SAND2011-1265C

Design and Performance of a Scalable, Parallel Statistics Toolkit

Philippe Pébay*, David Thompson*, Janine Bennett°, and Ajith Mascarenhas°
Sandia National Laboratories
*MS {*9159,°9051}, P.O. Box 969*
Livermore, CA 94551 U.S.A.
Email: {pppebay,dcthomp,jcbenne,aamasca}@sandia.gov

***Abstract*—Most statistical software packages implement a broad range of techniques but do so in an ad hoc fashion, leaving users who do not have a broad knowledge of statistics at a disadvantage since they may not understand all the implications of a given analysis or how to test the validity of results. These packages are also largely serial in nature, or target multicore architectures instead of distributed-memory systems, or provide only a small number of statistics in parallel.**

This paper surveys a collection of statistics algorithm implementations developed as part of a common framework over the last 3 years. The framework strategically groups modeling techniques with associated verification and validation techniques to make the underlying assumptions of the statistics more clear. Furthermore it employs a design pattern specifically targeted for distributed-memory parallelism, where architectural advances in large-scale high-performance computing have been focused. Moment-based statistics (which include descriptive, correlative, and multicorrelative statistics; principal component analysis (PCA); and k-means statistics) scale nearly linearly with the data set size and number of processes. Entropy-based statistics (which include order and contingency statistics) do not scale well when the data in question is continuous or quasi-diffuse but do scale well when the data is discrete and compact. We confirm and extend our earlier results by now establishing near-optimal scalability with up to 10,000 processes.

***Keywords*-Informatics, Statistics, Principal Component Analysis, Clustering, Parallel Computing, Design Patterns**

I. Introduction

Many tools provide ad hoc, serial, statistical analysis capabilities. These two descriptors (ad hoc and serial) are important limitations which this paper attempts to address. Ad hoc implementations are, by definition, well-suited to a specific task or range of tasks but can be hard to adapt to new workflows and, more importantly, hard for inexperienced users to evaluate and combine into a meaningful analysis workflow. Serial implementations of statistical algorithms are frequently less memory intensive than other scientific software because common assumptions such as independence of observations lead to streaming or on-line algorithms. However, as data sets grow to peta-scale and beyond [1], relying on serial algorithms is no longer feasible. Existing statistical software is either serial, parallelized only across multiple cores of a single node, has limited support for parallel computation of statistics, or requires that parallel operations be explicitly specified, cf. in particular [2] and [3] regarding R and Stata, respectively, and [4] for a survey.

Consider, for example, R – which is an interpreter designed for statistical calculations plus a set of modules that implement various statistics. Firstly, the interpreter itself is serial. A number of modules provide shared-memory and distributed-memory parallel algorithm implementations for specific algorithms such as hierarchical clustering, however most do not focus on distributed data but rather distributed computation (e.g., using parallelism to perform many random samples for Markov-chain Monte Carlo analysis instead of computing statistics on large distributed data sets). Other modules for R provide low-level access to parallel message-passing libraries such as MPI [5] but require users to explicitly direct the communication in order to obtain a result. This is undesirable since, even if users were familiar with the underlying libraries, the development of numerically stable parallel versions of serial algorithms is not a trivial task. The canonical repository for R scripts has a detailed list of what parallel functionality is provided [6].

This paper presents a consistent design pattern and programmatic interface for a wide range of data-parallel statistical analyses and describes how each statistic fits the pattern and how its implementation scales in a distributed-memory parallelism context. The design decisions made during development were motivated by two primary factors: first, we wanted to mimic the predominant types of data analysis workflows, so that a data analyst using our framework would find it natural and intuitive to use; second, we wanted the design to be conducive to embarrassingly parallel implementations when possible. This is accomplished by isolating those parts of the analysis which by construction are not embarrassingly parallel (due to the mathematics of the statistical analysis itself, not due to our design) so that parallel design trade-offs are limited to those components where embarrassingly parallel implementations are not viable.

All of the algorithms discussed here are implemented as C++ classes in VTK [7], itself part of the Titan Informatics

This work was supported by the Office of Defense Programs and the Office of Advanced Scientific Computing Research of the U.S. Department of Energy, and by the Laboratory Directed Research and Development program at Sandia National Laboratories. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the U.S. Department of Energy under contract DE-AC04-94AL85000.

Table I: A table of observations used as input data by a statistics algorithm.

row	A	B	C	D	E
1	0	1	0	1	1.03315
2	1	2	2	2	0.76363
3	0	3	4	6	0.49411
4	1	5	6	24	0.04492
5	0	7	8	120	0.58395
6	1	11	10	720	1.66202

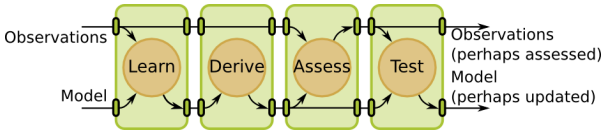

Figure 1: The 4 operations of statistical analysis and their interactions with input observations and models. When an operation is not requested, it is eliminated by connecting input to output ports.

Toolkit [8]. The source code is freely and publicly available in order that the results in this paper may be reproduced and compared to other implementations as desired.

In this paper we summarize the scalable, parallel statistical analysis toolkit which we have designed and implemented, in terms of available capabilities, design choices, and parallel performance. While many of the statistics have been documented in other publications, this paper provides guidelines for choosing among them. We begin in § II by providing a complete list of the currently implemented statistical workflows and of their functionality as they are articulated within our statistics design pattern. In § III we provide a detailed parametric study of the strong and weak parallel scaling of those statistics which we have not yet published, with up to 10,000 processes. We conclude this paper in § IV by outlining perspectives for further work and generalizing some previous findings made in a more limited context (cf. [9] and [10] in particular).

