SANDIA REPORT

SAND2009-7855
Unlimited Release
Printed November 2009

Scalable k-Means Statistics with Titan

Janine Bennett, 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.

Sandia National Laboratories

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

SAND2009-7855
Unlimited Release
Printed November 2009

Scalable k-Means Statistics with Titan

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

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 both the serial and parallel k-means statistics engines. It is a sequel to [PT08], [BPRT09], and [PT09] which studied the parallel descriptive, correlative, multi-correlative, principal component analysis, and contingency engines. The ease of use of the new parallel k-means engine is illustrated by the means of C++ code snippets and algorithm verification is provided. This report justifies the design of the statistics engines with parallel scalability in mind, and provides scalability and speed-up analysis results for the k-means engine.

Acknowledgments

The authors would like to thank:

- Ann Gentile and Jim Brandt of the OVIS project [BGM+09, WTP+08] for part of the support used by P. Pébay in the context of this work,
- Philip Kegelmeyer, Principal Investigator of the Network Grand Challenge LDRD which has been the main source of funding for this work,
- 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
1.3 Input and Output Ports . . . 11
2 Serial k-Means Statistics . . . 13
2.1 General k-Means Clustering Algorithm . . . 13
2.2 Implementation Details . . . 14
3 Parallel Statistics Classes . . . 16
3.1 Implementation Details . . . 16
3.2 Usage . . . 17
4 Results . . . 21
4.1 Algorithm Scalability . . . 21
4.2 Algorithm Correctness . . . 25
References . . . 27

This page intentionally left blank

1 Introduction

This report is a sequel to [PT08], [BPRT09], and [PT09] which focused on the parallel descriptive, correlative, multi-correlative, principal component analysis, and contingency 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.

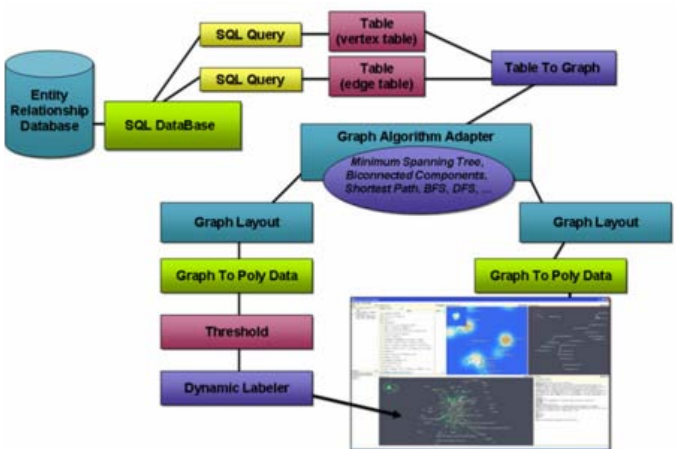

Figure 1. A theoretical application built with Titan.

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.

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 serve 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	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

As an example, consider Table 1. It has 6 observations of 5 variables. If the correlations between A, B, and C, and also between B, C and D are desired, two requests, R_1 and R_2 must be made. The first request R_1 would have columns of interest $\{A,B,C\}$ while R_2 would have columns of interest $\{B,C,D\}$. Calculating covariances for R_1 and R_2 in one pass is more efficient than computing each separately since $\text{cov}(B,B)$, $\text{cov}(C,C)$, and $\text{cov}(B,C)$ are required for both requests but need only be computed once.

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.* [Pó8]). For Table 1 with a request $R_1 = \{B\}$, these values are 6, 1, 11, $4.8\bar{3}$, $68.8\bar{3}$, $159.\bar{4}$, and $1759.819\bar{4}$, respectively.

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. For Table 1 with a request $R_1 = \{B\}$, these additional values are $13.7\bar{6}$, 3.7103, 0.520253, 0.936456, 1.4524, and 1.73616 respectively.

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). Table 1 shows this distance for $R_1 = \{B\}$ in column E.

