pGrid: Parallel Grid-Based Data Stream Clustering with MapReduce

05/05/2009

Prepared by
Xin Sun
Yu (Cathy) Jiao, Ph.D.
DOCUMENT AVAILABILITY

Reports produced after January 1, 1996, are generally available free via the U.S. Department of Energy (DOE) Information Bridge.

*Web site* [http://www.osti.gov/bridge](http://www.osti.gov/bridge)

Reports produced before January 1, 1996, may be purchased by members of the public from the following source.

National Technical Information Service
5285 Port Royal Road
Springfield, VA 22161
*Telephone* 703-605-6000 (1-800-553-6847)
*TDD* 703-487-4639
*Fax* 703-605-6900
*E-mail* info@ntis.gov
*Web site* [http://www.ntis.gov/support/ordernowabout.htm](http://www.ntis.gov/support/ordernowabout.htm)

Reports are available to DOE employees, DOE contractors, Energy Technology Data Exchange (ETDE) representatives, and International Nuclear Information System (INIS) representatives from the following source.

Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831
*Telephone* 865-576-8401
*Fax* 865-576-5728
*E-mail* reports@osti.gov
*Web site* [http://www.osti.gov/contact.html](http://www.osti.gov/contact.html)

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, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents 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 or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.
Applied Software Engineering Research Group

pGrid: Parallel Grid-Based Data Stream Clustering with MapReduce

Xin Sun
Yu (Cathy) Jiao, Ph.D.

Date Published: 05/05/2009

Prepared by
OAK RIDGE NATIONAL LABORATORY
Oak Ridge, Tennessee 37831-6283
managed by
UT-BATTELLE, LLC
for the
U.S. DEPARTMENT OF ENERGY
under contract DE-AC05-00OR22725
# CONTENTS

<table>
<thead>
<tr>
<th>CONTENTS</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONTENTS</td>
<td>iii</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>v</td>
</tr>
<tr>
<td>abstract</td>
<td>1</td>
</tr>
<tr>
<td>1. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>2. RELATED WORK</td>
<td>3</td>
</tr>
<tr>
<td>2.1 DATA STREAM CLUSTERING</td>
<td>3</td>
</tr>
<tr>
<td>2.1.1 Single-Phase Schemes</td>
<td>3</td>
</tr>
<tr>
<td>2.1.2 Two-Phase Schemes</td>
<td>3</td>
</tr>
<tr>
<td>2.2 D-STREAM: A TWO-PHASE GRID-BASED CLUSTERING ALGORITHM</td>
<td>4</td>
</tr>
<tr>
<td>2.2.1 Definitions</td>
<td>4</td>
</tr>
<tr>
<td>2.2.2 The D-Stream Algorithm</td>
<td>4</td>
</tr>
<tr>
<td>2.3 THE MAPREduce FRAMEWORK</td>
<td>5</td>
</tr>
<tr>
<td>3. THE PGRID ALGORITHM</td>
<td>7</td>
</tr>
<tr>
<td>3.1 BASIC DEFINITIONS</td>
<td>7</td>
</tr>
<tr>
<td>3.2 THE PGRID WORKFLOW</td>
<td>7</td>
</tr>
<tr>
<td>3.2.1 The Online Component</td>
<td>8</td>
</tr>
<tr>
<td>3.2.2 The Offline Component</td>
<td>8</td>
</tr>
<tr>
<td>3.3 COMPUTATIONAL COMPLEXITY ANALYSIS</td>
<td>13</td>
</tr>
<tr>
<td>4. PERFORMANCE EVALUATION</td>
<td>14</td>
</tr>
<tr>
<td>4.1 DATASET</td>
<td>14</td>
</tr>
<tr>
<td>4.2 EXPERIMENT SETUP</td>
<td>15</td>
</tr>
<tr>
<td>4.3 PERFORMANCE METRICS</td>
<td>15</td>
</tr>
<tr>
<td>4.4 EXPERIMENTAL RESULTS</td>
<td>16</td>
</tr>
<tr>
<td>4.4.1 Overall Performance</td>
<td>16</td>
</tr>
<tr>
<td>4.4.2 Sensitivity</td>
<td>18</td>
</tr>
<tr>
<td>4.4.3 Parameter Influence on the Number of Clusters</td>
<td>18</td>
</tr>
<tr>
<td>5. CONCLUSION AND FUTURE WORK</td>
<td>20</td>
</tr>
<tr>
<td>5.1 CONCLUSION</td>
<td>20</td>
</tr>
<tr>
<td>5.2 FUTURE WORK</td>
<td>20</td>
</tr>
<tr>
<td>5.2.1 Efficiency Test</td>
<td>20</td>
</tr>
<tr>
<td>5.2.2 An Alternative Performance Metric</td>
<td>20</td>
</tr>
<tr>
<td>6. REFERENCE</td>
<td>21</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fig. 1</td>
<td>The D-stream algorithm [1]. ................................................................. 4</td>
</tr>
<tr>
<td>Fig. 2</td>
<td>The input and output of MapReduce in the term frequency count example. ...... 5</td>
</tr>
<tr>
<td>Fig. 3</td>
<td>A term frequency count example. .............................................................. 6</td>
</tr>
<tr>
<td>Fig. 4</td>
<td>The MapReduce framework. ......................................................................... 6</td>
</tr>
<tr>
<td>Fig. 5</td>
<td>The workflow of the pGrid algorithm. .......................................................... 7</td>
</tr>
<tr>
<td>Fig. 6</td>
<td>A 2-D example of the MapReduce clustering. .............................................. 10</td>
</tr>
<tr>
<td>Fig. 7</td>
<td>The MapReduce implementation. ................................................................. 10</td>
</tr>
<tr>
<td>Fig. 8</td>
<td>The 1st MapReduce process of the 2-D Example. ......................................... 11</td>
</tr>
<tr>
<td>Fig. 9</td>
<td>The 2nd MapReduce process of the 2-D example. ........................................ 12</td>
</tr>
<tr>
<td>Fig. 10</td>
<td>The 3rd MapReduce process of the 2-D example. ......................................... 12</td>
</tr>
<tr>
<td>Fig. 11</td>
<td>A 4-D pGrid clustering example. ............................................................... 13</td>
</tr>
<tr>
<td>Fig. 12</td>
<td>Basic features of individual TCP connections. .......................................... 14</td>
</tr>
<tr>
<td>Fig. 13</td>
<td>Content features within a connection suggested by domain knowledge. .......... 15</td>
</tr>
<tr>
<td>Fig. 14</td>
<td>Traffic features computed using a two-second time window. ..................... 15</td>
</tr>
<tr>
<td>Fig. 15</td>
<td>Average SSQ of pGrid and D-Stream. ......................................................... 16</td>
</tr>
<tr>
<td>Fig. 16</td>
<td>Average cluster purity of pGrid and D-Stream. .......................................... 16</td>
</tr>
<tr>
<td>Fig. 17</td>
<td>Number of clusters generated by pGrid and D-Stream compared with the ground truth. ................................................................. 17</td>
</tr>
<tr>
<td>Fig. 18</td>
<td>Average SSQ of pGrid with different len values. ........................................ 17</td>
</tr>
<tr>
<td>Fig. 19</td>
<td>Cluster purity of pGrid with different len values. ...................................... 18</td>
</tr>
<tr>
<td>Fig. 20</td>
<td>Number of clusters generated by pGrid with different len values. ............... 18</td>
</tr>
<tr>
<td>Fig. 21</td>
<td>Number of clusters generated by pGrid with different len at t=60. ............... 19</td>
</tr>
<tr>
<td>Fig. 22</td>
<td>Number of clusters generated by GridPal with different dl at t=60. .............. 19</td>
</tr>
</tbody>
</table>
ABSTRACT