II. The Parallel Statistics Workflows

The following 7 parallel statistics workflows are implemented at the time of writing (February 2011):

- descriptive statistics,
- histograms and order statistics,
- bivariate linear correlation and regression,
- contingency statistics and information entropy,
- multi-variate linear correlation,
- principal component analysis,
- k-means clustering.

In this section, we explain what type of input data sets the statistical engines can process, how our design constraints have resulted in the splitting of the statistical analysis workflow into 4 distinct operations, and how those are articulated for each of these 7 parallel statistical engines. We describe each operation in detail, illustrate their use in the case of descriptive statistics, and explain how we implemented them in the context of data parallelism to achieve as much parallel speed-up as possible.

A. Data Organization

Each statistical workflow processes data sets stored in one or more tables: observations in the first table, and statistical model and test data in the other tables. In particular, each column of the input observations table is a variable, while each row is an observation (i.e., a simultaneous occurrence of one value for each variable). By design, the statistics classes do not allow table rows with missing entries. This is because application-specific rules are generally used to determine how such rows should be processed. Typically applications will either discard such rows or interpolate missing values from spatial or temporal neighbors. As a result, observation tables must be dense, as illustrated in Table I, which has 6 observations of 5 variables. Preprocessing raw measurements into simultaneously observed tuples is not part of the implementation and is beyond the scope of this paper. However, the interested reader can refer to [11] for one example of interpolation performed prior to statistical analysis, in the case of asynchronous data collection.

B. Operations

In order to meet the two overlapping but not exactly congruent design requirements outlined in § I, we partition the statistical analysis workflow into 4 operations: Learn a model from observations, Derive statistics from a model, Assess observations with a model, and Test the null hypothesis. These operations, when all are executed, occur in order as shown in Figure 1. However, it is also possible to execute only a subset of these, for example when it is desired that previously computed models, or models constructed with expert knowledge, be used in conjunction with existing data. Note that in earlier publications (e.g., [7], [9], [10]) only the first 3 operations are mentioned; the Test operation, which we initially saw as a part of Derive, was separated out for reasons we discuss in § II-C. These operations, performed on a request comprising a set of columns of the input observations table, are denominated as follows:

1) Learn: Calculate a primary statistical model from an input data set. By primary, we mean the minimal (in particular, non-redundant) representation of the desired model for a given statistical technique. For example, with descriptive statistics, those are the sample size, minimum, maximum, mean, and centered M_2, M_3 and M_4 aggregates (*cf.* [9]). For a request limited to column B of Table I, these values are 6, 1, 11, $4.8\bar{3}$, $68.8\bar{3}$, $159.\bar{4}$, and $1759.819\bar{4}$, respectively.

2) Derive: Calculate a more detailed statistical model from a minimal model. By "more detailed", we mean a model which includes the primary model along with additional, derived or redundant statistics, generally more commonly used for analysis purposes than the primary

Table II: The different operations currently made available by the Titan statistics classes.

	Learn	Derive	Assess	Test
Descriptive	Calculate minimum, maximum, mean, and centered M_2, M_3 and M_4 aggregates [9]	Calculate variance, standard deviation, skewness, and kurtosis (various estimators available for each statistic)	Mark with relative deviation (one-dimensional Mahalanobis distance [12])	Calculate Jarque-Bera statistic [13] and perform χ^2 goodness of fit test
Order	Calculate histogram	Calculate arbitrary quantiles (e.g., quartiles, deciles, etc.)	Mark with quantile index	Calculate Kolomogorov-Smirnov test statistic [14]
Correlative	Calculate minima, maxima, means, and centered M_2 aggregates [9]	Calculate variances, covariance, Pearson correlation r, and both linear regressions	Mark with squared two-dimensional Mahalanobis distance [12]	Calculate bivariate Jarque-Bera-Srivastava statistic [15] and χ^2 goodness of fit test
Contingency	Calculate the bivariate contingency table (also called a 2-dimensional histogram)	Calculate joint, conditional, and marginal probabilities, as well as information entropies	Mark with joint and conditional PDF values, as well as pointwise mutual information	Calculate Pearson χ^2 test of independence without and with Yates correction [16]
Multi-Correlative	Calculate means and pairwise centered M_2 aggregates [9]	Calculate covariance matrix and its (lower) Cholesky decomposition	Mark with squared multi-dimensional Mahalanobis distance [12]	N/A
PCA	Identical to the multi-correlative algorithm	Identical to the multi-correlative algorithm, plus the eigenvalues and eigenvectors of the covariance matrix [17]	Mark with coordinates in basis of all, or only first eigenvectors with cumulative energy above a given threshold	Calculate multivariate Jarque-Bera-Srivastava statistic [15] and perform χ^2 goodness of fit test
k-Means	Compute k cluster centers given a positive integer k [18]	Calculate global and local rankings amongst sets of clusters, and total error [19]	Mark with closest cluster id and associated distance for each set of cluster centers	In progress

statistics from which they derive. For instance in the case of descriptive statistics, the following derived statistics are calculated from the minimal model: variance, standard deviation, skewness, kurtosis, and sum. These additional values for column B of Table I are $13.7\overline{6}$, 3.7103, 0.52025, -1.4524, and 29 respectively.

3) Assess: Annotate each observation with a number of quantities relative to a given a statistical model. These quantities generally measure how well the particular observation coincides with the model. The model used to annotate an observation need not have been calculated from the same data. With, e.g., descriptive statistics, each datum is marked with its relative deviation with respect to the model mean and standard deviation (i.e., the one-dimensional Mahalanobis distance [12]). Table I, described as having 5 input variables earlier, can also be interpreted as having 4 input variables (A–D), plus a new column (E) containing the Mahalanobis distance of each observation in column B with respect to the priorly calculated mean and standard deviation.