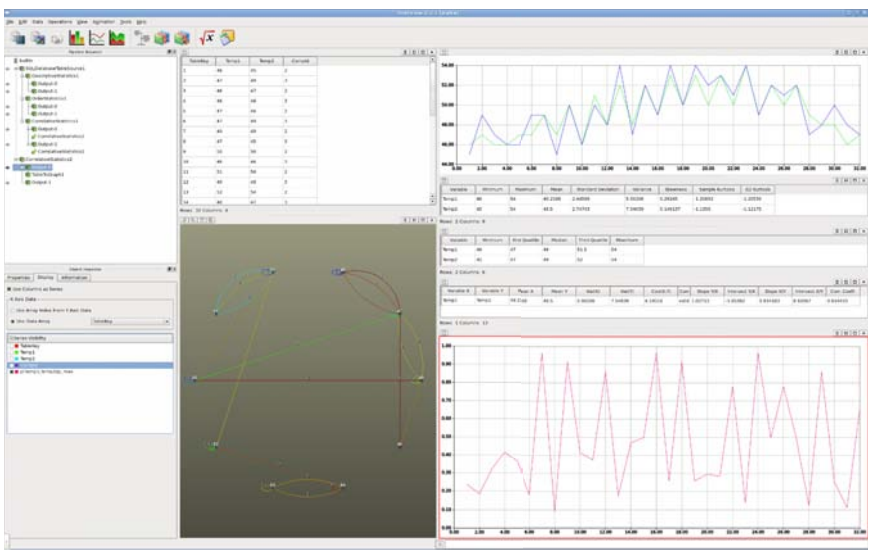

Figure 2. An example utilization of Titan's statistics algorithms in OverView.

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 (1_2 and G_1 estimators), kurtosis (g_2 and G_2 estimators);

Assess: mark with relative deviations (one-dimensional Mahlanobis 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 Mahlanobis 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 Mahlanobis 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: compute optimized set(s) of cluster centers from initial set(s) of cluster centers. In the default case, the initial set comprises the first k observations. However, the user can specify one or more sets of cluster centers (with possibly differing numbers of clusters in each set) via an optional input table, in which case an optimized set of cluster centers is computed for each of the input sets.

Derive: calculate the global and local rankings amongst the sets of clusters computed in the learn phase. The global ranking is the 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 wth closest cluster id and associated distance for each set of cluster centers.

1.3 Input and Output Ports

The statistics algorithms have by default 3 input ports and 3 output ports as follows:

Input Port 0: This port is identified as `vtkStatisticsAlgorithm::INPUT_DATA` and is used for learn data.

Input Port 1: This port is identified as `vtkStatisticsAlgorithm::LEARN_PARAMETERS` and is used for learn parameters (initial guesses, etc.).

Input Port 2: This port is identified as `vtkStatisticsAlgorithm::INPUT_MODEL` and is used for a priori models.

Output Port 0: This port is identified as `vtkStatisticsAlgorithm::OUTPUT_DATA` and mirrors the input data, plus optional assessment columns.

Output Port 1: This port is identified as `vtkStatisticsAlgorithm::OUTPUT_MODEL` and containts any generated model.

Output Port 2: This port is identified as `vtkStatisticsAlgorithm::ASSESSMENT` and is currently not used by any of the statistics algorithms.

By default, the input and output ports are all of type `vtkTable`. There are several exceptions to this default setting, as follows:

- `vtkContingencyStatistics`,
- `vtkKMeansStatistics`,
- `vtkMultiCorrelativeStatistics`, and
- `vtkPCAStatistics`.

Instead, those use `vtkMultiBlockDataSet` for both the `vtkStatisticsAlgorithm::INPUT_MODEL` port and the `vtkStatisticsAlgorithm::OUTPUT_MODEL` port. Moreover, `vtkPCAStatistics` has a fourth input port that is used to provide normalization values.

