	fail	194.9	fail	207.2

Table 6: Comparing only summarization; data set KDDnet $d = 100$; 1 node local 1/2; time in secs.

n	R	SQL queries	Γ operator
10k	0.9	3.1	0.3
100k	6.5	27.1	1.3
1M	44.1	612.5	13.7
10M	fail	13286.5	150.2

eventually crashes due to RAM capacity. These times prove the power and superiority of our operator and our 2-phase algorithm.

Clearly, Γ can be used on other systems, beyond an array DBMS. We investigate how Γ performs on each system. For the column DBMS we use a query storing X on a vertical layout with one entry X_{ij} per row (i, j, v). Table 6 provides a straight comparison of our operator programmed in each system. For R the data set was loaded in RAM and Γ was obtained with a matrix multiplication using %*% (i.e. the fastest mechanism available). For SQL queries we defined the table in such a way that the DBMS could do a merge-join (fastest join possible in time $O(n)$. Our operator is 3X faster than R and two orders of magnitude faster than SQL queries.

We would like to emphasize our time measurements on SciDB do not include the time to load the data set (neither on the column DBMS) and apply a redimension operator to convert it from table to matrix because they are a one-time event. To give an idea about this bottleneck with KDDnet $d = 100$, n=1M, loading took 118 secs and redimensioning 208 secs, totaling 326 secs. In contrast, our operator took 14 seconds as shown on Table 6.

We now analyze if it is indeed necessary to program a specialized summarization Γ operator at the chunk level, instead of taking advantage of other built-in operators in the

Table 7: Comparing SciDB programming mechanisms; data set KDDnet $d = 100$; 1 node local 1/2; time in secs.

n	AQL/AFL	spgemm()	SciDB-R	Γ operator
10k	168.3	3.1	19.1	0.3
100k	stop	15.3	76.8	1.3
1M	stop	177.8	stop	13.7
10M	stop	fail	stop	150.2

Table 8: Correlation matrix ρ; local 2/4; data set Year $d = 90$; time in secs.

n	size	SciDB-R	Γ operator
500k	0.4 GB	240	6
1M	0.7 GB	559	13
2M	1.4 GB	fail	26
8M	5.8 GB	fail	103
16M	11.6 GB	fail	205

Table 9: Time to compute models using Γ in local cluster local cluster 1/4; dataset Year $d = 90$; time in secs.

n	Γ operator	LR	PCA	VS
1M	19	0.02	0.08	143
2M	36	0.02	0.09	146
8M	118	0.02	0.09	145
16M	233	0.02	0.09	148

array DBMS. Table 7 compares all available programming alternatives on KDDnet. SciDB offers two flavors of matrix multiplication: sparse (spgemm()) and dense (gemm()). We tested both of them and spgemm() was able to handle larger matrices. Clearly, our operator scales better and more importantly, it does not present RAM limitations.

4.3. Parallel Processing

In Table 10, we present another set of experiments measuring running time for our custom operator varying data set size (n). In order to analyze scalability and parallelism, these experiments are conducted increasing the number of SciDB instances, in 1 and 2 servers, respectively. Such setup allows us to measure intra-node parallelism interleaving CPU and I/O. In our local system we vary from 1 to 4 instances, whereas experiments on the Amazon EC2 cluster, instances vary from 4 to 8. These execution time measurements are plotted in Figure 1. In order to evaluate scalability, the parallel speed up can be easily computed from the time measurements in Table 10. Clearly, the scalability of our custom operator is nearly perfect, when our servers run with up to 2 instances. The small RAM of our servers and contention for CPU and disk hinder an ideal speedup for 4 instances on one server.