The ability to quickly understand large volumes of data stream is of great interest to industry, academia, and government. In addition, it is a critical challenge in many application domains such as Internet news feed analysis, network intrusion detection systems, and analysis of scientific data. Real-time pattern recognition from the data streams is a promising approach to this challenge; however, due to the sheer speed and size of the data, data points must often be analyzed on the fly. The one-pass-through requirement and the lack of efficient clustering algorithms to identify and distinguish patterns limits the power and scalability of this approach. To overcome these limitations, in this report we introduce a novel clustering algorithm called pGrid. It is a parallel grid-based data stream clustering method that uses the MapReduce framework to quickly find patterns in the data streams. We present our algorithm with detailed examples and an in-depth performance evaluation against the D-Stream algorithm. Our evaluation results show that our algorithm performs comparably to the state-of-the-art methods in quality of the clustering results, while it is significantly more efficient and scalable.

1. INTRODUCTION

Clustering high-dimensional data in real time is a challenging problem with ample applications such as network intrusion detection systems (NIDSs), Internet news feed analysis, weather monitoring, emergency response systems, e-business, telecommunication, distributed sensor networks, and real-time analysis of scientific data. In these applications, large volumes of high-dimensional data flow to data collection centers at a high data rate. The processing of this data needs to be highly efficient in order to achieve real time response. For instance, a network intrusion detection system receives network traffic and the system logs at the rate of tens of gigabits per second. One way to capture abnormal activities from this massive amount of data is through efficient data clustering in real time. As the data rate and data size increase, parallel clustering algorithms become much more desirable since they are scalable and can process data in a timely manner.

An efficient parallel clustering algorithm for data stream must address two challenges: i) the algorithm must form clusters without knowledge of the entire data set in one pass over each data point, and ii) the algorithm must be able to manage incremental updates of the clusters as new data arrive. Most of the existing parallel data clustering algorithms are limited to variations of the k-means algorithm [1,2,18], which cannot handle clusters of arbitrary shape very well.

Chen and Tu proposed D-Stream [6], a grid-based partitional clustering algorithm for data streams. It outperforms many existing algorithms for data streams. However, it is not scalable and hence is not well suited for high-volume data streams. We believe a parallel grid-based algorithm can provide unprecedented accuracy and efficiency when clustering data streams. In addition, because of its scalability, such an algorithm meets the demands of high-bandwidth real-time applications such as NIDSs.

The efficiency of the Google search engine in retrieving information across a large network of computer clusters is largely attributed to the employment of the MapReduce framework [8]. In this work,
we apply the concept of MapReduce to data clustering and propose pGrid, a novel parallel grid-based clustering algorithm that is capable of handling dynamic data streams.

The rest of the report is organized as follows: Section 2 introduces related work in the areas of data clustering and the MapReduce framework. Section 3 discusses the details of the pGrid algorithm. Section 4 presents the performance evaluation, and finally, Section 5 concludes our discussion.
2. RELATED WORK

In this section, we first review some of the existing data stream clustering literature. We then provide details of the D-Stream algorithm, which has much in common with the pGrid algorithm. Finally, we introduce the MapReduce framework and its Java realization called Hadoop.