In the following sections, we present implementation details on the serial and parallel versions of the *k*-means statistics algorithm, provide a basic user manual thereof, provide verification results, and examine the scalability of the parallel implementation.

2 Serial k-Means Statistics

In this section, we present a brief refresher on the serial k-means clustering algorithm, along with usage details for our implementation.

2.1 General k-Means Clustering Algorithm

The term cluster analysis is used to describe a series of unsupervised learning algorithms that partition objects into groups according to a measure of association. With k a positive integer, k-means clustering [Mac67] is one such algorithm that classifies objects into k clusters by minimizing some distance metric between the data and the corresponding cluster centers.

The general algorithm consists of the following steps:

1. Choose the number of clusters, k.
2. Determine k initial cluster centers.
3. Assign each data observation to the nearest cluster center.
4. Recompute the new cluster centers.
5. Repeat the steps 3 and 4 until a prescribed convergence criterion is met (either the maximum number of iterations is reached or the number of cluster assignment changes reaches some minimum value).

This straightforward k-means clustering algorithm is easy to implement; note, however, that it is a heuristic algorithm and therefore that convergence is not guaranteed. Aside from this intrinsic limitation, its main disadvantage is that the value of k must be pre-determined and its results are heavily reliant on the initial cluster centers used. Relatively recent work [ZHD+01, DH04] has proven that the continuous solutions of the discrete k-means clustering membership indicators are the data projections on the principal eigenvectors of the associated covariance matrix. This is equivalent to showing that the cluster centroids are given by spectral expansion of the data covariance matrix truncated at $k-1$ terms.

Another consideration that should be noted when considering k-means clustering, is that it requires that the data being processed has some notion of a mean, which is used to compute the new cluster centers. When data does not have a computable mean, an alternate apporach is to use the k-*medoids* algorithm [Bis06] which uses data observations as cluster centers.

2.2 Implementation Details

2.2.1 Distance Functors and data types

The code can handle a wide variety of data types as it operates on `vtkAbstractArrays` and is not limited to `vtkDataArrays`. A default distance functor that computes the sum of the squares of the Euclidean distance between two objects is provided. The default distance functor can be overridden to use alternative distance metrics.

2.2.2 Initial Cluster Centers

The initial cluster centers used by the algorithm can be specified by the user via a `vtkTable` on port `vtkStatisticsAlgorithm::LEARN_PARAMETERS`. Because the desired value of k is often not known in advance and the results of the algorithm are dependent on the initial cluster centers, we provide a mechanism for the user to test multiple runs or sets of cluster centers within a single call to the Learn phase. The first column of the `vtkStatisticsAlgorithm::LEARN_PARAMETERS` table identifies the number of clusters k in the particular run, while the remaining columns are a subset of the columns contained in the table on port `vtkStatisticsAlgorithm::INPUT_DATA`. We require that all user specified clusters be of the same dimension n and consequently, that the `vtkStatisticsAlgorithm::LEARN_PARAMETERS` table have $n+1$ columns. Due to this restriction, only one request can be processed for each call to the Learn phase and subsequent requests are silently ignored (recall that the requests identify the variables that are to be used when performing the computations).

When the user does not supply an initial set of clusters, then the first `DefaultNumberOfClusters` input data observations are used as initial cluster centers and a single run is performed.

2.2.3 Engine Output

The output of the Learn and Derive phases are the first and second `vtkTables` respectively in the `vtkMultiBlockDatSet` that is output on port `vtkStatisticsAlgorithm::OUTPUT_MODEL`. The `vtkTable` output for the the Learn phase contains a row for each cluster center in each run that includes the following column entries:

Run ID: A unique ID specifying the run number.

K: Number of clusters in the run.

Iterations: Number of iterations prior to termination of the run.

Error: Sum of squared Euclidean distances from each observation in the cluster to the cluster center. Alternative metrics can be used by overriding the distance functor.

