

# **SANDIA REPORT**

SAND2009-6006

Unlimited Release

Printed September 2009

## **Parallel Contingency Statistics with Titan**

Philippe Pébay and David Thompson

Prepared by

Sandia National Laboratories

Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under Contract DE-AC04-94-AL85000.

Approved for public release; further dissemination unlimited.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

**NOTICE:** This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from  
U.S. Department of Energy  
Office of Scientific and Technical Information  
P.O. Box 62  
Oak Ridge, TN 37831

Telephone: (865) 576-8401  
Facsimile: (865) 576-5728  
E-Mail: reports@adonis.osti.gov  
Online ordering: <http://www.osti.gov/bridge>

Available to the public from  
U.S. Department of Commerce  
National Technical Information Service  
5285 Port Royal Rd  
Springfield, VA 22161

Telephone: (800) 553-6847  
Facsimile: (703) 605-6900  
E-Mail: orders@ntis.fedworld.gov  
Online ordering: <http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online>



## Parallel Contingency Statistics with Titan

Philippe Pébay

Sandia National Laboratories  
M.S. 9159, P.O. Box 969  
Livermore, CA 94551, U.S.A.  
pppebay@sandia.gov

David Thompson

Sandia National Laboratories  
M.S. 9159, P.O. Box 969  
Livermore, CA 94551, U.S.A.  
dcthomp@sandia.gov

### Abstract

This report summarizes existing statistical engines in [VTK/Titan](#) and presents the recently parallelized contingency statistics engine. It is a sequel to [\[PT08\]](#) and [\[BPRT09\]](#) which studied the parallel descriptive, correlative, multi-correlative, and principal component analysis engines. The ease of use of this new parallel engines is illustrated by the means of C++ code snippets. Furthermore, this report justifies the design of these engines with parallel scalability in mind; however, the very nature of contingency tables prevent this new engine from exhibiting optimal parallel speed-up as the aforementioned engines do. This report therefore discusses the design trade-offs we made and study performance with up to 200 processors.

# Acknowledgments

The authors would like to thank:

- Tim Shead, for valuable technical discussions, in particular on—but not limited to—efficient parallel communication of variable-length character strings,
- Brian Wylie, for his comments on the integration of scalable statistical tools in [VTK/Titan](#).

# Contents

1	Introduction .....	7
1.1	The Titan Informatics Toolkit .....	7
1.2	Statistics Functionality in Titan .....	8
2	Contingency Statistics .....	12
2.1	Contingency Tables .....	12
2.2	Joint and Marginal Distributions .....	12
2.3	Conditional Probabilities .....	13
2.4	Pointwise Mutual Information .....	14
2.5	Information Entropy .....	15
3	Parallel Statistics Classes .....	17
3.1	Implementation Details .....	17
3.2	Usage .....	18
4	Results .....	20
4.1	Algorithm Scalability .....	20
4.2	Algorithm Correctness .....	25
	References .....	27

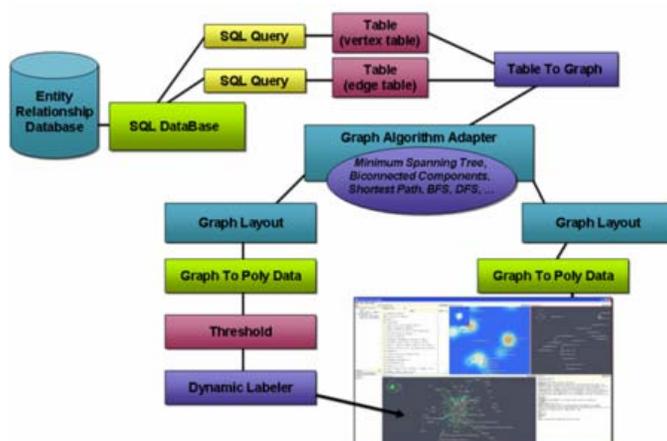
This page intentionally left blank

# 1 Introduction

This report is a sequel to [PT08] and [BPRT09], which focused on the parallel descriptive, correlative, multi-correlative, and principal component analysis engines; please refer to these references for a detailed presentation of these engines as well as an assessment of their scalability and speed-up properties.

## 1.1 The Titan Informatics Toolkit

The Titan Informatics Toolkit is a collaborative effort between Sandia National Laboratories and Kitware Inc. It represents a significant expansion of the Visualization ToolKit (VTK) to support the ingestion, processing, and display of informatics data. By leveraging the VTK engine, Titan provides a flexible, component based, pipeline architecture for the integration and deployment of algorithms in the fields of intelligence, semantic graph and information analysis.



**Figure 1.** A theoretical application built with Titan.

A theoretical application built from Titan/VTK components is schematized in Figure 1. The flexibility of the pipeline architecture allows effective utilization of the Titan components for different problem domains. An actual implementation is OverView, a generalization of the ParaView scientific visualization application to support the ingestion, processing, and display of informatics data. The ParaView client-server architecture provides a mature framework for performing scalable analysis on distributed memory platforms, and OverView will use these capabilities to analyze informatics problems that are too large for individual workstations.

The Titan project represents one of the first software development efforts to address the merging of scientific visualization and information visualization on a substantive level. The VTK parallel client-server layer will provide an excellent framework for doing scalable analysis on distributed

memory platforms. The benefits of combining the two fields are already reaping rewards in the form of functionality such as the cell lineage application below.

## 1.2 Statistics Functionality in Titan

A number of univariate, bivariate, and multivariate statistical tools have been implemented in Titan. Each tool acts upon data stored in one or more tables; the first table serves as observations and further tables serves as model data. Each row of the first table is an observation, while the form of further tables depends on the type of statistical analysis. Each column of the first table is a variable.

### 1.2.1 Variables

A univariate statistics algorithm only uses information from a single column and, similarly, a bivariate from 2 columns. Because an input table may have many more columns than an algorithm can make use of, Titan must provide a way for users to denote columns of interest. Because it may be more efficient to perform multiple analyses of the same type on different sets of columns at once as opposed to one after another, Titan provides a way for users to make multiple analysis requests of a single filter.