2.1 DATA STREAM CLUSTERING

Clustering methods can be broadly classified into two main categories: partitional clustering algorithms and hierarchical clustering algorithms. Partitional clustering methods can be further divided into distance-based and density-based solutions. Distance-based methods are those that need to calculate distances (either pairwise or to the centroid) of data points before clustering. K-means [1,2,21], similarity-histogram-based [13, 15], and Expectation Maximization (EM) [22] are examples of distance-based methods. As the name suggests, density-based methods form clusters by using data point density. Some examples include grid-based [6, 16], micro-cluster-based [1, 2, 7], kernel density estimation [18, 19, 20, 14], and wavelet density estimation [11, 12] approaches. These solutions primarily target static data sets.

In contrast to static data, a data stream is “a real-time, continuous, ordered (implicitly by arrival time or explicitly by timestamp) sequence of items. It is impossible to control the order in which items arrive, nor is it feasible to locally store a stream in its entirety [9].” There are two major categories of approaches to data stream clustering: single-phase schemes and two-phase schemes.

2.1.1 Single-Phase Schemes

Single-phase schemes can be viewed as a time window based version of static data clustering [10, 17]. They first partition data streams into segments, and then conduct data clustering on each of these segments. In other words, single-phase schemes follow a divide-and-conquer strategy.

Single-phase clustering schemes enable data stream clustering, but they are not true real-time clustering solutions. Moreover, they cannot capture the evolving characteristics of a data stream since they assign equal weights on outdated and recent data [6]. If the data stream evolves over time, these algorithms view it as several segments of static data: they cannot discover time-dependent patterns [6].

2.1.2 Two-Phase Schemes

A two-phase scheme consists of an online component and an offline component [1, 2, 3, 4, 19]. The online component processes the raw data stream and generates a statistical summary of the data stream. The offline component is triggered periodically and uses the statistical summary calculated by the online component to generate clusters. Two-phase data stream clustering schemes are more time efficient than single-phase schemes because the clustering process, which is the most time-consuming process, is only executed periodically. As a result, two-phase schemes have been the most widely used data stream clustering methods in recent years.

A grid-based clustering algorithm falls in the class of the two-phase schemes: it first partitions the data space into discretized grids. While the data stream is active, the online component projects data onto the grids. The offline component is triggered periodically to cluster the grids based on the extracted features, or characteristic vectors, of the grids captured in the online phase. D-Stream is a grid-based, two-phase clustering algorithm [6]. We discuss the details of this algorithm in the next section.
2.2 D-STREAM: A TWO-PHASE GRID-BASED CLUSTERING ALGORITHM

2.2.1 Definitions

We summarize some of the important notations and definitions introduced by Chen and Tu’s D-Stream as follows [6]:

- \( d \): the dimension of the data.
- \( g \): a grid and the smallest unit for clustering.
- \( len \): the number of partitions on each dimension, i.e., the number of grids in a dimension.
- \( \beta \): a constant used to define sporadic grids (see below).
- \( \text{Density Coefficient} = \lambda(\text{tc} - \text{ta}) \), where \( \lambda (0 < \lambda \leq 1) \) is the decay factor, \( \text{tc} \) is the current time, and \( \text{ta} \) is the time when the data point arrived.
- \( D \): Grid Density, \( D \), is the sum of the density coefficients of all data points. It is updated every gap time interval. \( D_m \) and \( D_l \) represent the upper and lower bound of density threshold set by the algorithm.
- \( \text{Dense Grids} \): grids that have \( D \geq D_m \).
- \( \text{Transitional Grids} \): grids that have \( D_l \leq D \leq D_m \).
- \( \text{Sparse Grids} \): grids that have \( D \leq D_l \).
- \( \text{Sporadic Grids} \): sparse grids that have very few data and can be removed before clustering.

Characteristic Vector: It is a tuple \((t_g, t_m, D, \text{label}, \text{status})\), where \( t_g \) is the last time \( g \) was updated, \( t_m \) is the last time \( g \) was removed as a sporadic grid, \( D \) is the grid density at the last update, \( \text{label} \) is the class label of the grid, and \( \text{status}=\{\text{SPORADIC, NORMAL}\} \).

2.2.2 The D-Stream Algorithm

![Image](image)

Fig. 1 The D-stream algorithm [1].

The online component of D-Stream first projects data points onto corresponding grids, and then updates the characteristic vector. The offline component combines dense and transitional grids which are neighboring grids into clusters – neighboring grids are adjacent grids on the \( k \)th dimension and share the same index values on all other \( d-1 \) dimensions. The offline process is iterative until the clustering result converges.
More specifically, D-Stream assumes that the model has a discrete time step, where the time stamps are integers starting from 0. When the algorithm first starts, we need to initialize an empty hash table called grid_list. While the data stream is active, we first read a new data record and determine the density grid \( g \) that contains it. If \( g \) is not in the current grid_list, we insert \( g \) into grid_list; otherwise, we keep the current grid_list. Then we update the characteristic vector of \( g \). If time equals gap, which is a pre-defined parameter, we call the function initial_clustering(grid_list). Otherwise, if time is a multiple of gap, then we detect and remove sporadic grids from grid_list and call the function of adjust_clustering(grid_list). The definition of sporadic grid is in Section 3.1. time is increased by 1 each time we finish the processing of one data record.

Compared with K-means based methods, D-Stream has the advantages of being able to find clusters of arbitrary shapes, can handle noise well, and does not require any prior knowledge of the number of clusters. In addition, D-Stream partitions the data space into discretized grids and summarizes the characteristics of the data stream. By doing so, the amount of information being maintained for clustering is significantly reduced. Nevertheless, the D-Stream algorithm has some disadvantages. First, it sequentially examines neighboring grids on all dimensions, which is very time-consuming. Second, it does not address the sensitivity of the resulting number of clusters to different parameters, which may have significant impact on the class purity measures.