Cardinality: Number of elements in cluster.

Cluster center: The optimized cluster center coordinates are actually a series of column entries, one per variable requested by the user.

The `vtkTable` output for the Derive phase contains a row for each run that includes the following column entries:

Run ID: A unique ID specifying the run number.

K: Number of clusters in the run.

Iterations: Number of iterations prior to termination of the run.

Total Error: Sum of squared Euclidean distance from each observation to its nearest cluster center. Alternative metrics can be used by overriding the distance functor.

Local Rank: Ranking within runs with the same number of clusters (the run with the smallest error is ranked 1).

Global Rank: Ranking within all runs (the run with the smallest error is ranked 1).

Using the cluster centers from the first `vtkTable` of the `vtkMultiBlockDataSet` provided on port `vtkStatisticsAlgorithm::INPUT_MODEL` (i.e. the output of the Learn phase), the Assess phase outputs a `vtkTable` on `vtkStatisticsAlgorithm::OUTPUT_DATA` that contains the closest cluster center and associated distance for each run appended to the `vtkTable` input on `vtkStatisitcsAlgorithm::INPUT_DATA`.

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 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 parallelism methodology used to enable parallelism within VTK, most notably in 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 (November 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 or by reimplementing virtual methods that are called by the Learn phase) 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 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 [PÓ8]. Because the PCA statistics class derives from the multi-correlative statistics algorithm and inherits its Learn phase, a static method is defined within the parallel multi-correlative statistics algorithm to gather all necessary statistics, cf. [BPRT09] for details. As explained in the aforementioned references, all those parallel classes exhibit optimal parallel speed-up properties. Similarly, the contingency statistics class derives from the bivariate statistics class and implements its own aggregation mechanism for the Learn phase. However, unlike the other statistics algorithms which rely on statistical moments (descriptive, correlative, multi-correlative, PCA, and k-means), this aggregation operation is, in general, not embarrassingly parallel and, therefore, optimal parallel scale-up is not observed when this class is not used outside of its intended domain of applicability, as explained in [PT09].

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 k-means statistics, in parallel, on each column of an input set `inData` of type `vtkTable*` with an associated set of input parameters and no subsequent data assessment. It is assumed here the input data table has at least 3 columns.

In Listing 1, requests for columns of interest are specified by calling `SetColumnStatus()` multiple times to identify the variables to be used, followed by a call to `RequestSelectedColumns()`. The k-means statistics classes are currently the only statistics classes limited to one request per call to the Learn phase, with additional requests being silently ignored. It is important to note that these requests only identify the variables used when performing k-means clustering, they do not impact the number of runs (i.e. the number of sets of clusters) that are computed during a particular call to Learn. The number of sets of runs is inferred from the `vtkTable` that is input in the `vtkStatistics::LEARN_PARAMETERS` port. When no table is provided the number of runs is 1 and the first k data points are used as initial cluster centers.

All multivariate algorithms specify requests using a number of calls to `SetColumnStatus()` followed by a call to `RequestSelectedColumns()`, and most multivariate algorithms allow for multiple requests. Consider the example from §1.2.1 and Table 1 where 2 requests are mentioned: $\{A,B,C\}$ and $\{B,C,D\}$. The code snippet in Listing 2 shows how to queue these requests for a `vtkPPCAStatistcs` object. Note that `SetColumnStatus()` should be called to turn off any previously-selected columns that are no longer of interest.

Bivariate and unvariate algorithms have slightly different usage patterns. Bivariate algorithms call `AddColumnPair()` and univariate algorithms call `AddColumn()` for each set of requests.