Table 10: Scalability of Γ summarization operator; dataset Year $d = 90$; time in secs.

n	Local 1/1	Local 1/2	Local 1/4	Local 2/4	Cloud 1/4	Cloud 2/8
500k	25	12	9	6	3	2
1M	51	26	19	13	7	3
2M	101	55	36	26	14	6
8M	410	227	118	103	59	26
16M	818	455	232	205	109	52

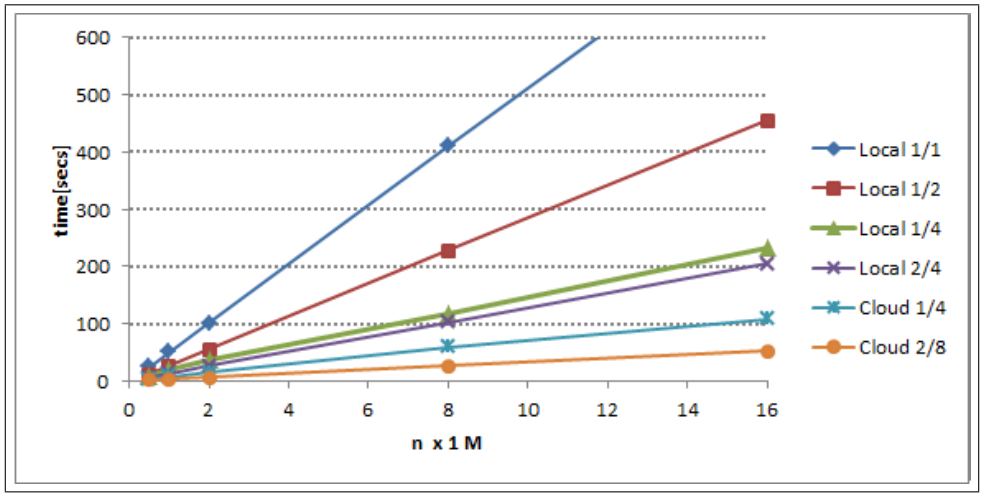

Figure 1: Time for 1,2,4,8 SciDB instances varying n (data set size).

5. Related Work

Optimizing matrix computations when matrices cannot fit in RAM is a classical topic in numerical linear algebra [Demmel (1997)], exploiting significantly different algorithms and data structures compared to database systems. On the cloud computing side the currently dominating technology is the Hadoop ecosystem including HDFS, MapReduce and no-SQL systems. Despite the prominence of linear models in statistics, there is not much work studying how to accelerate their computation on a DBMS. Integrating linear regression with a DBMS has received limited attention, compared to more popular models like clustering [Ordonez and Cereghini (2000)] and decision trees [Gehrke et al. (1999)]. More recently, [Hellerstein et al. (2012)] explains the MADlib library to compute statistical models with SQL mechanisms, combining queries and UDFs, following a similar summarization method presented to the one presented in [Ordonez (2010)] to compute linear regression. Nevertheless, neither [Ordonez (2010)] nor [Hellerstein et al. (2012)] had envisioned a vector-based outer product computation which can be pushed into an aggregation with arrays as input and output. From a mathematical point of view, variable selection and Bayesian models, are much more difficult. System requirements for an array DBMS are outlined in [Stonebraker et al. (2009)], whereas [Stonebraker et al. (2013)] provides a technical overview of SciDB. Optimizing the computation of statistical models and matrix operations on array DBMSs, with SciDB in particular, is not well explored.

6. Conclusions

We studied how to efficiently compute three different and well-known statistical models on an array DBMS. Specifically, we studied PCA, linear regression and Bayesian variable selection with MCMC methods. We identified common sufficient statistics across all models to summarize the data set. After the data set is summarized, sufficient statistics are used to derive the correlation matrix for PCA (solved with SVD), compute linear regression

coefficients (solved with least squares) and get a Bayesian variable selection model (using the well-known Gibbs sampler). Our main contribution is a new array operator that enables fast summarization of a large data set for a big family of linear models, including those presented in this paper. This operator has the outstanding property of pushing vector products to RAM, avoiding materializing on secondary storage a matrix transposition and eliminating a cross-product between two large matrices. We carefully analyzed parallel speedup, performance bottlenecks and scalability. Experimental evaluation on the SciDB parallel DBMS produced encouraging results. Our operator is remarkably efficient, two orders of magnitude faster than SQL, an order of magnitude faster than SciDB built-in operators and even faster than the R package when the input matrix fits in RAM. Moreover, our operator does not present any RAM limitations, whereas some SciDB operators do. The remaining model computations take negligible time for PCA and linear regression, whereas the Gibbs sampler does take significant time, but it iterates fast in RAM. In summary, our summarization array operator has linear scalability as data set size grows and linear speedup as the number of processing nodes grows. We believe this operation will help array DBMSs being used in more big data analytics problems.