4) Test: Given a statistical model, and possibly a data set, calculate at least one test statistic so at least one hypothesis can be tested. For descriptive statistics, a Jarque-Bera test of goodness of fit [13] is performed, calculating the test statistic with the skewness and kurtosis, and then retrieving the corresponding p-value from the χ^2 distribution with 2 degrees of freedom by a single call to R for all variables at once; therefore, a second pass through the data is not needed. Again for column B of Table I, one obtains a test statistic of 0.79803 for a corresponding p-value of 0.67098.

The embodiments of Learn, Derive, Assess, and Test for all of the statistics implemented are shown in Table II.

C. Parallelism

While the first of our design goals for partitioning the workflow into 4 operations was to mimic the typical use patterns of statistical analysis, the second design goal was the enabling of scalable parallelization on distributed data. And indeed, from the parallelism standpoint, the partition reduces two operations to the map-reduce pattern [20] and the remaining two are embarrassingly parallel.

Specifically, Learn is essentially a special case of the map-reduce pattern [20], a framework within which the map function generates a set (key,value) pairs in parallel. All intermediate values associated with the same intermediate key are then merged by the reduce function to compute the final solution of interest. In some of our statistical algorithms, it is not necessary to communicate the keys for there is a fixed number of them, identical across all processes, and these keys may be ordered uniquely, so sending values alone is unambiguous. However, for other algorithms, tables with an arbitrary number of key-value pairs must be communicated and different keys may be present on each process.

By construction, Learn is the only operation which always requires inter-process communication; for instance, in the case of descriptive statistics, cardinality, extremal values, and centered aggregates up to the fourth order must be exchanged and updated to assemble a global model. The Test operation may, for some types of analyses, also require inter-process communications when a second pass through

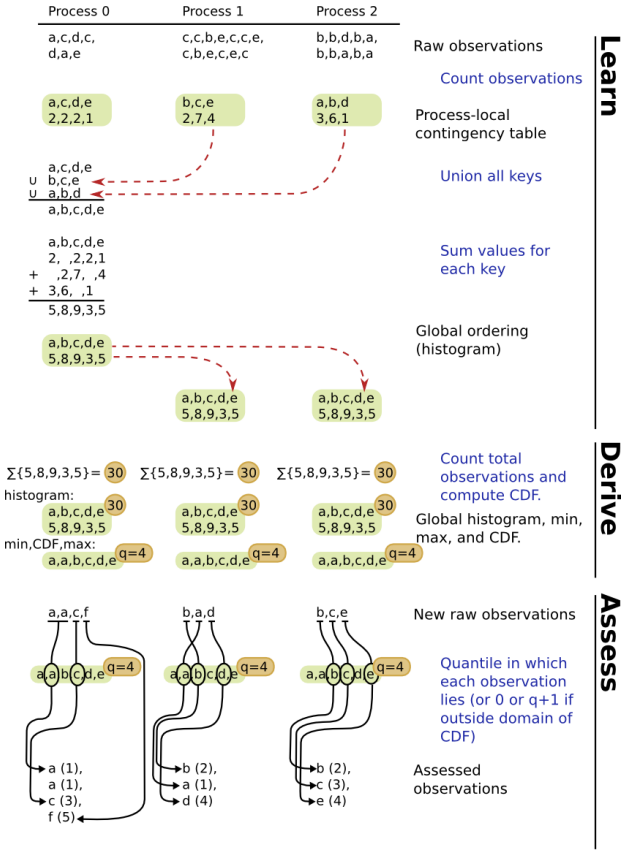

Figure 2: Example showing parallel execution of the Learn, Derive, and Assess operations of order statistics on 3 processes.

the data is required by the type of statistical test being performed. As mentioned in § II-B Test was separated from Derive because:

(i) Test tends to be more computationally intensive and in our current implementation relies in some cases on calls to the R library [21] for p-value retrieval,

(ii) For some statistics algorithms it requires a second pass through the data in order to compute per-observation quantities involving derived model information that cannot be obtained in an on-line fashion[1], henceforth requiring a second step of parallel updates, and

(iii) it rests on assumptions which are not accepted by all statisticians, namely statistical significance.

We note that the Learn (and Test) operations of most of the currently implemented statistical algorithms demonstrate optimal parallel speed-up and scalability, as they rely on a small number of statistics to be exchanged and updated, such as in the aforementioned descriptive statistics case.

[1]For example, the multi-variate normality test requires that multi-skewness and multi-kurtosis be computed, but those estimators in turn need each observation to be projected to the eigenvectors of the data set, which cannot be computed incrementally and thus are not available until after all observations have been processed.

In [9] we have shown how to perform these updates in a numerically stable, yet single-pass, way, for all statistics algorithms which make use of moments and co-moments, hence resulting in optimal parallel scaling for the Learn and Test operations for such statistics. In [10], on the other hand, we have discussed the design trade-offs and limitations encountered when computing contingency tables in parallel, which resulted in our choosing of what we called the *Full-reduce+broadcast* approach; in this case, the Learn operation of the algorithm is difficult to characterize because its parallel scaling properties range from embarrassingly parallel to serial, depending on the distribution of the input data. We nonetheless showed that the case where our parallel design does not scale well exactly corresponds to the class of problems for which contingency tables are not suited for analysis, hence validating our design pattern even for non moment-based algorithms. For the Test operation of those engines which currently rely on R for the retrieval of the p-values corresponding to calculated χ^2 statistics, it is important to note that the invokation of R is done by each process independently from all other processes, each one retrieving the same p-value from a previously globally calculated χ^2 statistic. Therefore, the use of the (serial) R package does not invalidate the embarrassingly parallel nature of this operation.

The Derive and Asses operations can always be computed in an embarrassingly parallel fashion. The calculations for the derivation of additional statistics from a primary statistical model always need only be executed once, without communication, independently and after all parallel updates of primary variables. Likewise, the Assess operation annotates only the observations in the portion of the data residing on the local process, and thus no communication is required. The result is a new dataset distributed in the same way across processes but with additional columns for the annotations.