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 3 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 and
  // assume that it has at least 3 columns
  vtkTable* inData = static_cast<vtkTable*>( arg->inData );

  // Assume the input dataset is passed to us and
  // assume that it has at least 3 columns
  vtkTable* inLearnParameters = static_cast<vtkTable*>( arg->inInitClusters );

  // Create parallel k-means statistics class
  vtkPKMeansStatistics* pks = vtkPKMeansStatistics::New();

  // Set input data ports
  pks->SetInput( vtkStatisticsAlgorithm::INPUT_DATA, inData );
  pks->SetInput( vtkStatisticsAlgorithm::LEARN_PARAMETERS, inLearnParameters );

  // The first and third columns from inData are used to perform clustering
  pks->SetColumnStatus( inData->GetColumnName[0], 1 );
  pks->SetColumnStatus( inData->GetColumnName[2], 1 );
  pks->RequestSelectedColumns();

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

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

```
vtkPPCAStatistics* pps = vtkPKMeansStatistics::New();

// Turn on columns of interest
pps->SetColumnStatus( "A", 1 );
pps->SetColumnStatus( "B", 1 );
pps->SetColumnStatus( "C", 1 );
pps->RequestSelectedColumns();

// Columns A, B, and C are still selected, so first we turn off
// column A so it will not appear in the next request.
pps->SetColumnStatus( "A", 0 );
pps->SetColumnStatus( "D", 1 );
pps->RequestSelectedColumns();
```