In this report, we propose a novel parallel grid-based approach called pGrid. The proposed pGrid improves the efficiency of the offline component of the D-Stream algorithm. The work also presents results of sensitivity studies that shed light on the relationship between different parameters and the number of generated clusters. The clustering process is designed to take advantage of the MapReduce framework and pGrid is implemented using Hadoop, a Java implementation of the MapReduce concept [5].

### 2.3 THE MAPREDUCE FRAMEWORK

MapReduce is designed to support parallel computations over large data sets on clusters of computers. The key components of the MapReduce framework include the input reader, the Map function, the partition function, the compare function, the Reduce function, and the output writer.

Data are fed into the Map function as <key, value> pairs. Map function produces one or more intermediate values along with new output keys. In the Reduce function, all the intermediate values of a given output key are combined together into a list. The lists are then combined into one or more final values for the output key.

Here is an example of MapReduce. Assume that we have a set of documents, and we want to count the term frequency of each word in the document set. This application can be implemented by using the MapReduce framework.

Suppose we have two documents: doc1 “Hello world” and doc2 “Hello ORNL”. The keys are “doc1” and “doc2”, while the values are “Hello world” and “Hello ORNL”, respectively. In the Map function, we need to convert the input pairs <k1, v1> to output pairs List(<k2, v2>), where k2 and v2 are new key and value. In our case, we use the words in the documents, namely “Hello, world, ORNL” as k2. The value of the output is the term frequency of each word. As illustrated by Fig. 3, the output of the Map task is
“<Hello, (1,1)>”, “<world, 1>” and “<ORNL, 1>”.

The second function is Reduce. In the example, the new key (k3) after Reduce remains to be the same key as k2, which is a word in the documents. However, the value is changed to the summation of frequencies of a word in all documents. After Reduce, the final output is “<Hello, 2>”, “<world, 1>” and “<ORNL, 1>”.

A generic architectural view of the MapReduce framework is shown in Fig. 4.

Hadoop is a Java implementation of the MapReduce framework [5]. It divides the application into many small fragments of tasks. Each task may be executed or re-executed on any node of a computer cluster. Hadoop also provides a fault-tolerant distributed file system and it automatically handles node failures. We implement the pGrid algorithm using Hadoop. Details of pGrid are discussed in the following section.
3. THE PGRID ALGORITHM

In this section, we first introduce the basic definitions and concepts of the pGrid algorithm. Then we describe the major workflow and key components, followed by two examples. Finally, we analyze the computational complexity of the algorithm.

3.1 BASIC DEFINITIONS

pGrid is a parallel, grid-based, two-phase clustering algorithm for data streams. Like D-Stream, it partitions the multi-dimensional data space into grids and clusters the grids instead of raw data points. In our discussion, we follow all the notations and definitions of D-Stream as described in Section 2.2; we will introduce new definitions related to pGrid as the discussion goes along.

Definition 3.1.1 (gridID) The unique identifier of a grid, e.g., g₀, g₁, …

Definition 3.1.2 (gridIndex) For a grid, the gridIndex is a d-dimensional vector <v₁, v₂, …, v_{d-1}, v₃>, where vₖ is the index of the grid on the kth dimension.

We use the (gridID, gridIndex) pair to uniquely identify a grid g.

3.2 THE PGRID WORKFLOW

In this section, we will discuss the general workflow of the pGrid algorithm (Fig. 5).

We partition the d-dimensional data space into grids, where each dimension contains len partitions. The workflow consists of an online component and an offline component. Like D-Stream, the online component is invoked whenever a new data point arrives. The incoming data point is projected onto its corresponding grid according to its attributes, and then the grid density is updated.

In contrast, the offline component is triggered periodically after a certain time interval. Sporadic grids are removed. Only dense and transitional grids are considered for clustering at each time interval. The offline component first clusters grids on each dimension in parallel into local clusters, and then combines
the local clusters on different dimensions to obtain the final clustering result (global clusters). The offline
component is designed and implemented by using the MapReduce framework. Parallel execution of the
clustering process significantly reduces the computation time and scales well to high dimensional data.
This is the major distinguishing factor from D-Stream, in which the clusters are generated by sequentially
inspecting grids along all dimensions. We will discuss both the online and offline component in detail in
the following sub-sections.

3.2.1 The Online Component

With the data space partitioned, when a new data point \( x_i <x_{i,1}, x_{i,2}, ..., x_{i,k}, ..., x_{i,d} > \) comes, the online
component projects it into its corresponding grid with \( \text{gridIndex} <v_{i,1}, ..., v_{i,k}, ..., v_{i,d-1}, v_{i,d} > \) using the
following formula:

\[
v_{i,k} = \begin{cases} 
\frac{x_{i,k} \times \text{len}}{\theta_k} & x_{i,k} \leq \theta_k \\
\text{len} - 1 & x_{i,k} > \theta_k 
\end{cases}
\]

3.2.2 The Offline Component

This section defines concepts related to parallel clustering, describes the clustering rules, and provides
a 2-D example that illustrates the clustering process.