Figure 2 shows the communication patterns and parallel computations performed when calculating order statistics on simple character data distributed across 3 processes; note that the Test operation is not depicted here as its implementation is not complete yet for all statistical engines and some changes are to be expected in the near future.

D. Code Structure

Each statistics algorithm is first implemented in C++ in a serial fashion as a subclass of a base statistics algorithm class that provides virtual methods for each of the 4 phases (Learn, Derive, Assess, and Test). Each serial implementation then serves as a base class for a parallel implementation which overrides the Learn and, optionally, Test methods as shown in Figure 3. These methods invoke the serial superclass's implementation and then use an abstract communication-class instance to perform the inter-process communication required for the reduction operation. This separation between serial and parallel implementations also allows for

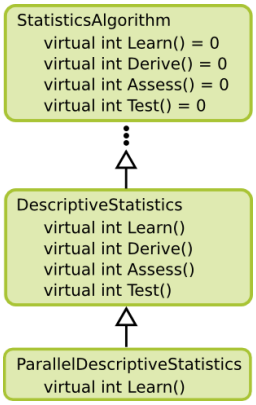

Figure 3: The inheritance pattern of the descriptive statistics implementation. This serves to illustrate how parallel communication is separated into a subclass which may use its superclass's implementation for all but the reduction operation.

experiments with different programming models: e.g., one where a full reduction operation might not be necessary to sufficiently, in some sense, approximate the global model.

The descriptive, correlative, multi-correlative, and PCA statistics algorithms perform their respective parallel aggregations using the update formulas presented in [9], cf. [22] for implementation details. These moment-based parallel statistics classes thus all have optimal parallel scalability properties. Similarly, the order and contingency statistics classes respectively derive from the univariate and bivariate statistics classes and implement their own aggregation mechanisms for the Learn operation. However, unlike moment-based statistics (descriptive, correlative, multi-correlative, PCA, and k-means), these aggregation operations are not, in general, embarrassingly parallel; as a result, these statistical methods scale sub-optimally when used outside of their intended domain of applicability, as shown in [10] for contingency statistics, and here in § III for order statistics.

III. Parallel Scalability Studies

In earlier publications we have studied the parallel speed-up and scalability properties of some of the parallel statistics engines of Titan. For instance, in [9] we demonstrated that thanks to numerically stable, yet single pass, parallel update formulas, our implementation of moment-based statistics such as descriptive, correlative, multi-correlative statistics, and PCA achieved near optimal strong and weak scaling.

On the other hand, we dealt in [10] with the parallel computation of contingency statistics, where scalability of the algorithm is particularly difficult to characterize because it can run in an embarrassingly parallel manner, in a completely serial manner, or anywhere in between, depending on the nature of the input data. However, by making design trade-offs in the parallelization of the Learn operation, we were able to show that the types of input resulting in poor scalability corresponds to the case where contingency tables should not be used, not from a computational, but from a statistical point of view. We have generalized these design trade-offs, namely, what we call the *Full-reduce+broadcast* pattern, and applied them to other statistics where the variability of the size of the primary statistics model manifests itself: histogram and order statistics.

In this section, we present the results of a series of tests with our implementation of the parallel order statistics algorithms and show that we retrieve speed-up results similar to contingency statistics. We therefore conjecture that this is a feature of the approach itself and not a mere accident. Finally, we contrast these results with another recent addition to the toolkit, the k-means statistics engine, which does indeed scale optimally as we apply to it our online updating methodology for moment-based statistics.

A. Evaluation Criteria

We intend here to study the scalability properties of the algorithms of interest independently of load-balancing issues and thus use a series of (pseudo-) randomly-generated samples, hence creating data sets of equal size on each process in order to obtain perfectly balanced test cases. Only Learn, Derive, and Assess operations are invoked, as Test relies on the external calls to R, whose execution we want to keep aside from our study. We also exclude the amount of time needed to create the input data tables from the analysis, as inputs can in reality come from a variety of outside sources, such as databases or flat files. With this setting, we use the following evaluation criteria:

1) strong scaling, i.e., at constant total work, also known as relative speed-up, and
2) weak scaling, i.e., at constant work per process, also known as rate of computation scalability.

We provide here a description of these evaluation criteria. Hereafter p will denote the number of processes and N (or $N(p)$ when it varies with p) the size of the problem, which in our case is the data set cardinality.

1) Strong Scaling: The wall clock time measured to execute the calculation is denoted $T_N(p)$. Then, strong scaling is defined as:

$$S_N(p) = \frac{T_N(1)}{T_N(p)}.$$

Some authors prefer to write the numerator as T_s rather than $T_N(1)$ to make it clear that the parallel algorithm should be compared to the most efficient serial implementation available and not just the parallel algorithm run on a single process. Evidently, optimal (linear) scaling is attained when $S_N(p) = p$ and, therefore, strong scaling results can be visually inspected by plotting S_N versus the number of processes: optimal scaling is revealed by a line, the angle bisector of the first quadrant.

2) Weak Scaling: The rate of computation is defined as:

$$r(p) = \frac{N(p)}{T_{N(p)}(p)},$$

where $N(p)$ now varies with p. Weak scaling is then measured by normalizing the rate of computation by that which is obtained with a single process. In particular, if the sample size is made to vary in proportion to the number of processes, i.e., if $N(p) = pN(1)$, then

$$R(p) = \frac{r(p)}{r(1)} = \frac{pT_{N(1)}(1)}{T_{pN(1)}(p)} = \frac{pT_{N(1)}(1)}{pT_{N(1)}(p)} = \frac{T_{N(1)}(1)}{T_{N(1)}(p)}.$$

Therefore, optimal (linear) scaling is attained with p processes when $R(p) = p$. Note that without linear dependency between N and p, the latter equality no longer implies optimal scalability. Parallel scalability can thus also be visually inspected by plotting the values of R versus the number of processes, and in this case also, optimality corresponds to the angle bisector of the first quadrant.