Listing 2: An example of requesting multiple multi-variate analyses.

```
struct StatisticsArgs
{
  vtkTable* inData;
  vtkTable* inInitClusters;
};

StatisticsArgs* args;

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

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

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

Listing 3: A snippet of code to show how to execute a subroutine (`Foo()`) in parallel. In reality, `inData` and `inInitClusters` would be prepared in parallel by `Foo()` but is assumed to be pre-populated here to simplify the example.

4 Results

The 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 behavior is to utilize the smallest number of nodes and thus to use two processes per 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 6 separate samples from 6 independent pseudo-random variables, each drawn from 8 different Gaussian distributions with unit standard deviation, distribution number i having mean $7i$. The result is a $6-$dimensional data set consisting of eight different clusters. 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, `vtkPKMeansStatistics`, with Learn, Derive, and Assess modes on, is executed on the six columns and the corresponding wall clock time is reported.

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 processes is denoted $T_N(p)$. Then, relative speed-up with p processes is

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

We achieve near optimal (linear) speedup with p processes when $S_N(p) = p$ and, therefore, relative speed-up results for $S_N(p)$ may be visually inspected by plotting $S_N(p)$ *versus* the number of processes: optimal speed-up is revealed by a line, the angle bisector of the first quadrant. The values of p were chosen to be increasing powers of 2, for convenience only, and making use of other values did not modify speed-up results. The results obtained on `catalyst` are provided in Tables 2 and plotted in Figure 3. 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

Table 2. Relative speed-up (at constant total work), with a total sample size of $N = 1000000$.

N/p	p	seconds uninitialized	$S_N(p)$ uninitialized	seconds initialized	$S_N(p)$ initialized
1,000,000	1	2044.1	1.0	542.4	1.0
500,000	2	1033.6	2.0	273.1	2.0
250,000	4	528.2	3.9	136.7	4.0
125,000	8	263.2	7.8	69.0	7.9
62,500	16	120.2	17.0	35.6	15.2
31,250	32	68.8	29.7	17.4	31.2
15,625	64	31.2	65.5	9.3	58.3

amenable to a data parallel implementation. As can be seen, the measured relative speed-up is optimal (within $\pm 5\%$ fluctuations attributable to OS jitter and such). It should be noted that in some instances the uninitialized runs appear to scale super-linearly. This can be attributed to the uninitialized runs terminating early at a locally optimal solution.

4.1.2 Rate of Computation Scalability

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

$N(p)$	p	seconds uninitialized	$R(p)$ uninitialized	seconds initialized	$R(p)$ initialized
250,000	1	503.5	1.0	133.3	1.0