3.2.2.1 Cluster definitions

Definition 3.2.2.1.1 (Possible Neighboring Grids) If two grids \( g_0: (g_0, <v_{0,1}, ..., v_{0,k}, ..., v_{0,d-1}, v_{0,d}>) \) and
\( g_1: (g_1, <v_{1,1}, ..., v_{1,k}, ..., v_{1,d-1}, v_{1,d}>) \) have the same indices in \( d-1 \) dimensions except the \( k^{th} \) dimension, they
are possible neighboring grids on the \( k^{th} \) dimension. Particularly, if \( g_0: (g_0, <v_{0,1}, ..., v_{0,k}, ... v_{0,d-1}, v_{0,d}>, ..., g_n: (g_n, <v_{n,1}, ..., v_{n,k}, ..., v_{n,d-1}, v_{n,d}>) \) are a group of possible neighboring grids on the \( k^{th} \) dimension,
which satisfy \( v_{0,m} = v_{1,m} = ... = v_{n,m} = v_m \) \((m=1, ..., d, m \neq k)\), then we call \( g_0, ..., g_n \) a group of possible
neighboring grids on the \( k^{th} \) dimension with \( \text{IndexKey} <v_1, v_2, ..., v_{k-1}, *, v_{k+1}, ..., v_d> \). The value of the \( k^{th} \) dimension, \( v_k \), is called \( \text{ValueInDim} \) of a grid.

Definition 3.2.2.1.2 (Neighboring Grid) If two grids \( g_0: (g_0, <v_{0,1}, ..., v_{0,k}, ... v_{0,d-1}, v_{0,d}>) \) and \( g_1: (g_1, <v_{1,1}, ..., v_{1,k}, ..., v_{1,d-1}, v_{1,d}>) \) are adjacent in the \( k^{th} \) dimension \((k \leq d)\), and have the same indices in all other
\( d-1 \) dimensions, i.e., \( v_{0,m} = v_{1,m} = ... = v_{n,m} = v_m \) \((m=1, ..., d, m \neq k)\) and \( v_{0,k} = v_{1,k} \pm 1 \), then \( g_0 \) and \( g_1 \) are neighboring grids on the \( k^{th} \) dimension. Two grids can be neighboring grids on the \( k^{th} \) dimension only if they are possible
neighboring grids on the \( k^{th} \) dimension, and their \( \text{ValueInDim} \) are next to each other.

Definition 3.2.2.1.3 (Local Cluster) Cluster generated by combining possible neighboring grids with the same \( \text{IndexKey}. \) \textbf{Clustering Rule 1} in section 3.2.2.2 provides the details.

Definition 3.2.2.1.4 (Global Cluster) Cluster generated by merging local clusters according to
\textbf{Clustering Rule 2} described in section 3.2.2.2. Global Clusters are the final clustering results of the
pGrid algorithm.
3.2.2.2 Clustering rules

Clustering Rule 1 governs the creation of local clusters and Clustering Rules 2 defines how global clusters are generated.

**Clustering Rule 1**: When we cluster grids that have the same IndexKey on the \( k \)th dimension, dense grids that are neighboring grids form a local cluster on the \( k \)th dimension. If a transitional grid has one neighboring grid that is dense, the transitional grid belongs to the local cluster to which the dense grid belongs. If a transitional grid has two neighboring grids that are both dense and belong to two different local clusters, the transitional grid belongs to the larger local cluster.

**Clustering Rule 2**: If two local clusters contain the same grid, they form a global cluster. If two global clusters contain the same local cluster or global cluster, these two global clusters form a larger global cluster. For instance, if \( \{g_0, g_1\} \subseteq \text{LocalCluster}_0 \) and \( \{g_1, g_2\} \subseteq \text{LocalCluster}_1 \), then \( \{g_0, g_1, g_2\} \subseteq \text{GlobalCluster}_0 \).

3.2.2.3 A 2-D MapReduce clustering example

Now we will walk through a clustering example and apply the pGrid algorithm on 2-dimensional data. The data space is partitioned into 3*3 grids. In other words, the dimension \( d=2 \) and the number of partitions on each dimension \( len=3 \). We have 9 grids in the data space, namely \( g_0, ..., g_9 \). The (gridID, gridIndex) pairs of these 9 grids are shown on the upper-left table in Fig. 6. Assume that among the 9 grids, only \( g_1, g_2, g_5, g_6 \) are dense grids, while the other grids are sparse grids. For simplicity, we only include dense grids in this example. Note that only dense grids and transitional grids are clustered.

There are two levels of parallelism in this process: on the level of each dimension and on the level of IndexKey within a dimension.

**First level parallelism**: The offline component clusters the dense and transitional grids along the X and Y dimensions in parallel. Since the clustering processes on both dimensions are the same, we only describe how local clusters are generated along the Y dimension.

On the Y dimension, we have three different IndexKeys \(<0,*>, <1,*>, <2,*>\). Grid \( g_1 (1,0) \)'s IndexKey is \(<1,*>\) and its ValueInDim is 0. Each IndexKey uniquely identifies a group of possible neighboring grids on that dimension. We mark groups of possible neighboring grids by red circles in Fig. 6.

**Second level parallelism**: Now we start to form local clusters by using Clustering Rule 1. We examine all the possible neighboring grids with the same IndexKey in parallel. For instance, when we process IndexKey \(<0,*>\), we scan grids \( g_0, g_3 \) and \( g_6 \). In the example, \( g_6 \) is a dense grid, while \( g_0 \) and \( g_3 \) are sparse grids. Therefore, \( g_6 \) itself forms a local cluster \( a_1 \). Similarly, we generate local clusters \( a_2 \) and \( a_3 \) on the Y dimension.

The next step is to combine these local clusters into global clusters. According to Clustering Rule 2, \( a_1 \) and \( b_1 \) should be combined to form a global cluster \( glb1 \), since they both contain grid \( g_6 \). Local clusters \( a_2 \) and \( b_1 \) share \( g_1 \), \( a_3 \) and \( b_1 \) share \( g_2 \), and \( a_3 \) and \( b_2 \) share \( g_2 \), and therefore, local clusters \( a_2, b_2, \) and \( b_3 \) are combined into global cluster \( glb2 \).
3.2.2.4 Implementing pGrid using MapReduce

The implementation of the pGrid algorithm involves three Map/Reduce processes (Fig. 7). Each Map/Reduce process changes the <key, value> input. In this section, we describe each of the Map/Reduce step by using the same 2-D example as the one we have shown in section 3.2.2.3.