There are many research opportunities. The next step is to investigate how to handle a summarization matrix that does not fit in RAM. This is particularly important for genomics and text data. Our operator does not exploit LAPACK to perform vector multiplications in RAM. Thus it is necessary to investigate if it is worth converting vectors to major column order to be consumed by LAPACK. Our operator has wide applicability, but it certainly does not work for every model. However, the same approach going from matrix to vector, back to matrix, may work for other models. Last but not least, we need to develop incremental algorithms that can produce model approximations as the data set is being scanned.

Acknowledgments

We thank the comments from Michael Stonebraker to understand SciDB storage and retrieval mechanisms. This work was conducted while the first author was visiting MIT.

References

P. Baumann, A. Dumitru, and V. Merticariu. The array database that is not a database: file based array query answering in Rasdaman. In *Advances in Spatial and Temporal Databases*, pages 478–483. Springer, 2013.

J.W. Demmel. *Applied Numerical Linear Algebra*. SIAM, 1st edition, 1997.

J. Gehrke, Venkatesh Ganti, and R. Ramakrishnan. BOAT-optimistic decision tree construction. In *Proc. ACM SIGMOD Conference*, pages 169–180, 1999.

A. Gelman, J.B. Carlin, H.S. Stern, and D.B. Rubin. *Bayesian Data Analysis*. Chapman and Hall/CRC, 2003.

E. George and R. McCulloch. Variable selection via Gibbs sampling. *Journal of the American Statistical Association*, 88(423):881–889, 1993.

I. Guyon and A. Elisseef. An introduction to variable and feature selection. *The Journal of Machine Learning Research*, 3:1157–1182, 2003.

T. Hastie, R. Tibshirani, and J.H. Friedman. *The Elements of Statistical Learning*. Springer, New York, 1st edition, 2001.

J. Hellerstein, C. Re, F. Schoppmann, D.Z. Wang, and et. al. The MADlib analytics library or MAD skills, the SQL. *Proc. of VLDB*, 5(12):1700–1711, 2012.

F. Liang. Mixtures of g priors for Bayesian variable selection. *Journal of the American Statistical Association*, 103(481), 2008.

J.M. Marin and C.P. Robert. *Bayesian Core: A Practical Approach to Computational Bayesian Statistics*. Springer, 2007.

C. Ordonez. Statistical model computation with UDFs. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 22(12):1752–1765, 2010.

C. Ordonez and P. Cereghini. SQLEM: Fast clustering in SQL using the EM algorithm. In *Proc. ACM SIGMOD Conference*, pages 559–570, 2000.

C. Ordonez and S. Pitchaimalai. Bayesian classifiers programmed in SQL. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 22(1):139–144, 2010.

M. Stonebraker, D.J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E.J. O'Neil, P.E. O'Neil, A. Rasin, N. Tran, and S.B. Zdonik. C-Store: A column-oriented DBMS. In *Proc. VLDB Conference*, pages 553–564, 2005.

M. Stonebraker, S. Madden, D. J. Abadi, S. Harizopoulos, N. Hachem, and P. Helland. The end of an architectural era: (it's time for a complete rewrite). In *Proc. VLDB Conference*, pages 1150–1160, 2007. ISBN 978-1-59593-649-3.

M. Stonebraker, J. Becla, D.J. DeWitt, K.T. Lim, D. Maier, O. Ratzesberger, and S.B. Zdonik. Requirements for science data bases and SciDB. In *Proc. CIDR Conference*, 2009.

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.

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.