**Table 1.** A table of observations that might serve as input to a statistics algorithm.

row	port	protocol	row	port	protocol	row	port	protocol
1	80	HTTP	8	1122	HTTP	15	80	SMTP
2	80	HTTP	9	80	HTTP	16	20	FTP
3	80	HTTP	10	25	SMTP	17	20	FTP
4	80	HTTP	11	25	SMTP	18	20	FTP
5	80	HTTP	12	25	SMTP	19	122	FTP
6	80	HTTP	13	25	SMTP	20	20	FTP
7	8080	HTTP	14	25	SMTP	21	20	FTP

As an example, consider Table 1, containing a sample of 21 observations of network traffic across an interface, characterized in terms of (port,protocol) pairs. One can calculate univariate statistics on each of the columns of this table, or bivariate statistics using any pair of these columns (including the row index column itself).

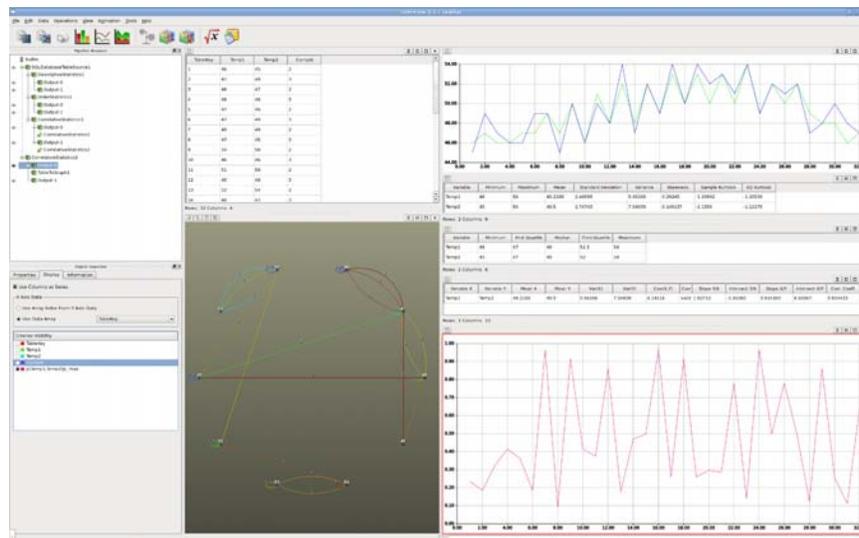
## 1.2.2 Phases

Each statistics algorithm performs its computations in a sequence of common phases, regardless of the particular analysis being performed. These phases can be described as:

**Learn:** Calculate a “raw” statistical model from an input data set. By “raw”, we mean the minimal representation of the desired model, that contains only primary statistics. For example, in the case of descriptive statistics: sample size, minimum, maximum, mean, and centered  $M_2$ ,  $M_3$  and  $M_4$  aggregates (cf. [P08]).

**Derive:** Calculate a “full” statistical model from a raw model. By “full”, we mean the complete representation of the desired model, that contains both primary and derived statistics. For example, in the case of descriptive statistics, the following derived statistics are calculated from the raw model: unbiased variance estimator, standard deviation, and two estimators ( $g$  and  $G$ ) for both skewness and kurtosis.

**Assess:** Given a statistical model – from the same or another data set – mark each datum of a given data set. For example, in the case of descriptive statistics, each datum is marked with its relative deviation with respect to the model mean and standard deviation (this amounts to the one-dimensional Mahalanobis distance).



**Figure 2.** An example utilization of Titan’s statistics algorithms in [OverView](#).

An example of the utilization of Titan’s statistical tools in [OverView](#) is illustrated in Figure 2; specifically, the descriptive, correlative, and order statistics classes are used in conjunction with various table views and plots. With the exception of contingency statistics which can be performed

on any type (nominal, cardinal, or ordinal) of variables, all currently implemented algorithms require cardinal or ordinal variables as inputs.

At the time of writing, the following algorithms are available in [Titan](#):

1. Univariate statistics:

(a) Descriptive statistics:

**Learn:** calculate minimum, maximum, mean, and centered  $M_2$ ,  $M_3$  and  $M_4$  aggregates;

**Derive:** calculate unbiased variance estimator, standard deviation, skewness ( $l_2$  and  $G_1$  estimators), kurtosis ( $g_2$  and  $G_2$  estimators);

**Assess:** mark with relative deviations (one-dimensional Mahalanobis distance).

(b) Order statistics:

**Learn:** calculate histogram;

**Derive:** calculate arbitrary quartiles, such as “5-point” statistics (quartiles) for box plots, deciles, percentiles, etc.;

**Assess:** mark with quartile index.

2. Bivariate statistics:

(a) Correlative statistics:

**Learn:** calculate minima, maxima, means, and centered  $M_2$  aggregates;

**Derive:** calculate unbiased variance and covariance estimators, Pearson correlation coefficient, and linear regressions (both ways);

**Assess:** mark with squared two-dimensional Mahalanobis distance.

(b) Contingency statistics:

**Learn:** calculate contingency table;

**Derive:** calculate joint, conditional, and marginal probabilities, as well as information entropies;

**Assess:** mark with joint and conditional PDF values, as well as pointwise mutual informations.

3. Multivariate statistics:

These filters all accept multiple requests  $R_i$ , each of which is a set of  $n_i$  variables upon which simultaneous statistics should be computed.

(a) Multi-Correlative statistics:

**Learn:** calculate means and pairwise centered  $M_2$  aggregates;

**Derive:** calculate the upper triangular portion of the symmetric  $n_i \times n_i$  covariance matrix and its (lower) Cholesky decomposition;

**Assess:** mark with squared multi-dimensional Mahalanobis distance.

(b) PCA statistics:

**Learn:** identical to the multi-correlative filter;

**Derive:** everything the multi-correlative filter provides, plus the  $n_i$  eigenvalues and eigenvectors of the covariance matrix;

**Assess:** perform a change of basis to the principal components (eigenvectors), optionally projecting to the first  $m_i$  components, where  $m_i \leq n_i$  is either some user-specified value or is determined by the fraction of maximal eigenvalues whose sum is above a user-specified threshold. This results in  $m_i$  additional columns of data for each request  $R_i$ .