500,000	2	513.0	2.0	133.9	2.0
1,000,000	4	492.5	4.1	137.2	3.9
2,000,000	8	526.9	7.7	139.7	7.6
4,000,000	16	495.4	16.3	141.7	15.1
8,000,000	32	514.5	31.3	142.7	29.9
16,000,000	64	574.4	56.1	141.8	60.2

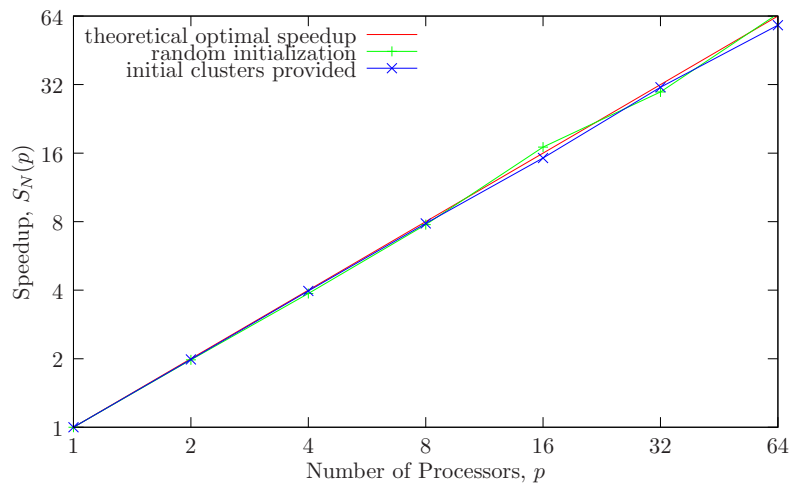

Figure 3. Relative speed-up at constant total work with a total data size of $N = 1,000,000$.

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 processes p. We then measure its scalability by normalizing it with respect to the rate of computation obtained with a single process, 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 processes, 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 processes 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 $R(p)$ *versus* the number of processes, where optimal scalability is also indicated by the angle bisector of the first quadrant.

In order to assess rate of computation scalability (at constant work per process), a series of increasingly large samples was generated. Each sample consisted of 6 variables, each with np observations, with $p \in \{1,2,4,8,16,32,64\}$ denoting the number of processes and $n = 250000$ denoting the number of sample points per process. Note that the scheduler is left to decide whether one or two cores per node are occupied by the p processes. Forcing all cluster nodes to utilize either exactly one, or exactly two of their cores did not result in a measurable difference. Corresponding wall clock times measured on `catalyst` are given in Table 3 and plotted in Figure 4.

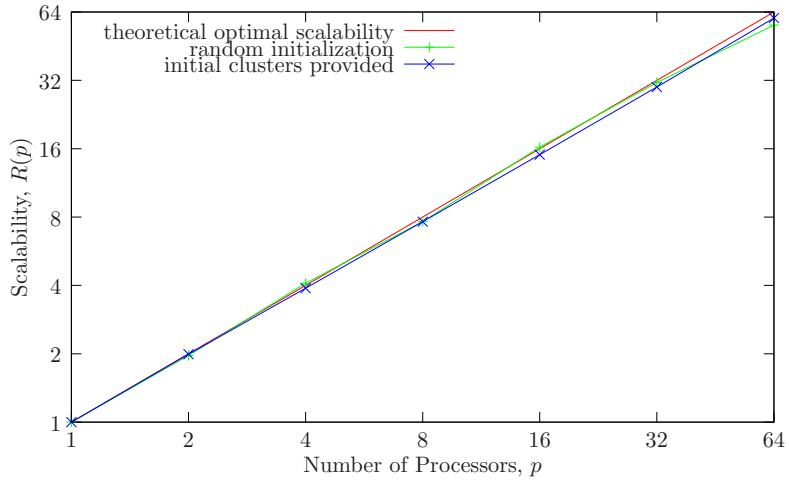

Figure 4. Rate of computation scalability at constant work per process of $N(p)/p = 250,000$.

These clearly exhibit optimal scalability (again within $\pm 5\%$ fluctuations attributable to OS jitter and such), 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. Again, this can be attributed to the uninitialized runs terminating early at a locally optimal solution.

4.2 Algorithm Correctness

In order to assess algorithm correctness, we make use of the same type of test cases as § 4.1, except that only 5 Gaussian distributions are utilized, and that they are only 2-dimensional so that the numerical results obtained by the `vtkPKMeansStatistics` class can be visually inspected. Also, the mean of distribution number i is decreased from $7i$ to $6i$.

Relatively large input sets are used ($n = 100,000$), in order to mitigate the risk of statistical bias due to insufficient sampling. We compare the results obtained with the Learn mode of the statistical engines to the known 5 cluster centers used to generate the input. This comparison is done by simple visual inspection of the numerical results: Figure 5 (a), (b), and (c) show the results of the Learn mode of the k-means statistics engine for $k = 4$, 5, and 6 respectively. The runs that generated these images were not provided with initial cluster centers as input, and thus the engine initialized the cluster centers to the first k observations of the input data for each run. As such, all cluster centers were initialized to observations drawn from the 2-dimensional distribution centered at the origin.

