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—**With the exponential growth in the amount of data that is being generated in recent years, there is a pressing need for applying machine learning algorithms to large data sets. *SystemML* is a framework that employs a declarative approach for large scale data analytics. In SystemML, machine learning algorithms are expressed as scripts in a high-level language, called DML, which is syntactically similar to R. DML scripts are compiled, optimized, and executed in the SystemML runtime that is built on top of MapReduce.**

As the basis of virtually every quantitative analysis, *descriptive statistics* provide powerful tools to explore data in SystemML. In this paper, we describe our experience in implementing descriptive statistics in SystemML. In particular, we elaborate on how to overcome the two major challenges: (1) achieving numerical stability while operating on large data sets in a distributed setting of MapReduce; and (2) designing scalable algorithms to compute order statistics in MapReduce. By empirically comparing to algorithms commonly used in existing tools and systems, we demonstrate the advantage of SystemML in achieving numerical accuracy. We also highlight the valuable lessons we have learned through the exercise of implementing scalable and numerically stable descriptive statistics.

I. Introduction

The growing need to analyze massive datasets has led to an increased interest in implementing machine learning algorithms on MapReduce [10], [13]. SystemML [12] is an Apache Hadoop based system for large scale machine learning, developed at IBM Research. In SystemML, machine learning algorithms are expressed as scripts written in a high-level language, called DML, with linear algebra and mathematical primitives. SystemML compiles these scripts, applies various optimization techniques based on data and system characteristics, and translates them into efficient runtime on MapReduce. A wide variety of ML techniques can be expressed in SystemML, including classification, clustering, regression, matrix factorization, and ranking.

Besides complex machine learning algorithms, SystemML also provides powerful constructs to compute *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. Table I lists the descriptive statistics that are 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.

TABLE I
DESCRIPTIVE STATISTICS SUPPORTED IN SYSTEMML

Univariate	**Scale variable:** Sum, Mean, Harmonic mean, Geometric mean, 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
	Categorical variable: Mode, Per-category frequencies
Bivariate	**Scale-Scale variables:** Covariance, Pearson correlation
	Scale-Categorical variables: Eta, ANOVA F measure
	Categorical-Categorical variables: Chi-squared coefficient, Cramer's V, Spearman correlation

Most of descriptive statistics, except for order statistics, can be expressed in certain summation form, and hence it may seem as if they are trivial to implement on MapReduce. However, straightforward implementations often lead to disasters in numerical accuracy, due to overflow, underflow and round-off errors associated with finite precision arithmetic. Such errors often get magnified with the increasing volumes of data that is being processed. However in practice, the issue of numerical stability is largely ignored, especially in the context of large scale data processing. Several well-known and commonly used software products still use numerically unstable implementations to compute several basic statistics. For example, McCullough and Heiser highlighted in a series of articles that Microsoft Excel suffers from numerical inaccuracies for various statistical procedures [16], [17], [18], [19]. In their studies, they conducted various tests related to univariate statistics, regression, and Monte Carlo simulations using the NIST Statistical Reference Datasets (StRD) [3]. They note that numerical improvements were made to univariate statistics only in the recent versions of Excel [16]. In the context of large-scale data processing, Hadoop-based systems such as PIG [21] and HIVE [22] still use numerically unstable implementations to compute several statistics. PIG as of version 0.9.1 supports only the very basic *sum* and *mean* functions, and neither of them use numerically stable algorithms. The recent version of HIVE (0.7.1) supports more statistics including *sum*, *mean*, *variance*, *standard deviation*, *covariance*, and *Pearson correlation*. While *variance*, *standard deviation*, *covariance*, and *Pearson correlation* are computed using stable methods, both *sum* and *mean* are computed using numerically unstable methods. These examples highlight the fact that the issue of numeric stability has largely been ignored in practice, in spite of its utmost importance.

In this paper, we share our experience in achieving numerical stability as well as scalability for descriptive statistics on MapReduce, and bring the community's attention to the important issue of numerical stability for large scale data processing. Through a detailed set of experiments, we demonstrate the scalability and numerical accuracy achieved by SystemML. We finally conclude by highlighting the lessons we have learned in this exercise.

II. Numerical Stability

Numerical stability refers to the inaccuracies in computation resulting from finite precision floating point arithmetic on digital computers with round-off and truncation errors. Multiple algebraically equivalent formulations of the same numerical calculation often produce very different results. While some methods magnify these errors, others are more robust or stable. The exact nature and the magnitude of these errors depend on several different factors like the number of bits used to represent floating point numbers in the underlying architecture (commonly known as *precision*), the type of computation that is being performed, and also on the number of times a particular operation is performed. Such round-off and truncation errors typically grow with the input data size. Given the exponential growth in the amount of data that is being collected and processed in recent years, numerical stability becomes an important issue for many practical applications.

One possible strategy to alleviate these errors is to use special software packages that can represent floating point values with *arbitrary* precision. For example, BigDecimal in Java provides the capability to represent as well as operate on arbitrary-precision signed decimal numbers. Each BigDecimal number is a Java object. While the operations like addition and subtraction on native data types (double, int) are performed on hardware, the operations on BigDecimal are implemented in software. Furthermore, JVM has to explicitly manage the memory occupied by BigDecimal objects. Therefore, these software packages often suffer from significant performance overhead. In our benchmark studies, we observed up to 2 orders of magnitude slowdown for addition, subtraction, and multiplication using BigDecimal with precision 1000 compared to native *double* data type; and up to 5 orders of magnitude slowdown for division.