(c) K-Means statistics:

**Learn:** calculate new cluster centers for data using initial cluster centers. When initial cluster centers are provided by the user using an additional input table, multiple sets of new cluster centers are computed. The output metadata is a multiblock dataset containing at a minimum one `vtkTable` with columns specifying the following for each run: the run ID, number of clusters, number of iterations required for convergence, RMS error associated with the cluster, the number of elements in the cluster, and the new cluster coordinates.

**Derive:** calculates the global and local rankings amongst the sets of clusters computed in the learn phase. The global ranking is determined by the error amongst all new cluster centers, while the local rankings are computed amongst clusters sets with the same number of clusters. The total error is also reported;

**Assess:** mark with closest cluster id and associated distance for each set of cluster centers.

In the following sections, we present implementation details on the parallel version of the contingency statistics algorithm, provide a basic user manual thereof, and examine its correctness as well as its parallel speed-up properties.

## 2 Contingency Statistics

In this section, we present a brief refresher on contingency statistics, and their relationship to conditional probabilities, pointwise mutual information, and information entropy.

### 2.1 Contingency Tables

**Definition 2.1.** A *two-way contingency table* is a tabular representation of categorical data, which shows counts or frequencies for particular combinations of values of two discrete random variables  $X_1$  and  $X_2$ . Each cell in the table represents a mutually exclusive pair of  $(X_1, X_2)$  realization.

In this report as well as in the implementation of contingency statistics in [Titan](#), we will make use of counts and not of frequencies, as input data tables comprise realizations of  $(X, Y)$  and thus are directly amenable to occurrence counting.

*Example 2.1.* The sample given in Table 1 of § 1.2.1 can be summarized by the means of the following two-way contingency table:

port,protocol	HTTP	FTP	SMTP
20	0	5	0
25	0	0	5
80	7	0	1
122	0	1	0
1122	1	0	0
8080	1	0	0

which reflects the fact that the categorical variables port and protocol respectively exhibit 6 and 3 different outcomes. Also note that the sum of all counts indeed amounts to the sample size, 21, also called *grand total*. Normalizing the contingency table with respect to the grand total yields frequencies as opposed to counts.

*Remark 2.1.* The concept of contingency table is extended in a straightforward fashion for  $n$  random variables  $(X_i)_{1 \leq i \leq n}$  fashion to *n-way contingency tables*, where counts for each mutually exclusive pair of  $(X_1, \dots, X_n)$  realization are stored. This is what we will mean by *contingency tables* in this report as well as in the implementation.

### 2.2 Joint and Marginal Distributions

A *joint frequency distribution* for the pair  $(X, Y)$  can be obtained from the contingency table of these two variables, by normalizing the table by its grand total. This joint frequency distribution

may be viewed as an *empirical joint probability distribution* for the pair  $(X, Y)$  which, in turn, can be marginalized with respect to either of the random variables, as follows for the marginal distribution of  $X$ :

$$p_X(x) = \sum_{y \in Y(\omega)} p_{X,Y}(x, y)$$

where  $Y(\omega)$  denotes the set of all outcomes of  $Y$ . The corresponding formula for  $p_Y$  is obtained by switching  $X$  and  $Y$ :

$$p_Y(y) = \sum_{x \in X(\omega)} p_{X,Y}(x, y).$$

*Example 2.2.* The contingency table of Example 2.1 results in the following joint and marginal probabilities (with 3 significant digits):

$p_{\text{port,protocol}}$	HTTP	FTP	SMTP	$p_{\text{port}}$
20	0	0.238	0	0.238
25	0	0	0.238	0.238
80	0.333	0	0.0476	0.381
122	0	0.0476	0	0.0476
1122	0.0476	0	0	0.0476
8080	0.0476	0	0	0.0476
$p_{\text{protocol}}$	0.429	0.286	0.286	

The rightmost column and bottom-most row are marginal probabilities while the  $6 \times 3$  matrix in the interior is the joint frequency distribution. This table indicates that, relative to the observed data set,  $(80, \text{HTTP})$  is a highly probable outcome, whereas  $(80, \text{SMTP})$ ,  $(122, \text{FTP})$ ,  $(1122, \text{HTTP})$  and  $(8080, \text{HTTP})$  have a much lower probability of occurrence. Is this sufficient to assert that, for instance,  $(80, \text{HTTP})$  is a normal event, whereas any of the low probability events is abnormal and should be labeled as such? Evidently not, as, for instance,  $(8080, \text{HTTP})$  is known to be a normal pairing—it is just that the HTTP protocol is mostly used over port 80. This is an example of an important principle: a low joint probability does not mean that the variable values are not associated; it may simply be an indication that the co-occurrence is infrequent. Infrequent events may still be strongly associated.

## 2.3 Conditional Probabilities

Recall that, given two random variables  $X$  and  $Y$ , the *conditional probability of  $y$  given  $x$*  is the probability of a realization  $y$  of  $Y$  given the occurrence of a realization  $x$  of  $X$ , and is

$$p_{Y|X}(y|x) := \frac{p_{X,Y}(x, y)}{p_X(x)},$$

provided  $p(x) \neq 0$ , and is undefined otherwise. Similarly,

$$p_{X|Y}(x|y) := \frac{p_{X,Y}(x,y)}{p_X(y)},$$

provided  $p(y) \neq 0$ , and is undefined otherwise. Therefore, the knowledge of the joint (and, as a result, marginal) probability distributions of  $X$  and  $Y$  is all that is needed calculate the conditional probabilities of  $X|Y$  and  $Y|X$ .