B. Test Platforms

We first conducted systematic, multiple-parameter study of the scalability properties of all algorithms using a 240-CPU computational cluster; we then validated some of the observed scalability results at a much larger scale with up to 10,000 cores on a tera-scale system.

1) `catalyst`: The small test platform was the `catalyst` computational cluster at Sandia National Laboratories, which comprises 120 dual-CPU 3.06 GHz Pentium Xeon compute nodes with 2 GB of memory each, operating with a Linux 2.6.17.11 kernel. The results reported in this article were computed using two processes per node, i.e., one per processor, except for the single-process baseline runs. Also, the numbers of processes was chosen as increasing powers of 2, for convenience only; using other values did not change the outcome of the evaluation.

2) `jaguar`: The very large platform is the `jaguar` tera-scale cluster at Oak Ridge National Laboratories. Its Cray XT4 partition comprises 7,832 compute nodes in addition to dedicated login/service nodes. Each compute node contains a quad-core AMD Opteron 1354 (Budapest) processor running at 2.1 GHz, 8 GB of DDR2-800 memory (although some nodes use DDR2-667 memory), and a SeaStar2 router. The resulting partition contains 31,328 processing cores, more than 62 TB of memory, over 600 TB of disk space, and a peak performance of 263 teraflop/s (263 trillion floating point operations per second). We carried out the scalability study on this XT4 partition using up to 10,000 cores.

C. The All-reduce Pattern and k-Means Statistics

The phrase *cluster analysis* describes a class of unsupervised learning algorithms whose primary objective is to partition data sets into subsets (*clusters*) according to a given measure of association. For instance, with k a positive integer, k-means clustering [18] is one such algorithm which groups observations into k clusters by minimizing a metric between each observation and the center of the cluster to which it is assigned. The general algorithm can be summarized as follows:

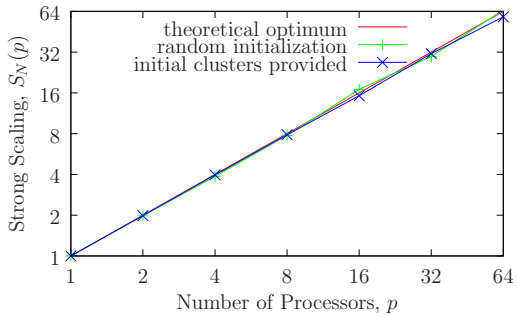

Figure 4: Parallel k-means statistics: strong scaling (at constant total work), with a total data size of $N = 10^6$ doubles.

1) Choose k, the number of clusters.
2) Determine k initial cluster centers.
3) Assign each observation to the nearest center.
4) Recompute the new cluster centers.
5) Repeat 3 and 4 until a convergence criterion is met.

This straightforward k-means clustering algorithm is currently available in Titan; note, however, that it is a heuristic algorithm and therefore convergence is not guaranteed. In addition to this intrinsic limitation, the value of k must be provided to the algorithm and the final results may strongly depend on the initial choice of cluster centers. Another consideration that should be noted regarding k-means clustering is that it requires that the data being processed have some notion of a mean, which is used to compute the new cluster centers. When data does not have a computable mean, an alternate approach is to use the *k-medoids* algorithm [23] which uses data observations as cluster centers. Our implementation uses means *stricto sensu* and thus belongs to the class of moment-based statistics. As such, the parallel subclass was implemented using the All-reduce approach we used for other moment-based parallel engines [9]; as a result, optimality can be expected for both strong and weak scaling. We therefore verify it here experimentally with only up to $p = 64$ processes on `catalyst`, for we already reported in [9] optimal scalability with up to $p = 1500$ for another moment-based statistics algorithm.

The parallel k-means engine is tested with input tables comprising 6 columns, each of which is created at run time by sampling from 8 independent pseudo-random Gaussian variables $(X_i)_{1 \le i \le 8}$ with mean $7i$ and unit standard deviation, thus creating 6-dimensional data clustered around those 8 different means.

1) Strong Scaling: All cases in the study of strong scaling have identical total size $N = 10^6$. The strong scaling results

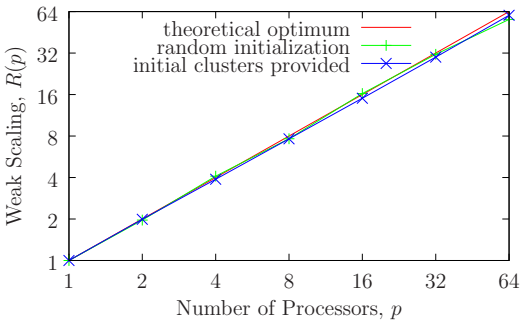

Figure 5: Parallel k-means statistics: weak scaling (at constant work per process), with $N(p)/p = 2.5 \times 10^5$.

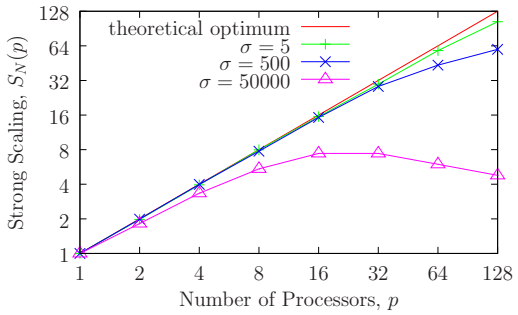

Figure 7: Parallel order statistics: strong scaling (at constant total work), with a total data size of $N = 2.56 \times 10^7$ doubles.

are plotted in Figure 4. The heuristic nature of the k-means algorithm makes it difficult to predict the time required for a particular input data set. However, because the k-means algorithm performs iterative passes through the data until a convergence criterion is met, it is particularly amenable to a data parallel implementation. As can be seen, the measured scaling is optimal (within $\pm 10\%$ fluctuations at least partially attributable to system overheads). One can also note some instances where uninitialized runs appear to scale super-linearly, which may be attributed to the early termination at a locally optimal solution.