First Map/Reduce Process

\[ \langle \text{gridID, gridIndex} \rangle \xrightarrow{\text{map}} \langle \text{IndexKey, ValueInDim} \rangle \xrightarrow{\text{reduce}} \langle \text{IndexKey, ValueInDim_list} \rangle \]

Second Map/Reduce Process

\[ \langle \text{IndexKey, ValueInDim list} \rangle \xrightarrow{\text{map}} \langle \text{gridID, localCHID} \rangle \xrightarrow{\text{reduce}} \langle \text{gridID, localCHID_list} \rangle \]

Third Map/Reduce Process

\[ \langle \text{gridID, localCHID list} \rangle \xrightarrow{\text{map}} \langle \text{gridID, globalCHID} \rangle \xrightarrow{\text{reduce}} \langle \text{globalCHID, gridID_list} \rangle \]

Fig. 6. A 2-D example of the MapReduce clustering.

Fig. 7. The MapReduce implementation.

The output of each MapReduce process is the input of the next one. The initial input is the \(<\text{gridID, gridIndex}>\) pairs of all the dense and transitional grids. The output of the first MapReduce process is a list of IndexKeys (i.e., \(<v_1, v_2, ..., v_k, *, v_{k+1}, ..., v_d>\)) and all the possible values associated \(v_k\) which is represented by the “*” in a IndexKey. The second MapReduce clusters the grids into local clusters.
according to Clustering Rule 1. The third MapReduce combines the local clusters into global clusters according to Clustering Rule 2.

We now use the same 2-D example to illustrate the three MapReduce processes. Fig. 8 shows the first MapReduce process. We obtain IndexKey and ValueInDim from a grid’s gridID. For instance, grid \( g_1 \) (1,0)’s IndexKey is \(<1,*>\) and its ValueInDim is 0. The outputs of the first MapReduce are lists of ValueInDims grouped by IndexKeys.

The second MapReduce generates local clusters on each dimension in parallel and on each IndexKey in parallel (shown in Fig. 9). Take the IndexKey \(<2,*>\) for example. On the Y dimension, there are two dense grids which have \(<2,*>\) as their IndexKey: \( g_1 \) and \( g_2 \) (The grid coordinates can be found in Fig. 6). Their ValueInDim are 0 and 1 respectively. Since \( g_1 \) and \( g_2 \) are dense grids with adjacent ValueInDims, they are clustered into the same local cluster \( a_3 \). The output is a list of gridIDs and the list of local clusters to which it belongs. Note that each grid may belong to more than one local cluster depending on which dimension we are examining.
The third MapReduce is illustrated in Fig. 10. The Map function combines local clusters into global clusters, and Reduce sorts the result with globalCltID as the key.

3.2.2.5 A 4-D MapReduce clustering example

When we move to high-dimensional data spaces, the IndexKey becomes more complicated. In this section, we use a 4-D example to demonstrate how the pGrid algorithm applies to high-dimensional data. Fig. 11 is an example of 4-dimensional data. There are 6 dense grids: g0, g6, g7, g8, and g3 have the same IndexKey of <*, 0, 0>, while g4 and g5 have the same IndexKey of <*, 0, 1>. Therefore, we have two groups of possible neighboring grids. We then generate local clusters in parallel on these two groups with different IndexKey.
Fig. 11. A 4-D pGrid clustering example.

In the group of possible neighboring grids represented by IndexKey <*, 0, 0, 0>, g₀ and g₁ are adjacent on the first dimension, so they form one local cluster. g₂ and g₃ form two separate local clusters because they are not adjacent to any other dense grid. In the group of possible neighboring grids represented by IndexKey <*, 0, 0, 1>, g₄ and g₅ are adjacent on the first dimension, they form another local cluster. The same process repeats on the 2nd, 3rd and 4th dimensions. Then we combine all the local clusters into global clusters following Clustering Rule 2.

3.3 COMPUTATIONAL COMPLEXITY ANALYSIS

We now compare the computational complexity of the D-Stream and pGrid algorithms.

Each time the offline component runs, D-Stream iteratively adjusts the clusters by merging or dividing them. In contrast, pGrid is not an iterative method. Because the number of iterations that the D-Stream algorithm executes varies depending on the data set, we only consider one iteration in the following analysis.

At each iteration of the D-Stream algorithm, in the worst-case scenario, there are $l_{en}^d$ grids in memory. For each of these grids, one needs to examine its neighboring grids in all $d$ dimensions. Therefore, the computational complexity is $O(l_{en}^d \times d)$. But in practice, the number of grids in memory is much smaller, under the assumption that the data space is sparse.

In the pGrid algorithm, Map2 and Map3 functions are on the critical path of the computation. In Map2, we need to scan all the grids with the a certain IndexKey, which costs $O(len)$. Since different IndexKeys are processed in parallel, the total time needed for Map2 remains to be $O(len)$, regardless of the number of IndexKeys. In Map3, each grid has at most $d$ local cluster (localClt) assignments, so we only need to scan these $d$ local clusters (localCltts) and append the corresponding mapping in the global mapping list. As a result, the computation time is $O(len + d)$. Note that we did not take into consideration the overhead introduced by the MapReduce task initialization process.
4. PERFORMANCE EVALUATION

In this section, we present the data set, experiment setup, performance metrics, and the analysis of experimental results.