*Example 2.3.* Using the joint and marginal distributions found in Example 2.2, one finds the following conditional probabilities for protocol—port and port—protocol (with 3 significant digits):

$p_{\text{protocol} \text{port}}$	HTTP	FTP	SMTP	$p_{\text{port} \text{protocol}}$	HTTP	FTP	SMTP
20	0	1	0	20	0	0.833	0
25	0	0	1	25	0	0	0.833
80	0.875	0	0.125	80	0.778	0	0.167
122	0	1	0	122	0	0.167	0
1122	1	0	0	1122	0.111	0	0
8080	1	0	0	8080	0.111	0	0

First, one readily notices that  $p_{\text{protocol}|\text{port}}$  essentially summarizes the fact that, when the port is known, then so is the protocol, with the exception of port 80. This is useful because, amongst the four aforementioned events with the lowest joint probability (0.0476), there is something fundamentally different about (80, SMTP). On the other hand,  $p_{\text{protocol}|\text{port}}$  does not distinguish between (1122, HTTP) and (20, FTP) but  $p_{\text{port}|\text{protocol}}$  clearly discriminates between these two cases, for it exhibits a much more fine-grained range of probability values. For instance, if one uses the condition  $p_{\text{port}|\text{protocol}} < 0.2$  as a classifier for unlikely associations, then one will retrieve the four low joint probability events. This criterion based on conditional probabilities is more nuanced than any the joint probability table can express for it discriminates between (80, SMTP) and (8080, HTTP) – albeit not in the desired way. So, while  $p_{\text{port}|\text{protocol}}$  appears to convey more information than  $p_{\text{protocol}|\text{port}}$ , it would still be desirable to combine them.

## 2.4 Pointwise Mutual Information

Recall the definition of the pointwise mutual information, which is a measure of association between realizations of discrete random variables. This satisfies the need we have noticed to combine both conditional probabilities:

**Definition 2.2.** The *pointwise mutual information* of a realization  $(x, y)$  of a pair of discrete random variables  $(X, Y)$  is defined as:

$$\text{pmi}(x, y) := \log \frac{p_{(X,Y)}(x, y)}{p_X(x)p_Y(y)},$$

where  $p_X$ ,  $p_Y$ , and  $p_{(X,Y)}$  respectively denote the probability density functions of  $X$ ,  $Y$ , and  $(X,Y)$ , for all possible outcomes of  $X$  and  $Y$ , and setting  $\text{pmi}$  to  $-\infty$  when the joint probability vanishes.

In other words, pointwise mutual information can also be defined as follows:

$$\text{pmi}(x,y) := \log \frac{p_{Y|X}(y|x)p_{X|Y}(x|y)}{p_{(X,Y)}(x,y)},$$

Therefore, mutual independence of the random variables  $x$  and  $y$  makes their pointwise mutual information to vanish everywhere.

*Example 2.4.* Continuing with Example 2.2, we obtain the following pointwise mutual informations (again with 3 significant digits):

	HTTP	FTP	SMTP
20	$-\infty$	1.26	$-\infty$
25	$-\infty$	$-\infty$	1.26
80	0.714	$-\infty$	-0.827
122	$-\infty$	1.26	$-\infty$
1122	0.847	$-\infty$	$-\infty$
8080	0.847	$-\infty$	$-\infty$

In particular, one now has  $\text{pmi}(80, \text{SMTP}) \approx -0.827$ , whereas all of the three other low-probability pairs,  $(122, \text{FTP})$ ,  $(1122, \text{HTTP})$  and  $(8080, \text{HTTP})$ , all have a have a positive pointwise mutual information; in fact, for the last two, the  $\text{pmi}$  is higher than that of the most probable event itself,  $(80, \text{SMTP})$ .

What this indicates is that there is a weak association between port 80 and protocol SMTP, relative to the association between port 80 and protocol HTTP; on the other hand, for the rarely occurring ports (122, 1122, and 8080), there are not enough observations for us to conclude much about their regular associations and thus nothing obvious appears. So, if used as a classifier, pointwise mutual information pinpoints that there is something fundamentally different between the  $(80, \text{SMTP})$  and the other (equally) low-probability outcomes.

## 2.5 Information Entropy

First, recall that:

**Definition 2.3.** The *information entropy* of a discrete random variable  $X$  is defined as:

$$H(X) := - \sum_{x \in X(\omega)} p_X(x) \log_b p_X(x), \tag{2.1}$$

where  $p_X$  is the probability density function of  $X$  and  $b$  is the base of the logarithm; in particular, when  $b = 2$  (resp.  $b = e$ ,  $b = 10$ ), the unit of information entropy is the *bit* (resp. *nat*, *digit*).

Information is used as a measure of the uncertainty associated with a random variable. For instance, a random variable which can only take on a single value, with probability 1, has a zero information entropy. The notion of information entropy is extended to conditional probability as follows:

**Definition 2.4.** The *conditional information entropy* of a discrete random variable  $Y$  given a discrete random variable  $X$  is defined as:

$$H(Y|X) := \sum_{x \in X(\omega)} p_X(x) H(Y|X = x)$$

which, when combined with (2.1), amounts to:

$$H(Y|X) = - \sum_{x \in X(\omega), y \in Y(\omega)} p_{X,Y}(x, y) \log_b p_{Y|X}(y|x),$$

with the same notations as previously used throughout this section.

In everything that follows, we will use  $b = e$ , which appears to be the most commonly used convention to express information entropies.

*Example 2.5.* Ending with our running Example 2.2, we thus obtain the following information entropies:

$$\begin{aligned} H(\text{port}, \text{protocol}) &= 1.62949 \\ H(\text{protocol}|\text{port}) &= H(Y|X) = 0.143531 \\ H(\text{port}|\text{protocol}) &= H(X|Y) = 0.550495. \end{aligned}$$

These values show in particular that the knowledge of the port informs much more about the protocol than the knowledge of the protocol does about the port.