2) Weak Scaling: A series of increasingly large samples was generated to assess weak scaling, each sample consisting of 6 variables with $N(p) = np$ observations, where $n = 2.5 \times 10^5$ is the per-process cardinality. The corresponding results are plotted in Figure 5. These clearly exhibit optimal scalability (again within fluctuations attributable to outside causes), thus experimentally verifying the embarrassingly parallel nature of these algorithms. As with the speed-up results, it should be noted that in some instances the uninitialized runs appear to scale super-linearly, which may be attributed to early termination at a locally optimal solution.

D. The Full-reduce+broadcast Pattern and Order Statistics

Order statistics can be calculated on data sets equipped with a total order such as numeric values or strings (endowed with the lexicographic order). Its Learn operation computes per-variable histograms, i.e., key-value maps from each distinct observation (key) to the number of times that observation appears in the table (value). The Derive operation then computes a probability mass function from each histogram, and a set of quantiles for each variable, where the number n_q of such quantiles (as well as the interpolation method when several choices are possible) is set by the user; for instance, with $n_q = 4$ (resp. $n_q = 10, 100$) the quantiles are quartiles (resp. deciles, percentiles). These are obtained by traversing the histogram keys in increasing order, until the sum of their values (cardinality) is equal or greater than iN/n_q for $i \in \{1, 2, \ldots, n_q\}$, where N is the total cardinality of the histogram. The Assess operation marks each datum with an index indicating to which inter-quartile it belongs; if it is smaller (resp. bigger) than the minimum (resp. maximum), then it is marked with 0 (resp. $n_q + 1$). Finally, the Test operation estimates, by means of a Kolmogorov-Smirnov statistic (cf. [14]), the difference between the empirical CDF of the observations and a piecewise-constant CDF.

For the series of tests presented here, input tables with a single column are created at run time by rounding to the nearest integer the outcomes of a centered normal pseudo-random variable with standard deviation $\sigma > 0$. The values of σ are chosen with increasing values in $\{5; 500; 50000\}$ in order to yield order tables with varying sizes, for the probability that any realization a of centered Gaussian fall outside the $[-3\sigma, 3\sigma]$ interval is only ca. 0.27%. With the chosen set of standard deviations, the observed histograms – illustrated with realizations in Figure 6 – have sizes on the order of 10, 10^3, and 10^5, respectively. The overall scaling is a strong function of the histogram size as the parallel communication costs for these histograms are vastly different. This single column is then selected as the column of interest for the order statistics engine.

1) Strong Scaling: All cases in the study of strong scaling are computed on `catalyst` and have identical total size $N = 2.56 \times 10^7$. The strong scaling results are plotted in Figure 7. We observe that with $\sigma = 5$, the ensuing local order tables are small enough that the cost of the parallel updates is negligible to the point that the algorithm becomes, effectively, embarrassingly parallel: the measured scaling is nearly optimal, within 4%, which may also be due in part to operating system overhead unrelated to the algorithm itself. Near optimal speed-up continues until, as a result of the decreasing amount of work per process, time taken by updates and overheads, albeit small in absolute terms, becomes sufficiently noticeable as compared to the overall computation time. With $\sigma = 5$, this trends begins to slightly appear with $p = 64$ and further with $p = 128$, where the speed-up is sub-optimal by ca. 19%. Eventually

Figure 6: Examples of values typically present in the $\sigma = 5$ (blue), 500 (yellow), and 50000 (green) input data sets used for benchmarking order statistics when $N = 3.2 \times 10^6$.

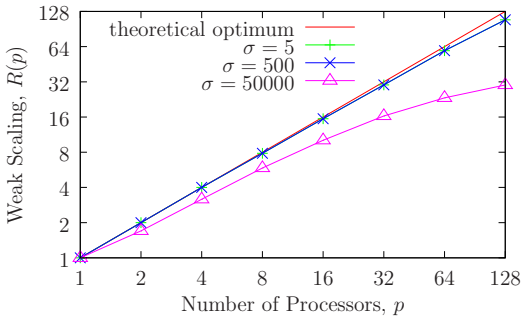

Figure 8: Parallel order statistics: weak scaling (at constant work per process) on `catalyst`, with $N(p)/p = 3.2 \times 10^6$.

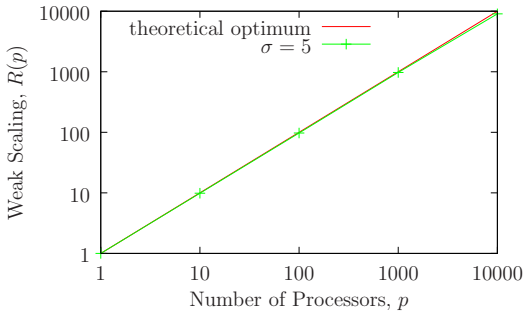

Figure 9: Parallel order statistics: weak scaling (at constant work per process) on `jaguar`, with $N(p)/p = 2 \times 10^7$.

the decreasing amount of work per process should result in no additional speed-up, but with $\sigma = 5$ we have not observed for p up to 200.

On the other hand, with larger values of the standard deviation, and thus with much larger histograms to be exchanged between processes, optimal speed-up is lost much earlier: specifically, with $p \geq 32$ when $\sigma = 500$, and as early as with $p = 2$ when $\sigma = 50000$. In fact, in the latter case with $p > 16$ no more parallel speed-up is achieved as the Amdahl limit [24] is reached and speed-down eventually occurs. Further, for sufficiently large (in a platform dependent sense) values of σ, no parallel speed-up can be achieved at all, effectively turning the algorithm into a serial implementation.

From this we can draw two main conclusions, confirming those which we had already discovered in [10] for the parallel contingency statistics engine:

(i) This algorithm scales optimally with categorical data, not with (quasi-)continuous data. This experimental observation is aligned with the fact that order statistics are intended for discrete rather than continuous measurements.

