Scalable and Numerically Stable Descriptive Statistics In SystemML

Yuanyuan Tian Shirish Tatikonda Berthold Reinwald
IBM Almaden Research Center, USA
{ytian, statiko, reinwald}@us.ibm.com

***Abstract*— There has been growing need for applying machine learning (ML) algorithms on very large datasets. SystemML is a declarative approach to scalable statistical ML. In SystemML, statistical ML algorithms are expressed as simple scripts in a high-level language. SystemML then complies and optimizes the scripts, and eventually translates them into efficient runtime on MapReduce. As the basis of virtually every quantitative analysis, *descriptive statistics* provide powerful tools to explore data in SystemML. This paper describes our experience in implementing descriptive statistics in SystemML. In particular, we elaborate on how to overcome the two major challenges: (1) numerical stability while operating on large datasets in the distributed setting of MapReduce; (2) efficient implementation of order statistics in MapReduce.**

I. Introduction

The growing need to analyze massive datasets has led to an increased interest in implementing ML algorithms on MapReduce. SystemML [7] is a large scale system for ML based on Apache Hadoop, developed in IBM research. In SystemML, statistical ML algorithms are expressed as simple scripts in a high-level language with linear algebra and mathematical primitives. SystemML then complies the scripts, applies various optimization techniques based on input data and system characteristics, and eventually translates them into efficient runtime on MapReduce. SystemML has been used for a large class of ML solutions, including classification, clustering, regression, matrix factorization, ranking etc.

Besides complex ML algorithms, SystemML also provides powerful tools to quantitively describe the basic features of data, called *descriptive statistics*. Descriptive statistics primarily include *univariate analysis* that deals with a single variable at a time, and *bivariate analysis* that examines the degree of association between two variables. They are the foundation of virtually every quantitative analysis of data. Table I lists the descriptive statistics currently supported in SystemML. In this paper, we describe our experience in addressing the two major challenges when implementing descriptive statistics in SystemML: (1) numerical stability while operating on large datasets in the distributed setting of MapReduce; (2) efficient implementation of order statistics in MapReduce.

II. Numerical Stability

Most of descriptive statistics, except for order statistics, can be expressed in certain summation form, thus may appear simple to be implemented in MapReduce. However, straightforward implementations can sometimes lead to disasters in numerical accuracy, due to round-off and truncation errors associated with finite precision arithmetic. As the need for analyzing large volumes of data increases, *numerical stability* becomes even more critical. In the MapReduce environment, one needs effective algorithms that can both exploit massive data parallelism and produce robust results with growing data sizes. To the best of our knowledge, none of the existing large scale platforms, such as PIG [10], HIVE [11] and JAQL [5], pay any attention to numerical stability[1]. Numerically unstable algorithms are still in use, even in well-known distributed database products [2]. In this section, we use summation and central moment as the driving examples to highlight the common pitfalls in implementing descriptive statistics, and describe how numerical stability is achieved in SystemML.

Univariate	Sum, Mean, Harmonic mean, Geometric mean, Mode, Min, Max, Range, Median, Quantiles, Inter-quartile mean, Variance, Standard deviation, Coefficient of variation, Central moment, Skewness, Kurtosis, Standard error of mean, Standard error of skewness, Standard error of kurtosis
Bivariate	Pearson's R, Chi-squared coefficient, Cramer's V Eta, ANOVA F-measure, Spearman correlation

TABLE I
Descriptive statistics supported in SystemML

Stable Summation. Summation is a fundamental operation in many statistical functions, such as mean, variance, and norms. The simplest method is to perform *recurisve summation*, which initializes $sum = 0$ and incrementally updates sum. It however suffers from numerical inaccuracies even on a single computer. For instance, with 2-digit precision, naive summation of numbers $1.0, 0.04, 0.04, 0.04, 0.04, 0.04$ results in 1.0, whereas the exact answer is 1.2. This is because once the first element 1.0 is added to sum, adding 0.04 will have no effect on sum due to round-off error. A simple alternative strategy is to first sort the data in increasing order, and subsequently perform the recursive summation. While it produces the accurate result for the above example, it is only applicable for non-negative numbers, and more importantly, it requires an expensive sort. There exist a number of other algorithms for stable summation [8]. One notable technique is proposed by Kahan [9]. It is the recursive summation with a correction term to reduce the rounding errors. Algorithm 1 shows the incremental update rule for Kahan algorithm.

One can easily extend the Kahan algorithm to the MapRe-

[1]Some of these systems support BigDecimal type, but it incurs heavy performance overhead, thus is not recommended for large scale data processing.

duce setting. The resulting algorithm is a MapReduce job in which each mapper applies KahanIncrement and generates a partial sum with correction, and a single reducer produces the final sum. Through error analysis, we can derive that when each mapper processes $\leq 10^{16}$ data items, this algorithm is robust w.r.t. the total number of data items to be summed using IEEE doubles (proof omitted).