## 3 Parallel Statistics Classes

### 3.1 Implementation Details

The purpose of building a full statistical model in two phases is parallel computational efficiency. In our approach, inter-processor communication and updates are performed only for primary statistics. The calculations to obtain derived statistics from primary statistics are typically fast and simple and need only be calculated once, without communication, upon completion of all parallel updates of primary variables. Data to be assessed is assumed to be distributed in parallel across all processes participating in the computation, thus no communication is required as each process assesses only its own resident data.

Therefore, in the parallel versions of the statistical engines, inter-processor communication is required only for the Learn phase, while both Derive and Assess are executed in an embarrassingly parallel fashion due to data parallelism. This design is consistent with the data-parallel methodology used to enable parallelism within [VTK](#) and [ParaView](#). Because the focus of this report is on the parallel speed-up properties of statistics engines, we will not report on the Derive or Assess phases, as these are executed independently from each other, on a separate process for each part of the data partition. However, because the Derive phase provides the derived quantities to which one is naturally accustomed (e.g., variance as opposed to  $M_2$  aggregate), the numerical results reported here are those that are yielded by the consecutive application of the Learn and then Derive phases.

At this point (September 2009) of the development of scalable statistics algorithms in [Titan](#), the following 6 parallel classes are implemented:

1. `vtkPDescriptiveStatistics`;
2. `vtkPCorrelativeStatistics`;
3. `vtkPMultiCorrelativeStatistics`;
4. `vtkPPCAStatistics`.
5. `vtkPContingencyStatistics`.
6. `vtkPKMeansStatistics`.

Each of these parallel algorithms is implemented as a subclass of the respective serial version of the algorithm and contains a `vtkMultiProcessController` to handle inter-processor communication. Within each of the parallel statistics classes, the Learn phase is the only phase whose behavior is changed (by reimplementing its virtual method) due to the data parallelism inherent in the Derive and Assess phases. The Learn phase of the parallel algorithms performs two primary tasks:

1. Calculate correlative statistics on local data by executing the Learn code of the superclass.

2. If parallel updates are needed (i.e. the number of processes is greater than 1), perform necessary data gathering and aggregation of local statistics into global statistics.

The descriptive, correlative and multi-correlative statistics algorithms perform the aggregation necessary for the statistics which they are computing using the arbitrary-order update and covariance update formulas presented in [P08]. Similarly, the contingency statistics class derives from the bivariate statistics class and implements its own aggregation mechanism for the Learn phase: the global contingency table (cf. 2.1) must be created by aggregating the local contingency tables across all processes participating in the calculation. However, unlike the case for the statistics algorithms which rely on statistical moments (descriptive, correlative, multi-correlative, and PCA), this type of aggregation operation is *not* embarrassingly parallel and, therefore, optimal parallel scale-up should *not* be expected.

## 3.2 Usage

It is fairly easy to use the serial statistics classes of [Titan](#); it is not much harder to use their parallel versions. All that is required is a parallel build of [Titan](#) and a version of MPI installed on your system.

For example, Listing 1 demonstrates how to calculate contingency statistics, in parallel, on 2 pairs of columns of an input dataset `inData` of type `vtkTable*`, with no subsequent data assessment. Requests for each pair of columns of interest are specified by calling `AddColumnPair()`, as is done for all bivariate algorithms. It is assumed here the input data table has at least 3 columns. For information as to the usage of univariate or multi-variate statistics classes, please refer to [PT08].

The examples thus far assume that you have already prepared an MPI communicator, loaded a dataset into the `inData` object, and are running in a parallel environment. It is outside the scope of this report to discuss I/O issues, and in particular how a `vtkTable` can be created and filled with the values of the variables of interest. See [VTK's](#) online documentation for details [[vtk](#)].

In the code example from Listing 1, the `vtkMultiProcessController` object passed to `Foo()` is used to determine the set of processes (which may be a subset of a larger job) among which input data is distributed. [VTK](#) uses subroutines of this form to execute code across many processes. In Listing 2 we demonstrate that, to prepare a parallel controller to execute `Foo()` in parallel using MPI, one must first (e.g. in the main routine) create a `vtkMPIController` and pass it the address of `Foo()`. Note that, when using MPI, the number of processes is determined by the external program which launches the application.

```

void Foo( vtkMultiProcessController* controller, void* arg )
{
    // Use the specified controller on all parallel filters by default:
    vtkMultiProcessController::SetGlobalController( controller );

    // Assume the input dataset is passed to us
    // Also is assume that it has a least 3 columns
    vtkTable* inData = static_cast<vtkTable*>( arg );

    // Create parallel contingency statistics class
    vtkPContingencyStatistics* pcs = vtkPContingencyStatistics::New();

    // Set input data port
    pcs->SetInput( 0, inData );

    // Select pairs of columns (0,1) and (0,2) in inData
    pcs->AddColumnPair( inData->GetColumnName[0], inData->GetColumnName[1] );
    pcs->AddColumnPair( inData->GetColumnName[0], inData->GetColumnName[2] );

    // Calculate statistics with Learn and Derive phases only
    pcs->SetLearn( true );
    pcs->SetDerive( true );
    pcs->SetAssess( false );
    pcs->Update();
}

```

**Listing 1:** A subroutine – that should be run in parallel – for calculating contingency statistics.

```

vtkTable* inData;
vtkMPIController* controller = vtkMPIController::New();
controller->Initialize( &argc, &argv );

// Execute the function named Foo on all processes
controller->SetSingleMethod( Foo, &inData );
controller->SingleMethodExecute();

// Clean up
controller->Finalize();
controller->Delete();

```

**Listing 2:** A snippet of code to show how to execute a subroutine (`Foo()`) in parallel. In reality, `inData` would be prepared in parallel by `Foo()` but is assumed to be pre-populated here to simplify the example.

## 4 Results

Several parallel runs have been executed on Sandia National Laboratories' catalyst computational cluster, which comprises 120 dual 3.06GHz Pentium Xeon compute nodes with 2GB of memory each. This cluster has a Gigabit Ethernet user network for job launch, I/O to storage, and user interaction with jobs, and a 4X Infiniband fabric high-speed network using a Voltaire 9288 InfiniBand switch. Its operating system has a Linux 2.6.17.11 kernel, and its batch scheduling system is the TORQUE resource manager [tor].