In the rest of this section, we discuss methods adopted in SystemML to calculate descriptive statistics, which are both numerically stable and computationally efficient. We will also highlight the common pitfalls that must be avoided in practice. First, we discuss the fundamental operations *summation* and *mean*, and subsequently present the methods that we use for computing higher-order statistics and *covariance*.

A. Stable Summation

Summation is a fundamental operation in many statistical functions, such as mean, variance, and norms. The simplest method is to perform *naive recursive 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. Both PIG and HIVE use this naive recursive algorithm to compute sum. A simple alternative strategy is to first sort the data in increasing order, and subsequently perform the naive 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 exists a number of other methods for stable summation [14]. One notable technique is proposed by Kahan [15]. It is a *compensated summation* technique – see Algorithm 1. This method maintains a *correction* or *compensation* term to accumulate errors encountered in naive recursive summation.

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 $(sum, correction)$ }
```

Kahan and Knuth independently proved that Algorithm 1 has the following relative error bound [14]:

$$\frac{|E_n|}{|S_n|} = \frac{|\hat{S}_n - S_n|}{|S_n|} \leq (2u + O(nu^2))\kappa_X, \qquad (1)$$

where $S_n = \sum_{i=1}^{n} x_i$ denotes the true sum of a set of n numbers $X = \{x_1, x_2, ..., x_n\}$, and $\hat{S}_n$ is the sum produced by the summation algorithm, $u = \frac{1}{2}\beta^{1-t}$ is the *unit roundoff* for a floating point system with base β and precision t. It denotes the upper bound on the relative error due to rounding. For IEEE 754 floating point standard with $\beta = 2$ and $t = 53$, $u = 2^{-53} \approx 10^{-16}$. Finally, κ_X is known as the *condition number* for the summation problem, and it is defined as the fraction $\frac{\sum_{i=1}^{n} |x_i|}{|\sum_{i=1}^{n} x_i|}$. The condition number measures the sensitivity of the problem to approximation errors, independent of the exact algorithm used. Higher the value of κ_X, the higher will be the numerical inaccuracies and the relative error. It can be shown that for a random set of numbers with a nonzero mean, the condition number of summation asymptotically approaches to a finite constant as $n \to \infty$. Evidently, the condition number is equal to 1 when all the input values are non-negative. It can be seen from Equation 1 that when $nu \leq 1$, the bound on the relative error is independent of input size n. In the context of IEEE 754 standard, it means that when n is in the order of 10^{16} the relative error can be bounded independent of the problem size. In comparison, the naive recursive summation has a relative error bound $\frac{|E_n|}{|S_n|} \leq (n-1)u\kappa_X + O(u^2)$ [14], which clearly is a much larger upper bound when compared to Equation 1.

One can easily extend the Kahan algorithm to the MapReduce 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 the upper bound on the relative error for this MapReduce Kahan summation algorithm: $\frac{|E_n|}{|S_n|} \leq [4u+4u^2+O(mu^2)+O(\frac{n}{m}u^2)+O(mu^3)+O(\frac{n}{m}u^3)+O(nu^4)]\kappa_X$ (see the Appendix for the proof). Here, m is the number of mappers (m is at most in 1000s). As long as $\frac{n}{m}u \leq 1$ (and $m \ll n$), the relative error is independent of the number of input data items. In the context of IEEE 754 standard, it can be shown that when $\frac{n}{m}$ is in the order of $2^{53} \approx 10^{16}$, the relative error can be bounded independent of n. In other words, as long as the number of elements processed by each mapper is in the order of 10^{16}, the overall summation is robust with respect to the total number of data items to be summed. Therefore, by partitioning the work across multiple mappers, the MapReduce summation method is able to scale to larger data sets while keeping the upper bound on relative error independent of the input data size n.

B. Stable Mean

Mean is a fundamental operation for any quantitative data analysis. The widely used technique is to divde the *sum* by the total number of input elements. Both PIG and HIVE relies on this approach, where *sum* is computed using the naive recursive summation algorithm. This straightforward method of computing mean however suffers from numerical instability. As the number of data points increases, the accuracy of *sum* decreases, thereby affecting the quality of *mean*. Even when the stable summation is used, *sum* divided by *count* technique often results in less accurate results.

In SystemML, we employ an incremental approach to compute the *mean*. This method maintains a running mean of the elements processed so far. It makes use of the update rule in Equation 2. In a MapReduce setting, all mappers apply this update rule to compute the partial values for *count* and *mean*. These partial values are then aggregated in the single reducer to produce the final value of *mean*.

$$n = n_a + n_b,\ \ \delta = \mu_b - \mu_a,\ \ \mu = \mu_a \oplus n_b\frac{\delta}{n} \tag{2}$$

In Equation 2, n_a and n_b denote the partial counts, μ_a and μ_b refer to partial means. The combined mean is denoted by μ, and it is computed using the KAHANINCREMENT function in Algorithm 1, denoted as $\oplus$ in the equation. In other words, we keep a correction term for the running mean, and $\mu = \mu_a \oplus n_b\frac{\delta}{n}$ is calculated via $(\mu.value, \mu.correction) =$ KAHANINCREMENT $(\mu_a.value, \mu_a.correction, n_b\frac{\delta}{n}, 0)$. Note that the use of KAHANINCREMENT is important to obtain stable value for μ. When the value of $n_b\frac{\delta}{n}$ is much smaller than μ_a, the resulting μ will incur a loss in accuracy – which is alleviated by using KAHANINCREMENT. As we show later in Section IV, this incremental algorithm results in more accurate results. We also use it in stable algorithms to compute higher order statistics and covariance (see Sections II-C & II-D).

C. Stable Higher-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 $m_p = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^p$. Central moment can be used to describe higher-order statistics. For instance, variance can be written as $\sigma^2 = \frac{n}{n-1}m_2$, skewness $g_1 = \frac{m_3}{m_2^{1.5}} \cdot (\frac{n-1}{n})^{1.5}$, and kurtosis $g_2 = \frac{m_4}{m_2^2} \cdot (\frac{n-1}{n})^2 - 3$. The standard two-pass algorithm produces relatively stable results (for m_2, the relative error bound is $nu + n^2u^2\kappa_X^2$, where $\kappa_X = (\frac{\sum_{i=1}^{n} x_i^2}{\sum_{i=1}^{n}(x_i-\bar{x})^2})^{\frac{1}{2}}$ is the condition number), 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 of squares and sum can be computed in a single pass. However, this algebraical rewrite is known to suffer from serious stability issues resulting from cancellation errors when performing substraction of two large and nearly equal numbers (relative error bound is $nu\kappa_X^2$). This rewrite, as we show later in Section IV-B, may actually produce a negative result for m_2, thereby resulting in grossly erroneous values for variance and other higher-order statistics.

In SystemML, we use a stable MapReduce algorithm based on an existing technique [6] 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. Again, $\oplus$ represents addition through KAHANINCREMENT. When $p = 2$, the algorithm has a relative error bound of $nu\kappa_X$. While there is no formal proof to bound the relative error when $p > 2$, we empirically observed that this method produces reasonably stable results for skewness and kurtosis. In the MapReduce setting, 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. Note that HIVE utilizes a similar update rule to compute variance and standard deviation. However, HIVE uses the basic addition for the updates instead of the more stable KAHANINCREMENT. In Section IV-D, we will evaluate the effect of KAHANINCREMENT by comparing the algorithms used in SystemML and HIVE for variance and standard deviation.

$$n = n_a + n_b,\ \ \delta = \mu_b - \mu_a,\ \ \mu = \mu_a \oplus n_b\frac{\delta}{n}$$

$$M_p = M_{p,a} \oplus M_{p,b} \oplus \{\sum_{j=1}^{p-2}\binom{p}{j}[(-\frac{n_b}{n})^j M_{p-j,a} + (\frac{n_a}{n})^j M_{p-j,b}]\delta^j + (\frac{n_a n_b}{n}\delta)^p[\frac{1}{n_b^{p-1}} - (\frac{-1}{n_a})^{p-1}]\}$$

D. Stable Covariance

Covariance ($\frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{n-1}$) is one of the basic bivariate statistic. It measures the strength of correlation between two datasets. A common textbook one-pass technique rewrites the equation as $\frac{1}{n-1}\sum_{i=1}^{n} x_i y_i - \frac{1}{n(n-1)}\sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i$. Similar to the one-pass rewrite of variance described in Section II-C, this rewrite also suffers from numerical errors and can produce grossly inaccurate results.

In SystemML, we adapt the technique proposed by Bennett *et al.* [6] to a MapReduce setting. This algorithm computes the partial aggregates in the mappers and combines the partial aggregates in a single reducer. The update rule for computing $C = \sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$ are shown below. Note that $\oplus$ represents addition through KAHANINCREMENT. The similar update rule is also used in HIVE for computing covariance, but again HIVE only applies basic addition instead of the more stable KAHANINCREMENT. Later in Section IV-D, we will compare SystemML against HIVE for covariance and Pearson correlation.

$$\begin{aligned} n =& n_a + n_b, \;\; \delta_x = \mu_{x,b} - \mu_{x,a}, \;\; \mu_x = \mu_{x,a} \oplus n_b \frac{\delta_x}{n} \\ \delta_y =& \mu_{y,b} - \mu_{y,a}, \;\; \mu_y = \mu_{y,a} \oplus n_b \frac{\delta_y}{n} \\ C =& C_a \oplus C_b \oplus \frac{n_a n_b}{n} \delta_x \delta_y \end{aligned}$$

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*. Although order statistics do not suffer from numerical stability problems, how to efficiently compute them in MapReduce is a challenge.

Arbitrary order statistics from a set of n numbers can be computed in $O(n)$ time using the popular BFPRT algorithm [7]. There exist several efforts such as [5] that attempt to parallelize the sequential BFPRT algorithm. The core idea is to determine the required order statistic *iteratively* by *dynamically redistributing* the data in each iteration among different cluster nodes to achieve load balancing. These iterative algorithms are suited for MPI-style parallelization but they are not readily applicable to MapReduce, as they require multiple MapReduce jobs with multiple disk reads and writes. Furthermore, message passing in MapReduce can only be achieved through the heavy-weight shuffling mechanism. In SystemML, we therefore devised a sort-based algorithm by leveraging Hadoop's inherent capability to sort large set of numbers. This algorithm needs only one full MapReduce job to sort the input data, plus one partial scan of the sorted data to compute the required statistics. Note that there have also been studies on parallel approximate order statistics algorithms [9], but in this paper, we focus only on algorithms for exact order statistics.

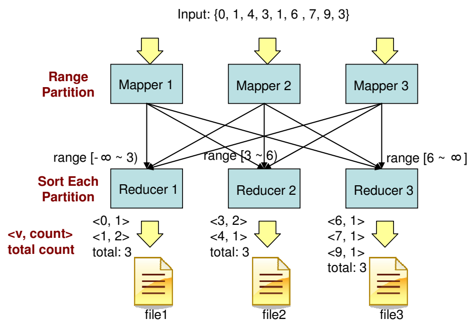

Fig. 1. Sort-Based Order Statistics Algorithm

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: As shown in Figure 1, 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.

IV. EXPERIMENTS

A. Experimental Setup

Experiment Cluster: The experiments were conducted with Hadoop 0.20 [2] on a cluster with 5 machines as worker nodes. Each machine has 8 cores with hyperthreading enabled, 32 GB RAM and 2 TB storage. We set each machine to run 15 concurrent mappers and 10 concurrent reducers.

Experiment Datasets: Although there are existing benchmark datasets to assess the accuracy of numerical algorithms and software, such as the NIST StRD Benchmark [3], these benchmarks mostly provide very small datesets. For example, the largest dataset in NIST StRD contains only 5000 data points. To test the numerical stability of distributed algorithms, we generate large synthetic datasets similar to the NIST StRD benchmark datasets for univariate summary statistics. More specifically, for the given number of data entries and the value range, the data generator creates the required number of random values uniformly distributed within the value range. For our experiments, we create datasets with varying sizes in the following 3 different ranges: R1:[1.0 - 1.5), R2:[1000.0 - 1000.5) and R3:[1000000.0 - 1000000.5).

Accuracy Measurement: In order to assess the numerical accuracy of the results produced by any algorithm, we need the true values of statistics. For this purpose, we rely on Java $\mathrm{BigDecimal}$ that can represent arbitrary-precision signed decimal numbers. We implemented the naive algorithms for various statistics using Java $\mathrm{BigDecimal}$ with precision 1000. More specifically, we implemented naive recursive summation for sum, sum divided by count for $mean$, one-pass algorithms for higher-order statistics as shown in Table II, and textbook one-pass algorithm for $covariance$. We consider obtained results as the "true values" of these statistics.

To quantitively measure accuracy, we use the Log Relative Error (LRE) metric described in [20]. If q is the estimated value from an algorithm, and t is the true value, then LRE is defined as

$$LRE = -\log_{10} |\frac{q-t}{t}|$$

LRE captures the number of correct significant digits of the estimated value q, thus the higher a LRE is, the more desirable the estimated value is.

TABLE II
TEXTBOOK ONE-PASS ALGORITHMS FOR HIGHER-ORDER STATISTICS

	Equations for 1-Pass Algorithm
variance	$\frac{1}{n-1}S_2 - \frac{1}{n(n-1)}S_1^2$
std	$(variance)^{\frac{1}{2}}$
skewness	$\frac{S_3 - \frac{3}{n}S_1 S_2 + \frac{2}{n^2}S_1^3}{n \times std \times variance}$
kurtosis	$\frac{S_4 - \frac{4}{n}S_3 S_1 + \frac{6}{n^2}S_2 S_1^2 - \frac{3}{n^3}S_1^4}{n \times (variance)^2} - 3$

$S_p = \sum_{i=1}^{n} x_i^p$, which can be easily computed in one pass.

B. Numerical Stability of Univariate Statistics

In this section, we empirically compare the numerical stabilities of different algorithms for a subset of univariate statistics.

Table III lists the accuracy (LRE values) of results produced by different summation algorithms and mean algorithms. For summation, we compare the MapReduce Kahan summation used in SystemML to the naive recursive summation and the sorted summation. The latter two algorithms are both adapted to the MapReduce environment. The naive recursive summation is simply adapted to compute partial recursive summation in each mapper and finally aggregates in the reducer. This adapted naive recursive summation is utilized in both PIG and HIVE for computing sum. For the sorted summation, we first apply the sorting algorithm described in Section III using MapReduce, then apply the adapted recursive summation on the sorted data. As shown in Table III, SystemML consistently produces more accurate results than the other two. The naive recursive summation and the sorted summation are neck and neck in terms of accuracy. In addition, the performance difference between the MapReduce Kahan summation and the naive recursive summation is negligible, whereas the sorted summation is almost 5 times slower due to the sort. We also compare the accuracy of mean algorithms in Table III. Again, the mean algorithm in SystemML consistently produces more accurate results than the naive approach (naive recursive sum divided by count).

The accuracy comparison of higher-order statistics are shown in Table IV. In SystemML, we employ the algorithm described in Section II-C to compute required central moments which are then used to calculate the corresponding higher-order statistics. As comparisons are the textbook one-pass algorithms to compute these higher-order statistics. Table II describes the equations for computing all the higher-order statistics in one pass. It is clearly shown in Table IV that SystemML algorithms are significantly more stable than the textbook one-pass algorithms. The advantage of SystemML is much higher in the higher-order statistics than sum and mean, because errors get more amplified as the order increases. For some datasets in ranges R2 and R3, the higher order statistics are completely wrong (0 digits matched). In particular, the R3 100million dataset and R3 1billion dataset produce negative variance, which leads to undefined standard deviation, skewness and kurtosis (shown as NA in Table IV).

Now let's study the effect of the 3 different value ranges on the accuracy of univariate statistics. Although the delta (difference between min and max values) of each range is the same (0.5), all algorithms including stable ones present decreasing accuracy from R1, R2 to R3. This is because the 3 ranges have increasing means. If we know ahead of time some basic characteristics of a dataset, we can simply shift the data before computing the statistics, then add the shifting effect back to the results. For example, it has been shown in [8] that to compute variance for a dataset with a large mean, substantial gains in accuracy can be achieved by shifting all the data by some approximation of the mean. Similarly, shifting can be applied to the other univariate statistics for increased accuracy.

C. Numerical Stability of Bivariate Statistics

We now discuss the numerical accuracies achieved by SystemML for bivariate statistics. We consider two types of statistics: *scale-categorical* and *scale-scale*. In the former type, we compute *Eta*[1] and *ANOVA-F*[2] measures, whereas in the latter case, we compute *Covariance* and *Pearson Correlation (R)*[3]. For scale variables, we use data sets in value ranges R1, R2 and R3 that were described in Section IV-A. For categorical variables, we generated data in which 50 different categories are uniformly distributed.

In order to compute these statistics, SystemML makes use the numerically stable methods for *sum*, *mean*, *variance* and

[1]Eta is defined as $(1 - \frac{\sum_{r=1}^{R}(n_r-1)\sigma_r^2}{(n-1)\sigma^2})^{\frac{1}{2}}$, where R is the number of categories, n_r is the number of data entries per category, σ_r^2 is the variance per category, n is the total number of data entries, and σ^2 is the total variance.

[2]ANOVA-F is defined as $\frac{\sum_{r=1}^{R} n_r(\mu_r-\mu)^2}{\sum_{r=1}^{R}(n_r-1)\sigma_r^2} \cdot \frac{n-R}{R-1}$, where R is the number of categories, n_r is the number of data entries per category, μ_r is the mean per category, σ_r^2 is the variance per category, n is the total number of data entries, and μ is the total mean.

[3]Pearson-R is defined as $\frac{\sigma_{xy}^2}{\sigma_x \sigma_y}$, where σ_{xy}^2 is the covariance, σ_x and σ_y are standard deviations.

TABLE III
NUMERICAL ACCURACY OF SUM AND MEAN (LRE VALUES)

	Size (million)	**Sum**			**Mean**	
Range		SystemML	Naive	Sorted	SystemML	Naive
R1	10	16.1	13.5	16.1	16.7	13.5
	100	16.3	13.8	13.6	16.2	13.8
	1000	16.8	13.6	13.5	16.5	13.6
R2	10	16.8	14.4	13.9	16.5	14.4
	100	16.1	13.4	13.4	16.9	13.4
	1000	16.6	13.1	13.9	16.4	13.1
R3	10	15.9	14.0	13.9	16.3	14.0
	100	16.0	13.1	13.4	16.9	13.1
	1000	16.3	12.9	12.2	16.5	12.9

TABLE IV
NUMERICAL ACCURACY OF HIGHER-ORDER STATISTICS (LRE VALUES)

	Size (million)	**Variance**		**Std**		**Skewness**		**Kurtosis**	
Range		SystemML	Naive	SystemML	Naive	SystemML	Naive	SystemML	Naive
R1	10	16.0	11.3	15.9	11.6	16.4	7.5	15.3	9.8
	100	16.2	11.5	16.8	11.8	14.9	7.1	15.6	9.3
	1000	16.0	11.3	16.4	11.6	14.5	6.5	15.6	8.9
R2	10	15.4	5.9	15.9	6.2	12.5	0	14.9	0
	100	15.6	5.3	15.8	5.6	12.0	0	14.9	0
	1000	16.2	4.9	16.4	5.2	12.1	0	15.2	0
R3	10	14.4	0	14.7	0	9.1	0	12.6	0
	100	12.9	0	13.2	NA	9.0	NA	13.2	NA
	1000	13.2	0	13.5	NA	9.4	NA	12.9	NA

NA represents undefined standard deviation, skewness or kurtosis due to a negative variance value.

covariance from Section II, whereas $Naive$ in comparison utilizes the naive recursive summation for sum, sum divided by count for mean, and textbook one-pass algorithms for variance and covariance.

The LRE values obtained for scale-categorical statistics are shown in Table V. From the table, it is evident that the statistics computed in SystemML have higher accuracy than the ones from $Naive$. It can also be observed that the accuracy achieved by both the methods reduces as the mean of input values increases – e.g., LRE numbers for R3 are smaller than those of R1. As we move from R1 to R3, the accuracy of $Naive$ drops much more steeply than SystemML. This is due to the inaccuracies in per-category and total sum and $variance$ quickly propagate and magnify the errors in Eta and ANOVA-F. This emphasizes that the use of inaccurate values in computations can lead to grossly erroneous results. Similar trends can be observed in case of covariance and Pearson correlation, as shown in Table VI. For the cases of R2 vs. R3 with 100million and 1billion datasets, $Naive$ algorithm produces negative variance values for R3 (see Table IV), which results in undefined values for Pearson-R (shown as NA in Table VI).

D. Comparison between SystemML and HIVE

The latest version of HIVE (0.7.1) provides a number descriptive statistics including *sum*, *mean*, *variance*, *standard deviation*, *covariance*, and *pearson correlation*. We now compare the accuracies achieved by SystemML against those of HIVE, for several statistics. Since HIVE (as well as PIG)

TABLE V
NUMERICAL ACCURACY OF BIVARIATE SCALE-CATEGORICAL STATISTICS: ETA AND ANOVA-F (LRE VALUES)

	Size (million)	**Eta**		**ANOVA-F**	
Range		SystemML	Naive	SystemML	Naive
R1	10	16.2	13.7	16.2	10.0
	100	16.6	13.7	15.6	10.0
	1000	16.5	13.6	15.8	9.9
R2	10	16.2	7.2	13.3	3.5
	100	16.6	7.4	13.4	3.7
	1000	16.5	7.9	13.4	4.3
R3	10	16.2	0	10.2	0
	100	15.9	1.9	10.0	0
	1000	16.5	1.2	10.0	0

computes *sum* and *mean* using the naive recursive summation algorithm, the comparison between SystemML and HIVE on these two statistics will be same as the results presented in Table III.

For higher order statistics, covariance, and Pearson correlation, both SystemML and HIVE employ the same incremental update rules, as shown in Sections II-C & II-D. The primary difference is that SystemML makes use of KAHANINCREMENT to perform aggregations, whereas HIVE relies on the basic addition. The differences in accuracy achieved by SystemML and HIVE are shown in Table VII and Table VIII. Evidently, SystemML is able to produce more accurate results than HIVE for all statistics across the board. In SystemML, the correction terms maintained in Kahan technique helps in

TABLE VI
NUMERICAL ACCURACY OF BIVARIATE SCALE-SCALE STATISTICS: COVARIANCE AND PEARSON-R (LRE VALUES)

	Size (million)	**Covariance**		**Pearson-R**	
Range		SystemML	Naive	SystemML	Naive
R1 vs. R2	10	15.0	8.4	15.1	6.2
	100	15.6	8.5	15.4	6.4
	1000	16.0	8.7	15.7	6.2
R2 vs. R3	10	13.5	3.0	13.5	3.0
	100	12.8	2.8	12.7	NA
	1000	13.6	3.9	13.8	NA

NA represents undefined pearson-R due to a negative variance value.

reducing the effect of truncation errors.

TABLE VII
ACCURACY COMPARISION BETWEEN SYSTEMML AND HIVE FOR VARIANCE AND STANDARD DEVIATION (LRE VALUES)

	Size (million)	**Variance**		**Std**	
Range		SystemML	HIVE	SystemML	HIVE
R1	10	16.0	13.5	15.9	13.8
	100	16.2	13.7	16.8	14.0
	1000	16.0	14.1	16.4	14.4
R2	10	15.4	12.8	15.9	13.1
	100	15.6	12.5	15.8	12.8
	1000	16.2	13.8	16.4	14.1
R3	10	14.4	9.3	14.7	9.6
	100	12.9	9.5	13.2	9.8
	1000	13.2	10.3	13.5	10.6

E. Performance of Order Statistics

In this section, we evaluate the performance of the sort-based order statistics algorithm proposed in Section III. Script 1 shows a simple script of order statistics written in SystemML (refer to [12] for detailed language syntax of SystemML). It computes the median and some quantiles of a vector V. In this experiment, we fix the specified quantiles to be $P = \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$, and vary the size of V (the value range of data in V is R1). Figure 2 shows the execution times of this simple script with increasing sizes of V. Notice that although multiple order statistics are computed in this script, they all share the same sort. Furthermore, the required order statistics are all selected concurrently in parallel.

Script 1: A Simple Script of Order Statistics

```
# input vector (column matrix)
V = read("in/V");
# a vector specifying the desired quantiles
P = read("in/P");
# compute median
median = quantile(V, 0.5);
print("median: ", median);
# compute quantiles
Q = quantile(V, P);
write(Q, "out/Q");
```

V. DISCUSSION

We now summarize the lessons learned while implementing scalable and numerically stable descriptive statistics in SystemML.

TABLE VIII
ACCURACY COMPARISION BETWEEN SYSTEMML AND HIVE FOR COVARIANCE AND PEARSON-R (LRE VALUES)

	Size (million)	**Covariance**		**Pearson-R**	
Range		SystemML	HIVE	SystemML	HIVE
R1 vs. R2	10	15.0	14.2	15.1	13.0
	100	15.6	13.3	15.4	13.0
	1000	16.0	14.6	15.7	14.2
R2 vs. R3	10	13.5	10.0	13.5	9.8
	100	12.8	10.0	12.7	10.5
	1000	13.6	11.4	13.8	10.5

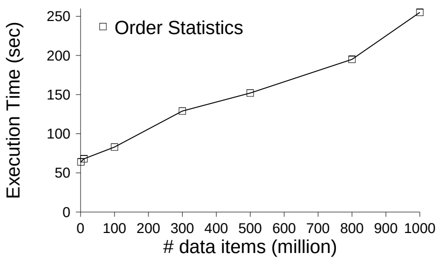

Fig. 2. Execution Time for Script 1

• **Many existing sequential techniques for numerical stability can be adapted to the distributed environment.**
We successfully adapted the existing sequential stable algorithms for summation, mean, central moments and covariance to mapreduce environments. Such adaptations are empirically shown to exhibit better numeric stability when compared to naive commonly used algorithms.

• **Performance doesn't have to be sacrificed for accuracy.**
While software packages like BigDecimal can be used to improve the numerical accuracy of computations, they incur significant performance overheads – we observed up to 5 orders of magnitude slowdown depending on the exact operation and the precision used. Similarly, the *sorted sum* technique that is widely used in sequential environments is not able to achieve similar accuracies when adapted to a distributed environment. Furthermore, its performance is hindered by the fact that the entire data has be sorted afront. In contrast, our stable algorithms for summation, mean, covariance, and higher order statistics achive good accuracy without sacrificing the run time performance. They make a single scan over the input data, and achieve comparable performance with unstable one-pass algorithms.

• **Shifting can be used for improved accuracy.**
When all the values in the input data set are large, it is useful to shift the elements by an approximation of mean prior to computing any statistics. This preprocessing technique helps in reducing the truncation errors, and often achieves better numerical accuracy. In the absense of such preprocessing, as shown in Sections IV-B & IV-C, the

accuracy of statistics degrades as the magnitude of input data elements increases (e.g., as the value range changes from R1 to R3).

• Divide-and-conquer design helps in scaling to larger data sets while achieving good numerical accuracy.

All the stable algorithms discussed in this article operate in a divide-and-conquer paradigm, where partial results are computed in the mappers that are combined in the reducer. These algorithms partitions the work into smaller pieces, and hence they can scale to larger data sets while keeping the upper bound on the relative error independent of the input data size. For example, the sequential version of Kahan summation algorithm can guarantee that the upper bound on the relative error is independent of the input data size as long as the *total number* of elements is in the order of 10^{16}. In contrast, the MapReduce version of Kahan summation algorithm can provide similar guarantees as long as the data size processed by *each* mapper is in the order of 10^{16} – i.e., the parallel version of the algorithm can scale to larger data sets without having a significant impact on the error bound. Here, we assume that the number of mappers is bounded by a constant that is significantly smaller than the input data size. While there is no format proof of such a result for covariance and other higher order statistics, we expect the distributed algorithms for these statistics to scale better than their sequential counterparts.

• Kahan technique is useful beyond a simple summation.

The strategy of keeping the correction term, as in Kahan summation algorithm, helps in alleviating the effect of truncation errors. Beyond the simple summation, we found this technique to be useful in computing stable values for other univariate and bivariate statistics, including mean, variance, covariance, Eta, ANOVA-F etc.

Acknowledgment

We would like to thank Peter Haas for his valuable suggestions and consultation on this work. We would also like to thank the other members of the SystemML team: Douglas R Burdick, Amol Ghoting, Rajasekar Krishnamurthy, Edwin Pednault, Prithviraj Sen, Vikas Sindhwani, Shivakumar Vaithyanathan, and Tian Zhang.

References

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

[2] Apache hadoop. http://hadoop.apache.org/.

[3] Nist statistical reference datasets. http://www.itl.nist.gov/div898/strd.

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

[5] D. Bader. An improved, randomized algorithm for parallel selection with an experimental study. *Journal of Parallel and Distributed Computing*, 64(9):1051–1059, 2004.