Figures (d), (e), and (f) also show the results of the Learn mode of the k-means engine for $k = 4$, 5, and 6 respectively, but with initial cluster centers chosen using outside knowledge to illustrate the difference in results when "good" initial centers are available – the initial centers are observations drawn from the distributions used to generate the input dataset but each center is an observation from a different distribution. Together these images highlight and confirm the fact that the k-means algorithm does not converge to an optimally global solution and is sensitive to initial input values. However, the algorithm is particularly powerful when *a priori* knowledge of the input data can be leveraged.

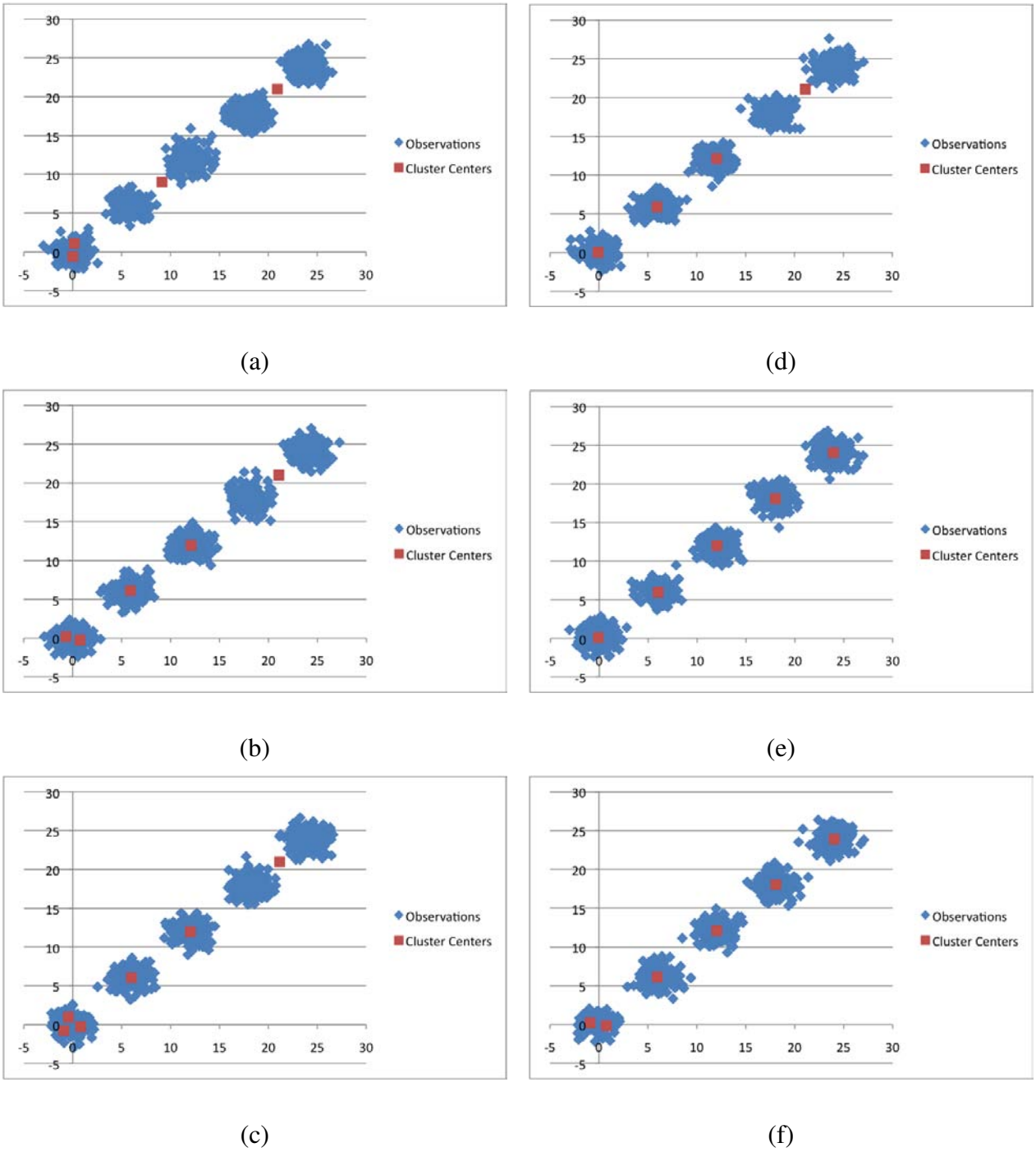

Figure 5. Results of the Learn mode of the k-means engine for $k = 4$ (top), 5 (middle), and 6 (bottom). On the left, no initial cluster centers were provided, whereas on the right, each initial guess was an observation drawn from a different distribution used to generate the training data.

References

[BGM+09] J. Brandt, A. Gentile, J. Mayo, P. Pébay, D. Roe, D. Thompson, and M. Wong. *OVIS 2.0 User's Guide*. Sandia National Laboratories, April 2009.

[Bis06] C. M. Bishop. *Pattern Recognition and Machine Learning.* Springer, 2006.

[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.

[DH04] Chris H. Q. Ding and Xiaofeng He. K-means clustering via principal component analysis. In Carla E. Brodley, editor, *ICML*, volume 69 of *ACM International Conference Proceeding Series*. ACM, 2004.

[Mac67] J. MacQueen. Some methods for classification and analysis of multivariate observations. In L. M. LeCam and J. Neyman, editors, *Proc. of the 5th Berkeley Symp. on Mathematics Statistics and Probability*, 1967.

[PÓ8] 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.

[PT09] P. Pébay and D. Thompson. Parallel contingency statistics with Titan. Sandia Report SAND2009-6006, Sandia National Laboratories, September 2009.

[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.

[WTP+08] M. H. Wong, D. C. Thompson, P. Pébay, J. R. Mayo, A. C. Gentile, B. J. Debusschere, and J. M. Brandt. OVIS-2: A robust distributed architecture for scalable RAS. In *Proc. 22nd IEEE International Parallel & Distributed Processing Symposium*, Miami, FL, April 2008.

[ZHD+01] Hongyuan Zha, Xiaofeng He, Chris H. Q. Ding, Ming Gu, and Horst D. Simon. Spectral relaxation for K-means clustering. In Thomas G. Dietterich, Suzanna Becker, and Zoubin Ghahramani, editors, *Advances in Neural Information Processing Systems 14 [Neural Information Processing Systems: Natural and Synthetic, NIPS 2001, December 3-8, 2001, Vancouver, British Columbia, Canada]*, pages 1057–1064. MIT Press, 2001.

DISTRIBUTION:

1	MS 9159	Janine Bennett, 8963
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