Whether the requested processes are distributed to one or two to a node is left to the scheduler to decide. On our system, the default behaviour is to utilize the smallest number of nodes and thus to use two processes per node. All reported results here were done with two processes per node, with the exceptions of the single-process run and one two-process run, marked with a ‡, which for an unexplained reason always failed when run with both processes on the same node.

### 4.1 Algorithm Scalability

In order to assess speed-up independently of the load-balancing scheme, a series of (pseudo-) randomly-generated samples is used. Specifically, input tables are created at run time by generating 2 separate samples of independent pseudo-random variables having a centered normal distribution with the same standard deviation  $\sigma > 0$ . Since our objective is to assess the scalability of the parallel statistics engines only, equally-sized slabs of data are created by each process in order to work with perfectly load-balanced cases. For the same reason, the amount of time needed to create the input data table is excluded from the analysis. In this test, `vtkPContingencyStatistics`, with Learn, Derive, and Assess modes on, is executed on this pair of columns, and various clock times are reported: unlike what was done in the earlier reports for nearly-embarrassingly parallel engine, the wall clock time is decomposed into several pieces in order to better understand the scalability of each component of the engine.

With this synthetic test case, we assess:

1. relative speed-up (at constant total work), and
2. scalability of the rate of computation (at constant work per processor).

#### 4.1.1 Relative Speed-Up

Given a problem of size  $N$  (as measured in our case by sample size), the wall clock time measured to complete the work with  $p$  processors is denoted  $T_N(p)$ . Then, relative speed-up with  $p$  processors is

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

**Table 2.** Relative speed-up (at constant total work), with a total sample size of  $N = 25,600,000$  doubles.

$N/p$	$p$	$\sigma = 5$ (sec. / $S_N(p)$ )	$\sigma = 50$ (sec. / $S_N(p)$ )	$\sigma = 200$ (sec. / $S_N(p)$ )
25,600,000	1	459 / 1.00	700 / 1.00	925 / 1.00
12,800,000	2	234 / 1.96	362 / 1.94	498 / 1.86 <sup>‡</sup>
6,400,000	4	120 / 3.83	185 / 3.79	288 / 3.21
3,200,000	8	59.3 / 7.74	95.0 / 7.38	181 / 5.11
1,600,000	16	30.2 / 15.2	53.1 / 13.2	132 / 7.00
800,000	32	15.1 / 30.4	32.5 / 21.6	111 / 8.33
400,000	64	7.94 / 57.8	24.4 / 28.7	110 / 8.41
200,000	128	4.42 / 104	25.0 / 28.0	112 / 8.26

Evidently, optimal (linear) speedup is attained with  $p$  processors when  $S_N(p) = p$  and, therefore, relative speed-up results for  $S_N$  may be visually inspected by plotting  $S_N$  *versus* the number of processors: optimal speed-up is revealed by a line, the angle bisector of the first quadrant.

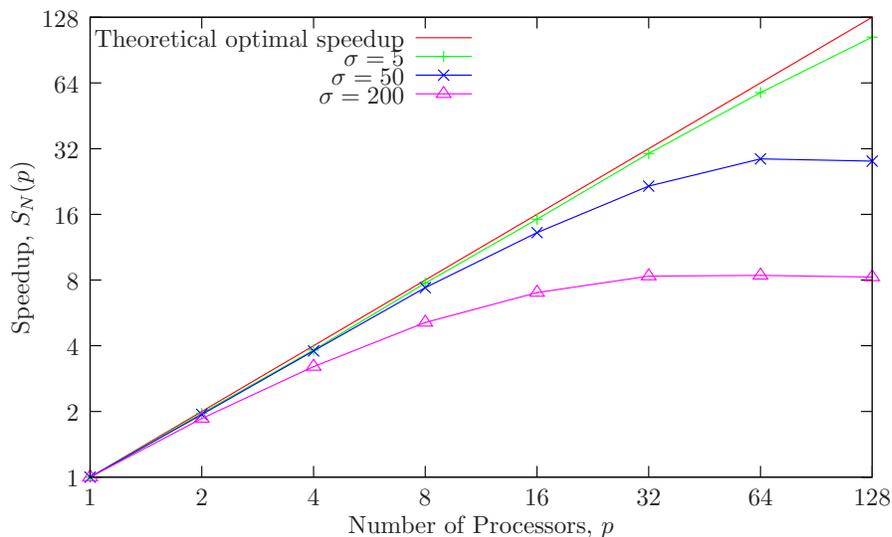
In order to assess relative speed-up, we use a test case that comprise 2 pseudo-random samples of size  $N = 1,000,000$ , generated by rounding 2 independent standard normal variables with the same standard deviation  $\sigma > 0$  to the nearest integer. The values of  $\sigma$  are chosen with increasing values of  $\{5, 50, 200\}$  in order to yield contingency tables with varying sizes, for the probability that any outcome a of bivariate Gaussian draw fall outside the  $[-3\sigma, 3\sigma]^2$  square is only

$$1 - \left( \frac{2}{\sqrt{\pi}} \int_0^{3\sigma} e^{-t^2} dt \right)^2 \approx 0.00539.$$

With the chosen set of standard deviations, the observed contingency tables have sizes of order  $10^3$ ,  $10^5$ , and  $10^6$ , respectively. The overall speedup is a strong function of the contingency table size as the parallel communication costs for these tables are vastly different.

The numbers of processes  $p$  were chosen to be increasing powers of 2, for convenience only: making use of other values did not modify speed-up results.