Algorithm 1 Kahan Summation Incremental Update

```
// $s_1$ and $s_2$ are partial sums, $c_1$ and $c_2$ are correction terms
KahanIncrement($s_1, c_1, s_2, c_2$){
  $corrected\_s_2 = s_2 + (c_1 + c_2)$
  $sum = s_1 + corrected\_s_2$
  $correction = corrected\_s_2 - (sum - s_1)$
  return $(sum, correction)$ }
```

Stable High Order Statistics. We now describe our stable algorithms to compute higher order statistics, such as variance, skewness and kurtosis. The core computation is to calculate the p^{th} central moment $m_p = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^p$. For instance, variance can be written as $s^2 = \frac{n}{n-1}m_2$ and skewness can be expressed as $g_1 = \frac{m_3}{m_2^{1.5}}$. The standard two-pass algorithm produces the numerically stable result but it requires two scans of data – one scan to compute $\bar{x}$ and the second to compute m_p. A common technique (pitfall) is to apply a simple textbook rewrite to get a one-pass algorithm. For instance, m_2 can be rewritten as $\frac{1}{n}\sum_{i=1}^{n} x_i^2 - \frac{1}{n^2}(\sum_{i=1}^{n} x_i)^2$. The sum and the sum of squares can be computed in a single pass. However, this algebraically equivalent rewrite is known to suffer from serious stability issues resulting from cancellation errors when performing substraction of two large and nearly equal numbers. This rewrite, in fact, may produce a negative result for m_2. Surprisingly, this notoriously unstable algorithm is still used in popular distributed database platforms [2] – emphasizing the need for immediate attention to numerical stability issues in large scale data processing.

In SystemML, we use a stable MapReduce algorithm based on an existing technique [4] to compute arbitrary order central moment in an incremental fashion. It makes use of an update rule (shown below) that combines partial results obtained from two disjoint subsets of data (denoted as subscripts a and b). Here, n, μ, M_p refer to the cardinality, mean, and $n \times m_p$ of a subset, respectively. Note that the same update rule is used in each mapper that maintains running values for these variables seen thus far, as well as in the (single) reducer that combines partial results computed by mappers.

$$\begin{aligned} n =& n_a + n_b, \;\; \delta = \mu_b - \mu_a, \;\; \mu = \mu_a + n_b\frac{\delta}{n} \\ M_p =& M_{p,a} + M_{p,b} + \sum_{j=1}^{p-2}\binom{p}{j}[(-\frac{n_b}{n})^j M_{p-j,a} \\ &+ (\frac{n_a}{n})^j M_{p-j,b}]\delta^j + (\frac{n_a n_b}{n}\delta)^p[\frac{1}{n_b^{p-1}} - (\frac{-1}{n_a})^{p-1}] \end{aligned}$$

III. Order Statistics

Order statistics are one of the fundamental tools in non-parametric data analysis. They can be used to represent various statistics, such as *min*, *max*, *range*, *median*, *quantiles* and *inter-quartile mean*. Arbitrary order statistics from a set of n numbers can be computed in $O(n)$ time using the popular BFPRT algorithm [6]. There exist several efforts such as [3] that attempt to parallelize the sequential BFPRT algorithm. The central idea is to determine the required order statistic *iteratively* by *dynamically redistributing* the data in each iteration among different cluster nodes to achieve load balance. These iterative algorithms are suited for MPI-style parallelization but they are not readily applicable to MapReduce. This is because each iteration requires multiple MapReduce jobs and accordingly multiple disk reads and writes, and message passing between cluster nodes can only be done through the heavy-weight shuffling mechanism. Therefore, in SystemML, we devise a sort-based algorithm by leveraging Hadoop's inherent capability to sort large set of numbers. This algorithm only incurs one full MapReduce job to sort the input data plus one partial scan of the sorted data.

Sort-Based Order Statistics Algorithm. This algorithm consists of three phases. The first two phases are inspired by Yahoo's TeraSort algorithm [1].

Sample Phase: This phase samples N_s items from the input data, which are then sorted and divided evenly into r partitions, where r is the number of reducers. The boundary points are used for range partitioning in the next phase.

Sort Phase: This phase is a MapReduce job that reads the input data, and uses the results from previous phase to range partition the data. Each range is assigned to a different reducer, which sorts the unique values and keeps the number of occurrences for each unique value. Each reducer also computes the total number of data items in its range.

Selection Phase: For a given set of required order statistics, the output of sort phase (r sorted HDFS files and number of items in each file) is used to identify the appropriate file(s) that need to be scanned. If multiple order statistics reside in a single file, the scan cost can be shared. Furthermore, different files are scanned concurrently for improved efficiency.

References

[1] http://sortbenchmark.org/Yahoo2009.pdf.

[2] Teradata RDBMS SQL Reference – Functions and Operators. http://www.teradataforum.com/teradata_pdf/b035-1101-122a_5.pdf.

[3] D. Bader. An improved, randomized algorithm for parallel selection with an experimental study. *JPDC*, 64(9), 2004.

[4] J. Bennett et al. Numerically stable, single-pass, parallel statistics algorithms. In *Cluster*, 2009.

[5] K. Beyer et al. JAQL: A Scripting Language for Large Scale Semistructured Data Analysis. In *VLDB*, 2011.

[6] M. Blum et al. Time bounds for selection. *JCSS*, 7(4), 1973.

[7] A. Ghoting et al. SystemML: Declarative Machine Learning on MapReduce. In *ICDE*, 2011.

[8] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, 2nd edition, 2002.

[9] W. Kahan. Further remarks on reducing truncation errors. *Comm. ACM*, 8(1):40, 1965.

[10] C. Olston et al. Pig latin: A not-so-foreign language for data processing. In *SIGMOD*, 2008.

[11] A. Thusoo et al. Hive - a petabyte scale data warehouse using hadoop. In *ICDE*, 2010.