4.1 DATASET

The data set we used in our experiments is the KDD Cup 99’ network intrusion data from the MIT Lincoln Lab [23]. The raw training data was about four gigabytes of compressed binary tcpdump data from seven weeks of network traffic. This was processed into about five million connection records. Similarly, the two weeks of test data yielded around two million connection records. In our experiments, we used a subset of 12,000 data points, where each data point represents a connection. Each data point contains 34 attributes. Fig. 12, 13, and 14 summarize the main information. More details about these attributes can be found in [1].

There are 5 main categories of connection types, which correspond to the normal connection and four types of attacks: DOS, R2L, U2R, and probing. Each attack type further contains several subcategories. The whole dataset contains 22 sub-categories. The subset we use contains 8 of the subcategories.

We first normalize all attributes of the data set to [0,1]. We screen a sample set of data points X of size n (n=12,000 in our experiments), \( X = \{ x_i: <x_{i,1}, ..., x_{i,k}, ..., x_{i,d-1}, x_{i,d}> \}, i = 1, ... , n \). The data points we used have non-negative values on all dimensions. Then we calculate the normalization factor of the \( k^{th} \) dimension as

\[
\theta_k = \frac{1}{\sqrt{n}} \sum_{i=1}^{n} x_{i,k}
\]

We normalize the \( k^{th} \) dimension to [0,1] by dividing \( x_{i,k} \) by \( \theta_k \). If the outcome value is larger than 1, we set it to 1. This is to reduce the impact of the outliers which are too large relative to the average value. Then we partition that dimension into \( len \) partitions. As a result, we obtain \( len^d \) grids in the data space.

<table>
<thead>
<tr>
<th>feature name</th>
<th>description</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>duration</td>
<td>length (number of seconds) of the connection</td>
<td>continuous</td>
</tr>
<tr>
<td>protocol_type</td>
<td>type of the protocol, e.g., tcp, udp, etc.</td>
<td>discrete</td>
</tr>
<tr>
<td>service</td>
<td>network service on the destination, e.g., http, telnet, etc.</td>
<td>discrete</td>
</tr>
<tr>
<td>src_bytes</td>
<td>number of data bytes from source to destination</td>
<td>continuous</td>
</tr>
<tr>
<td>dst_bytes</td>
<td>number of data bytes from destination to source</td>
<td>continuous</td>
</tr>
<tr>
<td>flag</td>
<td>normal or error status of the connection</td>
<td>discrete</td>
</tr>
<tr>
<td>land</td>
<td>1 if connection is from/to the same host/port; 0 otherwise</td>
<td>discrete</td>
</tr>
<tr>
<td>wrong_fragment</td>
<td>number of &quot;wrong&quot; fragments</td>
<td>continuous</td>
</tr>
<tr>
<td>urgent</td>
<td>number of urgent packets</td>
<td>continuous</td>
</tr>
</tbody>
</table>

Fig. 12. Basic features of individual TCP connections.
4.2 EXPERIMENT SETUP

We run all the experiments on a PC with a 2.79GHz CPU and 3GB memory. The pGrid algorithm is implemented with Hadoop 0.16.4 and JDK5.0 under Cygwin. For comparison, we implemented D-Stream in Eclipse with JDK5.0.

Unless otherwise noted, the parameters are set to be the same as in [1]: len=25, dm=3.0, dl=0.8, beta=0.3, λ=0.998. The data stream speed is set to be 200 data points per time unit. Time unit is the measure of time in our data stream simulation.

4.3 PERFORMANCE METRICS

SSQ and cluster purity are the two performance metrics used extensively in data stream clustering.
We also use the number of clusters to measure the difference between the clustering results and the ground truth.

SSQ is the sum of square distance, which is defined as follows. If current time is $t_c$, for a certain time horizon $t$ starting from time $t_c - t$, there are $N$ data points coming in. Among them, $N'$ data points are assigned to some clusters, while the other $N - N'$ data points are outliers. For each data point $p_i$, we find the centroid $C_{p_i}$ of its closest cluster, and compute the distance between $p_i$ and $C_{p_i}$, $d(p_i, C_{p_i})$. Then the SSQ at time $t_c$ is defined as the summation of $d^2(p_i, C_{p_i})$ for all these $N'$ data points.

Cluster Purity is defined as the average percentage of the dominant class label in each cluster. The higher percentage of the dominant class label in each cluster, the higher cluster purity is.

The number of clusters is defined as the number of clusters in the clustering result at a certain time $t_c$.

4.4 EXPERIMENTAL RESULTS

4.4.1 Overall Performance

In this section, we compare the pGrid and D-Stream algorithms by using the three performance metrics. The Time axis shows, when the performance metrics were calculated, how many time units have elapsed since the beginning of the data stream. Since the data set contains 12,000 data points and the stream speed is 200, the data stream lasts for 60 time units. We randomly choose 16, 30, 44, 58 as the times at which we measure the clustering quality, so that there are 14 time units in each time interval.

Fig. 15 shows the average SSQ of both the pGrid and D-Stream algorithms at different times. The results show that pGrid and D-Stream have similar performance on average SSQ. Since average SSQ
is measured in logarithm, the difference is smaller than one order of magnitude, which is not significant.

We notice that pGrid has different average SSQ with D-Stream, though it can be viewed as a parallel version of D-Stream. The reason is that the two algorithms treat transitional grids with slight difference. In D-Stream, if two clusters are connected only through a transitional grid, they will be separated into two smaller clusters. In pGrid, however, since we cluster on each dimension in parallel and combine the clustering results, we are unable to detect if some clusters are connected with each other only through a transitional grid. We argue that the density of a transitional grid is still comparatively high and does not affect the cluster result much; therefore, the result produced by pGrid is reasonable. This also explains the difference of cluster purity shown below.