The wall clock times obtained on `catalyst` are provided in Table 2 and plotted in Figure 3. We observe that with  $\sigma = 5$ , the ensuing local contingency 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 relative speed-up is almost optimal, (within 5%, which may also be due in part to operating system overhead unrelated to the algorithm itself). This remains true until the decreasing amount of work per processor results in a situation where the the contribution of updates and overheads – while small in absolute terms – become noticeable relative to the computation time. With  $\sigma = 5$  and  $N = 1,000,000$ , this trends begins to slightly appear with  $p = 64$ , and further



**Figure 3.** Relative speed-up at constant total work with a total data size of  $N = 25,600,000$  doubles.

with  $p = 128$ , where the speed-up slightly degrades to about 104 (as opposed to a theoretical optimum of 128). At some point, the decreasing amount of work per processor will ultimately result in a situation where communication will dominate computational work, but with  $\sigma = 5$  we have not observed, as speed-up continues beyond 128 processes. On the other hand, with larger values of the standard deviation, and thus with much larger contingency tables to be exchanged between processes, this trend begins to occur much earlier: specifically, with 16 or more processes when  $\sigma = 50$ , and as early as with 2 processes when  $\sigma = 200$ . Moreover, past 64 and 32 processes respectively, no more parallel speed-up is achieved; the Amdahl limit is reached; and speed-down eventually occurs. Theoretically, one could continue increasing  $\sigma$  until no parallel speed-up can be achieved at all, effectively turning the algorithm into a serial implementation.

From this we can draw two important conclusions:

1. This algorithm works well with categorical data, not with (quasi-)continuous data. This was to be expected, for contingency and information statistics are primarily intended for categorical data, not for continuous measurements.
2. One should be careful (as we have always been) with claims that some algorithms would, or would not, be *a priori* amenable to “Map-Reduce” implementations. As this example clearly shows, 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. One should therefore think in terms of a continuum of speed-up properties, for the same algorithm, and thus be wary about “golden bullets” even when they are offered and promoted by prestigious companies.

### 4.1.2 Rate of Computation Scalability

**Table 3.** Rate of computation scalability (at constant load per processor).

$N(p)$	$p$	$\sigma = 5$ (sec. / $R$ )	$\sigma = 50$ (sec. / $R$ )	$\sigma = 200$ (sec. / $R$ )
3,200,000	1	57.8 / 1.00	87.9 / 1.00	127 / 1.00
6,400,000	2	58.3 / 1.98	91.6 / 1.91	152 / 1.67
12,800,000	4	59.3 / 3.91	93.5 / 3.76	166 / 3.06
25,600,000	8	59.3 / 7.80	95.0 / 7.40	181 / 5.61
51,200,000	16	60.3 / 15.3	98.3 / 14.3	203 / 10.0
102,400,000	32	60.7 / 30.5	103 / 27.3	
204,800,000	64	62.5 / 59.2	113 / 49.8	
419,600,000	128	63.1 / 117	128 / 87.9	

The rate of computation is defined as

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

where  $N(p)$ , the sample size, now varies with the number of processors  $p$ . We then measure its scalability by normalizing it with respect to the rate of computation obtained with a single processor, as follows:

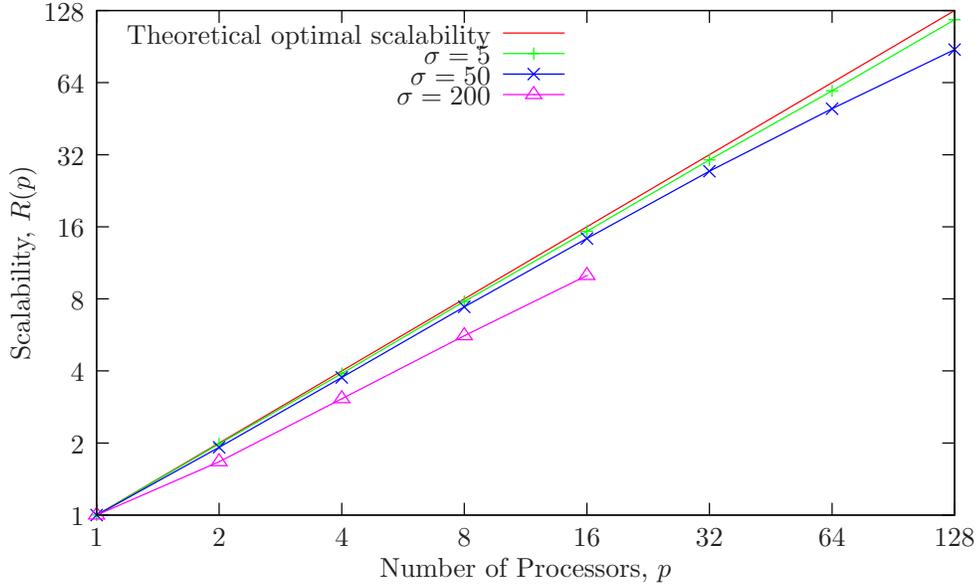
$$R(p) = \frac{r(p)}{r(1)} = \frac{N(p)T_{N(1)}(1)}{N(1)T_{N(p)}(p)},$$

In particular, if the sample size is made to vary in proportion to the number of processors, i.e., if  $N(p) = pN(1)$ , then

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

and thus, optimal (linear) scalability is also attained with  $p$  processors when  $R(p) = p$ . Note that without linear dependency between  $N$  and  $p$ , the latter equality no longer implies optimal scalability. Hence, under the above assumptions, scalability can also be visually inspected, with a plot of  $R$  versus the number of processors, where optimal scalability is also indicated by the angle bisector of the first quadrant.

The rate of computation scalability is now assessed using the same test case as in § 4.1.1, with the difference that, in order to maintain a constant work per processor, increasingly large samples are created: specifically, each data sets contains  $N(p) = np$  doubles, where  $n = 3.2 \times 10^6$  and



**Figure 4.** Rate of computation scalability at constant work per processor of  $N(p)/p = 3,200,000$ .

$p \in \{1, 2, 4, 8, 16, 32, 64, 128\}$  respectively denote the number of sample points per processor and the number of processes.