(ii) One should be careful with claims that some algorithms would, or would not, be *a priori* amenable to "Map-Reduce" implementations. As this case confirms, the same algorithm can behave as an embarrassingly parallel one, or as a completely coupled, intrinsically serial one, or anything in between depending solely on the value of a single input parameter.

As a result, once again we assert that there may be a continuum of speed-up properties for the same algorithm; there is no general solution to the problem of parallelizing data analysis algorithms in a scalable fashion.

2) Weak Scaling: We begin by assessing weak scaling on `catalyst` with the same test case as in § III-D1, with the difference that, in order to maintain a constant work per process, increasingly large samples are created: specifically, each data sets contains $N(p) = np$ doubles, where $n = 3.2 \times 10^6$. The weak scaling results are plotted in Figure 8. When $\sigma = 5$, and thus a relatively small order table, the algorithm exhibits nearly optimal scaling with the same observations as for the strong scaling. With $\sigma = 500$ one continues to observe parallel scalability for $p \geq 128$, with an overall order of about 0.96. However, with the very large histograms generated when $\sigma = 50000$, scalability is sub-optimal from the beginning with $p = 2$ processes, and this becomes more pronounced as the number or processes increases: with $p = 128$ the overall order drops to about 0.7.

We then verify the optimal scalability results for $\sigma = 5$ by increasing the size of the problem to $n = 7 \times 10^7$ and up to $p = 10^4$ on `jaguar`. The corresponding weak scaling results are plotted in Figure 9: near-optimal scalability continues up to $p = 10^4$ with an order of almost 0.99.

These observations confirm what we both noticed in § III-D1 for strong scaling and in [10] in the case of contingency statistics for strong and weak scalability, namely, that such algorithms should not be a statistic of choice when dealing with quasi-continuous data (which, in the context of floating-point representation, amounts to the same as extremely large discrete data sets). This situation means that either another statistical analysis method must be used, or the data should be quantized differently (e.g., with different bucket sizes). In any event, the main conclusion which we can make here is again that this engine scales optimally when used in the context of honestly discrete measurements, which is the category of data for which its underlying statistical method is intrinsically intended.

IV. Conclusion and Future Work

We have presented a comprehensive view of the scalable, parallel statistical analysis framework which we designed and implemented. We have described our decomposition of the statistical analysis workflow of all currently available statistical methods into 4 separate base operations, and demonstrated that this division is sensible both from the data analyst's perspective and for the sake of parallel scalability. Furthermore, we confirmed with the k-means statistics engine that our previously described single-pass, numerically robust parallel update formulas allow for embarrassingly parallel implementations of moment-based statistics algorithms. On the other hand, we extended our earlier and more limited results for non moment-based statistical methods, which are not amenable to an embarrassingly parallel implementation. Specifically, we confirmed that the design choices we made between efficiency and robustness for this class of problems allow for optimal strong and weak scaling when those statistical techniques are used in their appropriate context, namely, when the input data is not quasi-diffuse, but honestly represents discrete measurements.

A. Implementation

Our implementation is available as part of Titan, an open-source informatics toolkit built on top of the Visualization Tool Kit, VTK. The statistics classes, along with some of the test programs used above for testing purposes, are part of VTK. Specifically, they belong to the `Infovis` directory of VTK, publicly available via Git. Detailed download and build instructions are provided by these pages:

```
http://www.vtk.org/Wiki/VTK/Git
http://www.vtk.org/Wiki/VTK_Building_VTK
```

A number of programs which make use of the parallel statistics classes of Titan are available in the `VTK/Infovis/Testing/Cxx/` sub-directory of VTK. These can be used to reproduce or extend the parallel scalability tests in this paper. For additional details about VTK, please refer to the user manual [7] or the online documentation [25]. ParaView [26] and OVIS [11] are examples of open-source applications that use this parallel statistics functionality, in the context of high performance computing.

Ongoing and future work includes the implementation of Test operations for those few engines who do not yet have one, the generalization of contingency tables to n-way arrays, exploring other ways to measure and improve the efficiency of non moment-based parallel scalability for large and high-dimensional data, and expanding the functionality of our existing framework with additional engines as needed by client applications.

B. Efficiency

In [10], we introduced the global *efficiency* $\mathcal{E}_{\mathrm{g}}$ of a contingency as an *a posteriori* estimator obtained by comparing the number of entries in the global contingency table N_c to the number of input observations N:

$$\mathcal{E}_{\mathrm{g}} = 1 - \frac{N_q}{N}. \tag{1}$$

Here we generalize this estimator to all quantization-based statistics algorithms, by replacing N_c with the number of quantizers N_q in (1), and calling it the *reduction efficiency* estimator. The definition of *quantizer* depends on the particular statistical algorithm, but should be trivial in relevant cases: for instance, histogram values and populated contingency table entries for order and contingency statistics respectively.

Table III: Reduction efficiency for various values of the standard deviation of random inputs to order (left) and contingency (right) algorithms, where σ denotes the standard deviation of the truncated centered univariate and bivariate Gaussian pseudo-random inputs of size 8×10^6.

Engine	order statistics			contingency statistics		
σ	5	500	50000	5	50	200
$\mathcal{E}_{\mathrm{g}}$	0.9999	0.9994	0.9619	0.9977	0.9866	0.8732

In [10] we had reasoned that reduction efficiency near 0 indicates that data should either be discretized more coarsely to lower the number of distinct observations that may be encountered, or that contingency tables should be discarded for this particular problem because of the non-discrete nature of the underlying measurement. Furthermore, we reasoned that values near 1 should point at computational scalability, with the caveat that data may need to be discretized less coarsely for the statistical analysis to be meaningful. However, after computing a number of *a posteriori* estimations of reduction efficiency with both order and contingency statistics engines, we can conclude that it does not take values much smaller than 1 to result in poor parallel scaling: combining the numbers of Table III with those of Figures 7 and 8 (and their counterparts for contingency statistics in [10]), we can see that reductions smaller than 95% to 98% result in less than optimal scalability. Note that the values of σ translate differently in terms of the diffuseness of the random inputs for the two engines as one is univariate and the other is bivariate. Further study is required to determine an application-independent threshold. Moreover, when considering trade-offs between architectures and programming models, it may be more relevant to consider

$$\mathcal{E}_g = \frac{N_q}{C}\left(1 - \frac{N_q}{N}\right)$$

where C is a measure of the communication cost for an all-reduce of the model. For instance with a tree-based all-reduce of the model on a fat-tree network with bandwidth

B, one can express C as follows:

$$C = \frac{N_q \log n_d}{B}.$$

A key problem with both reduction efficiency estimators is that they require that global aggregation take place before the estimators themselves can actually be calculated. Thus, they can be used for the *a posteriori* evaluation of algorithmic properties. In future work it is important to explore of alternate *a priori* (i.e., prior to global aggregation) efficiency estimators. For instance, under reasonable statistical assumptions one might consider $\min_i \mathcal{E}_1(i)$, where $\mathcal{E}_1(i)$ is computed locally on each process i: this global-local estimator can be computed prior to global aggregation and henceforth be used as an objective function to govern *a priori* quantization.

ACKNOWLEDGMENTS

We thank Tim Shead for his help with parallel communication of variable-length strings, and Brian Wylie for his help with the integration into Titan. We also would like to thank two anonymous reviewers, whose comments and suggestions helped us to substantially improve this paper.

REFERENCES

[1] J. Becla, A. Hanushevsky, S. Nikolaev, G. Abdulla, A. Szalay, M. Nieto-Santisteban, A. Thakar, and J. Gray, "Designing a multi-petabyte database for LSST," in *Observatory Operations: Strategies, Processes, and Systems*, ser. Proceedings of the SPIE, vol. 6270, 2006.

[2] M. Schmidberger, M. Morgan, D. Eddelbuettel, H. Yu, L. Tierney, and U. Mansmann, "State-of-the-art in parallel computing with R," Department of Statistics, University of Munich, Tech. Rep. 47, 2009.

[3] Stata, "Stata/MP performance report," StataCorp LP, Tech. Rep., Jun. 2010, version 2.0.0. [Online]. Available: http://www.stata.com/statamp/statamp.pdf

[4] J. Nakano, *Parallel Computing Techniques*. Springer, 2004.

[5] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI. Portable Parallel Programming with the Message Passing Interface*, 2nd ed. MIT Press, 1999.

[6] D. Eddelbuettel, "High-performance and parallel computing with R," http://cran.r-project.org/web/views/HighPerformanceComputing.html, 2010, version 2010-09-16 downloaded 2010-11-15.

[7] I. Kitware, *The VTK User's Guide, version 5.4*. Kitware, Inc., 2010.

[8] "Titan informatics toolkit," http://titan.sandia.gov/.

[9] J. Bennett, P. Pébay, D. Roe, and D. Thompson, "Numerically stable, single-pass, parallel statistics algorithms," in *Proc. 2009 IEEE International Conference on Cluster Computing*, New Orleans, LA, Aug. 2009.

[10] P. Pébay, D. Thompson, and J. Bennett, "Computing contingency statistics in parallel: Design trade-offs and limiting cases," in *Proc. 2010 IEEE International Conference on Cluster Computing*, Heraklion, Crete, Greece, Sep. 2010.

[11] J. M. Brandt, V. D. Sapio, A. C. Gentile, J. Mayo, P. Pébay, D. Roe, D. Thompson, and M. H. Wong, "The OVIS analysis architecture," Sandia National Laboratories, Tech. Rep. SAND2010-5107, Jul. 2010.

[12] P. C. Mahalanobis, "On the generalised distance in statistics," *Proc. of the National Institute of Science of India*, no. 12, pp. 49–55, 1936.

[13] C. M. Jarque and A. K. Bera, "A test for normality of observations and regression residuals," *Revue Internationale de Statistique*, vol. 55, no. 2, pp. 163–172, 1987.

[14] D. Dacunha-Castelle and M. Duflo, *Probability and Statistics*. Springer-Verlag, 1986, vol. 1.

[15] K. Koizumi, N. Okamoto, and T. Seo, "On Jarque-Bera tests for assessing multivariate normality," *J. of Statistics: Advances in Theory and Applications*, no. 1, pp. 207–220, 2009.

[16] F. Yates, "Contingency tables involving small numbers and the χ^2 test," *Supplement to the Journal of the Royal Statistical Society*, vol. 1, no. 2, pp. 217–235, 1934.

[17] J. A. Lee and M. Verleysen, *Nonlinear Dimensionality Reduction*, ser. Information Science and Statistics. Springer-Verlag, 2007.

[18] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proc. of the 5th Berkeley Symp. on Mathematics Statistics and Probability*, L. M. LeCam and J. Neyman, Eds., 1967.

[19] J. Bennett, P. Pébay, and D. Thompson, "Scalable k-means statistics with Titan," Sandia National Laboratories, Tech. Rep. SAND2009-7855, Nov. 2009.

[20] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *OSDI'04: Sixth Symposium on Operating System Design and Implementation*, San Francisco, CA, Dec. 2004.

[21] "The R project for statistical computing," http://www.r-project.org/.

[22] J. Bennett, P. Pébay, D. Roe, and D. Thompson, "Scalable multi-correlative statistics and principal component analysis with Titan," Sandia National Laboratories, Tech. Rep. SAND2009-1687, Mar. 2009.

[23] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[24] P. S. Pacheco, *Parallel Programming with MPI*. Morgan Kaufmann, 1997.

[25] "VTK Doxygen documentation," http://www.vtk.org/doc/nightly/html.

[26] A. Henderson, *The ParaView Guide*. Kitware, Inc., 2004.