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

[7] M. Blum, R. Floyd, V. Pratt, R. Rivest, and R. Tarjan. Time bounds for selection*. *Journal of Computer and System Sciences*, 7(4):448–461, 1973.

[8] T. F. Chan, G. H. Golub, and R. J. LeVeque. Tony f. chan; gene h. golub; randall j. leveque. *The American Statistician*, 37:242–247, 1983.

[9] S. Chaudhuri, T. Hagerup, and R. Raman. Approximate and exact deterministic parallel selection. *Mathematical Foundations of Computer Science*, pages 352–361, 1993.

[10] C. Chu, S. Kim, Y. Lin, Y. Yu, G. Bradski, A. Ng, and K. Olukotun. Map-reduce for machine learning on multicore. In *Advances in Neural Information Processing Systems 19: Proceedings of the 2006 Conference*, volume 19, page 281. The MIT Press, 2007.

[11] C.-T. Chu, S. K. Kim, Y.-A. Lin, Y. Yu, G. R. Bradski, A. Y. Ng, and K. Olukotun. Map-reduce for machine learning on multicore. In *NIPS*, pages 281–288, 2006.

[12] A. Ghoting, R. Krishnamurthy, E. Pednault, B. Reinwald, V. Sindhwani, S. Tatikonda, Y. Tian, and S. Vaithyanathan. Systemml: Declarative machine learning on mapreduce. In *Proceedings of the 2011 IEEE 27th International Conference on data Data Engineering (ICDE)*, pages 231–242. IEEE.

[13] D. Gillick, A. Faria, and J. DeNero. Mapreduce: Distributed computing for machine learning.

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

[15] W. Kahan. Further remarks on reducing truncation errors. *Communications of the ACM*, 8(1):40, 1965.

[16] B. McCullough and D. Heiser. On the accuracy of statistical procedures in microsoft excel 2007. *Computational Statistics & Data Analysis*, 52(10):4570–4578, 2008.

[17] B. McCullough and B. Wilson. On the accuracy of statistical procedures in microsoft excel 97. *Computational Statistics and Data Analysis*, 31(1):27–38, 1999.

[18] B. McCullough and B. Wilson. On the accuracy of statistical procedures in microsoft excel 2000 and excel xp. *Computational Statistics & Data Analysis*, 40(4):713–721, 2002.

[19] B. McCullough and B. Wilson. On the accuracy of statistical procedures in microsoft excel 2003. *Computational Statistics & Data Analysis*, 49(4):1244–1252, 2005.

[20] B. D. McCullough. Assessing the reliability of statistical software: Part i. *The American Statistician*, 52:358–366, 1998.

[21] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: a not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1099–1110. ACM, 2008.

[22] A. Thusoo, J. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Antony, H. Liu, and R. Murthy. Hive-a petabyte scale data warehouse using hadoop. In *Proceedings of the 2010 IEEE 26th International Conference on data Data Engineering (ICDE)*, pages 996–1005. IEEE, 2010.

Appendix A
Proof of Error Bound for MapReduce Kahan Summation Algorithm

The relative error bound for the MapReduce Kahan summation algorithm is $\frac{|E_n|}{|S_n|} \leq [4u + 4u^2 + O(mu^2) + O(\frac{n}{m}u^2) + O(mu^3) + O(\frac{n}{m}u^3) + O(nu^4)]\kappa_X$.

Proof: As each of the m mappers uses Kahan algorithm, the k-th mapper has the error bound $E_{n_k} = |\hat{S}_{n_k} - S_{n_k}| \leq (2u + O(n_k u^2)) \sum_{i=1}^{n_k} |x_{k,i}|$, where n_k is the number of data items process by this mapper. As n_k is in $O(\frac{n}{m})$, we can simply derive $E_{n_k} \leq (2u + O(\frac{n}{m}u^2)) \sum_{i=1}^{n_k} |x_{k,i}|$.

Each mapper generates $\hat{S}_{n_k}$ with its correction term. In the single reducer, we again run the Kahan sum algorithm on these

m data items, therefore, we get the following:

$$
\begin{aligned}
&|\hat{S}_n - \sum_{i=1}^{m} \hat{S}_{n_k}| \\
&\leq (2u + O(mu^2)) \sum_{k=1}^{m} |\hat{S}_{n_k}| \\
&\leq (2u + O(mu^2)) \sum_{k=1}^{m} (|S_{n_k}| + E_{n_k}) \\
&\leq (2u + O(mu^2)) \sum_{k=1}^{m} (\sum_{i=1}^{n_k} |x_{k,i}| + (2u + O(\frac{n}{m}u^2)) \sum_{i=1}^{n_k} |x_{k,i}|) \\
&\leq (2u + O(mu^2))(1 + 2u + O(\frac{n}{m}u^2)) \sum_{i=1}^{n} |x_i| \\
&\leq (2u + 4u^2 + O(mu^2) + O(mu^3) + O(\frac{n}{m}u^3) + O(nu^4)) \sum_{i=1}^{n} |x_i|
\end{aligned}
$$

Now, let's derive the error bound for MapReduce Kahan algorithm.

$$
\begin{aligned}
&|\hat{S}_n - S_n| = |\hat{S}_n - \sum_{i=1}^{m} S_{n_k}| \\
=&|\hat{S}_n - \sum_{i=1}^{m} \hat{S}_{n_k} + \sum_{i=1}^{m} \hat{S}_{n_k} - \sum_{i=1}^{m} S_{n_k}| \\
\leq&|\hat{S}_n - \sum_{i=1}^{m} \hat{S}_{n_k}| + |\sum_{i=1}^{m} \hat{S}_{n_k} - \sum_{i=1}^{m} S_{n_k}| \\
\leq&|\hat{S}_n - \sum_{i=1}^{m} \hat{S}_{n_k}| + \sum_{i=1}^{m} |\hat{S}_{n_k} - S_{n_k}| \\
\leq&|\hat{S}_n - \sum_{i=1}^{m} \hat{S}_{n_k}| + \sum_{i=1}^{m}((2u + O(\frac{n}{m}u^2)) \sum_{i=1}^{n_k} |x_{k,i}|) \\
\leq&(2u + 4u^2 + O(mu^2) + O(mu^3) + O(\frac{n}{m}u^3) + O(nu^4)) \sum_{i=1}^{n} |x_i| \\
&+ (2u + O(\frac{n}{m}u^2)) \sum_{i=1}^{n} |x_i| \\
\leq&(4u + 4u^2 + O(mu^2) + O(\frac{n}{m}u^2) + O(mu^3) + O(\frac{n}{m}u^3) + O(nu^4)) \sum_{i=1}^{n} |x_i|
\end{aligned}
$$

As a result, the relative error bound is $\frac{|E_n|}{|S_n|} \leq [4u + 4u^2 + O(mu^2) + O(\frac{n}{m}u^2) + O(mu^3) + O(\frac{n}{m}u^3) + O(nu^4)]\kappa_{\mathrm{X}}$. ■