The wall clock times measured on `catalyst` are given in Table 3 and plotted in Figure 4. When  $\sigma = 5$ , and thus a relatively small contingency table, the algorithm exhibits nearly optimal scalability, (again within  $\pm 5\%$  which does not have to be entirely attributed to the algorithm itself). On the other hand, with large values of  $\sigma$ , as the global contingency table grows the communication costs become much more noticeable, but with  $\sigma = 50$  one continues to observe parallel scalability past 128 processes, with an overall order of about 0.92. However, with the very large input tables generated when  $\sigma = 200$ , the system is not able to allocate enough memory for the communication buffers with more than  $p = 16$  processes, and the execution fails after this point, where an overall order of about 0.83 was observed.

These observations confirm what was already noticed in § 4.1.1. In particular, the failure to scale the rate of computation because of a lack of memory when the tables are enormous confirms that contingency statistics 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). In this case, either a different analysis tool should be used, or the data should be re-quantized prior to contingency statistics analysis. At any rate, the key observation here is again that this engine scales optimally when used for what it was intended, and that its performance degrades as the input data drifts away from the intended category of data.

## 4.2 Algorithm Correctness

In order to assess the algorithm correctness of `vtkPContingencyStatistics`, we have examined the statistical models obtained when both `Learn` and `Derive` options are turned on with a variety of input data sets, both very small (which could then be verified point by point) and very large.

In this report, we do not report on each of these this tests cases, which would present very little interest, other than saying that for those who were too large to for manual verification, the validation method verifies that:

1. the calculated grand total is equal to the sum of the cardinalities of all input tables,
2. the overall cumulative distribution function (CDF) sums to 1 on all processes after the parallel updates, and
3.  $H(X, Y) \geq H(X|Y) + H(Y|X)$  (and  $H(X, Y) = H(X|Y) + H(Y|X)$  when  $X$  and  $Y$  are independent)

on all processes after the parallel updates have been performed.

In addition, for large, pseudo-randomly generated data sets whose statistical properties are known, we also compare the calculated information entropies  $H(X, Y)$ ,  $H(X|Y)$ , and  $H(Y|X)$  with the theoretical values. For example, with the same test cases as in § 4.1.1, we have, in theory,  $H(X|Y) = H(Y|X) = H(X) = H(Y)$  since  $X$  and  $Y$  are independent and identically distributed. We thus use, as a first approximation, the entropies<sup>1</sup> of the corresponding (but not rounded) normal univariate and bivariate distributions:

$$H(\mathcal{N}(\mu, \sigma^2)) = \ln(\sigma\sqrt{2\pi e})$$

and

$$H(\mathcal{N}_2(\mu, \Sigma)) = \ln(2\pi e\sqrt{|\Sigma|})$$

where  $\mu$ ,  $\sigma$ , and  $\Sigma$  respectively denote the mean, standard deviation, and covariance matrix of these distributions. Again because of the independence assumption, in our case  $(X, Y)$  is approximated with a bivariate normal distribution with covariance matrix  $\sigma^2\mathbf{I}_2$ , and thus  $\sqrt{|\Sigma|} = \sigma^2$ . The corresponding theoretical values for  $\sigma \in \{5, 50, 200\}$  are available in Table 4, with 4 significant digits.

Relatively large input sets are used ( $N = 16 \times 10^6$ ), in order to mitigate the risk of statistical bias due to insufficient sampling. The test cases are distributed across  $p = 4$  processes for values of  $\sigma$  in  $\{5, 50, 200\}$ . A comparison between theoretical and computed values, also with 4 significant digits, for one such test run is provided in Table 4, and one can *de visu* notice an excellent agreement between the computed values and the corresponding theoretical values for the approximating continuous random variables. It is also interesting to notice that, as  $\sigma$  increases, the discrepancies

---

<sup>1</sup>Although to be fully rigorous in the case of continuous random variables we should speak of their *differential entropy*, whereas the information entropy as we have defined it in § 2.5 is limited to the case of discrete random variables. The differential entropy of a random variable with PDF  $p$  is defined as  $-\int_{\mathbf{R}} p(x) \log p(x) dx$ .

**Table 4.** Theoretical versus computed entropies of 2 independent, centered normal distributions with standard deviation  $\sigma$ , rounded to the nearest integer. Computed values are for pseudo-random samples of size 16,000,000 distributed across 4 processes.

$\sigma$	$H(X,Y)$		$H(Y X)$		$H(X Y)$	
	theoretical	computed	theoretical	computed	theoretical	computed
5	6.057	6.060	3.028	3.030	3.028	3.030
50	10.66	10.66	5.331	5.326	5.331	5.326
200	13.44	13.38	6.717	6.662	6.717	6.663

between theoretical and computed values increase as well. This is because the input data sets increasingly diverge from ideal Gaussian inputs as the sample size becomes smaller, in relative terms with respect to the standard deviation. Increasing the sample size further results in an even better agreement between theoretical and computed values.

## References

- [BPRT09] J. Bennett, P. Pébay, D. Roe, and D. Thompson. Scalable multi-correlative statistics and principal component analysis with Titan. Sandia Report SAND2009-1687, Sandia National Laboratories, March 2009.
- [P08] P. Pébay. Formulas for robust, one-pass parallel computation of covariances and arbitrary-order statistical moments. Sandia Report SAND2008-6212, Sandia National Laboratories, September 2008.
- [PT08] P. Pébay and D. Thompson. Scalable descriptive and correlative statistics with Titan. Sandia Report SAND2008-8260, Sandia National Laboratories, December 2008.
- [tor] TORQUE Resource Manager. <http://www.clusterresources.com/pages/products/torque-resource-manager.php>.
- [vtk] VTK Doxygen documentation. <http://www.vtk.org/doc/nightly/html>.

## DISTRIBUTION:

1	MS 9159	Philippe P. Pébay, 8963
1	MS 9159	David Thompson, 8963
2	MS 9018	Central Technical Files, 8944
1	MS 0899	Technical Library, 9536





**Sandia National Laboratories**