Fig. 16 shows the cluster purity of both algorithms. While both algorithms achieved high cluster purity (> 90%), pGrid’s cluster purity is slightly lower than D-Stream. However, the maximum difference between the two is less than 5%. Note that the total number of clusters generated at each time unit may have significant impact on the cluster purity and average SSQ. Naturally, larger numbers of clusters will result in smaller average SSQ and higher cluster purity. Therefore, we conducted a set of experiments to investigate the number of clusters generated by each algorithm in comparison to the ground truth.

Fig. 17. Number of clusters generated by pGrid and D-Stream compared with the ground truth.

Fig. 17 plots the number of clusters generated by both algorithms and the ground truth. The ground truth is the actual number of clusters in the dataset, generated by manually labeling. Since the data stream evolves over time, at time 16 and 30, there are only 5 clusters in the data stream, but after time 44, the total number of clusters has increased to 8. As we can see that pGrid generates less number of total clusters than D-Stream in all cases and is closer to the ground truth. This explains why pGrid performed slightly worse than D-Stream in cluster purity.

Fig. 18. Average SSQ of pGrid with different len values.
4.4.2 Sensitivity

The computational complexity analysis in section 3.3 shows that the value of \( len \) is one of the determining factors of the computational cost for both algorithms, especially for the D-Stream algorithm. The larger the \( len \) value, the higher the computational cost. In this section, we study the impact of the parameter \( len \) on the performance of pGrid.

![Fig. 19. Cluster purity of pGrid with different \( len \) values.](image1)

![Fig. 20. Number of clusters generated by pGrid with different \( len \) values.](image2)

Experimental results indicate that average SSQ, cluster purity and number of clusters all have a general trend to decrease as \( len \) increases. This can be explained as follows. Grids are used as approximations of the actual data points. This approximation may introduce errors in the clustering results, especially near the boundaries of clusters. A larger \( len \) value leads to finer grids, and therefore the grids are a closer representation to the actual data points. Thus, the total number of clusters generated by pGrid is closer to the ground truth. In fact, Fig. 20 shows that when \( len = 200 \), at time 44 and 58, the number of clusters generated by pGrid is the same as the ground truth. The smaller number of clusters in turn causes the cluster purity to decrease, because if a cluster is assigned to each data point, the purity is guaranteed to be 100%.

4.4.3 Parameter Influence on the Number of Clusters

The experimental results indicate that the number of clusters has a direct impact on the cluster purity and SSQ. Therefore, we now study the parameter settings and discover patterns to control the number of clusters generated by pGrid.

In the following experiments, we set \( speed=200, \ lamda=0.998 \). The results shown in the figures are the corresponding values at time \( t=60 \).
In Fig. 21, we set $dm=3.0$, $dl=0.8$, $beta=0.3$, and investigate how the value of $len$ influences the number of clusters. We can see that when $len=25$, the number of clusters is the greatest.

Figs. 21 and 22 show the relationship among the number of clusters, $dm$, and $dl$, where $dm$ and $dl$ are the thresholds of dense/sparse grid, respectively. The number of clusters decreases as $dm$ increases. $dl$ does not have a significant impact on the number of clusters generated by pGrid.
5. CONCLUSION AND FUTURE WORK

5.1 CONCLUSION

In this report, we present a novel parallel grid-based clustering algorithm for high-dimensional data streams called pGrid. It uses the MapReduce framework to process each dimension in parallel. Experimental results show that pGrid performs comparably to the D-Stream algorithm in terms of average SSQ and cluster purity. In addition, pGrid explores the parallelism in the clustering process, and therefore it is much more efficient and scalable than the D-Stream approach, which is a sequential method. We expect applications that face the challenge of high volume, high rate data streams to significantly benefit from our algorithms. Some examples include NIDS and real-time news feed analysis.

We presented a theoretical analysis of the computation complexity for pGrid. As the next step, we plan to conduct the speed and scalability tests on computer clusters. We are also interested in exploring the potential of pGrid on a much more distributed environment such as the cloud computing.

5.2 FUTURE WORK

In the future, we plan to extend the pGrid algorithm from the following directions:

5.2.1 Efficiency Test

We would like to test the pGrid algorithm on a computer cluster with MapReduce, gather experimental results of the efficiency, and compare it with D-Stream. Due to time limitations, we did not run an efficiency test on a large computer cluster. Although the pGrid algorithm has been proved to be more efficient, the overhead of running Hadoop cannot be ignored. For each clustering process, we need to initialize 3 MapReduce jobs, which may take significant amount of time. Therefore, an efficiency test is necessary in the future.

5.2.2 An Alternative Performance Metric

Most of the existing data stream clustering research uses SSQ and cluster purity as the performance metrics. However, neither of these metrics takes the number of cluster generated by the algorithm into account. If an algorithm generates a large number of clusters, each of the clusters will appear to be “more pure”, and the SSQ will be smaller. This becomes obvious when we think about an extreme example in which each grid itself becomes a cluster.

As a result, we propose to incorporate the number of clusters as one of the metrics of clustering quality. A possible way to do this is to add a post-processing component in the clustering algorithm, which combines or splits the clusters based on similarities to get exactly the same number of clusters as the ground truth. We can then apply the same performance metrics, SSQ and cluster purity. By doing so, we avoid favoring algorithms that generate large numbers of clusters and, therefore, make the comparisons fairer.
6. REFERENCES


ACKNOWLEDGMENTS

Notice: This manuscript has been authored by UT-Battelle, LLC, under contract DE-AC05-00OR22725 with the US Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes.