Supporting Correlation Analysis on Scientific Datasets in Parallel and Distributed Settings

Yu Su
Computer Science and Engineering
The Ohio State University
Columbus, OH 43210
su1@cse.ohio-state.edu

Gagan Agrawal
Computer Science and Engineering
The Ohio State University
Columbus, OH 43210
agrawal@cse.ohio-state.edu

Jonathan Woodring
Los Alamos National Laboratory
Los Alamos, NM 87544
woodring@lanl.gov

Ayan Biswas
Computer Science and Engineering
The Ohio State University
Columbus, OH 43210
biswas.36@osu.edu

Han-Wei Shen
Computer Science and Engineering
The Ohio State University
Columbus, OH 43210
hwshen@cse.ohio-state.edu

ABSTRACT

With growing computational capabilities of parallel machines, scientific simulations are being performed at finer spatial and temporal scales, leading to a data explosion. Careful analysis of this data holds much promise for future scientific discoveries. Particularly, *correlation analysis*, which focuses on studying the potential relationships among multiple variables, is becoming a useful method for scientific analysis. This paper focuses on the problem of correlation analysis across large-scale simulation datasets, including 1) accelerating this analysis with the use of bitmap indexing as a representative summary of the data, 2) developing efficient algorithms for parallel execution, 3) performing analysis in distributed environments, i.e., for cases where different attributes are stored in geographically distributed repositories, and 4) combining sampling with correlation analysis. These algorithms have been implemented in a system that provides a high-level API for specification of the analyses, including allowing correlation analysis on specified value-based and dimension-based subsets of the data, and supports interactive and incremental analysis. We have extensively evaluated our framework for efficiency, and have also carried out case studies with domain scientists to establish how it can aid data-driven discovery process.

Categories and Subject Descriptors

I.3.1 [**Computing Methodologies**]: HARDWARE ARCHITECTURE—*Parallel Processing*; H.3.1 [**Information Systems**]: CONTENT ANALYSIS AND INDEXING—*Indexing Methods*; H.1.1 [**Information Systems**]: SYSTEMS AND INFORMATION THEORY—*Information Theory*

Keywords

Big Data; Correlation Analysis; Indexing; Parallel Processing

HPDC'14, June 23–27, Vancouver, BC, Canada.

1. INTRODUCTION

As science has become increasingly data-driven, and as data volumes and velocities are increasing, scientific advances in many areas will only be feasible if critical "big-data" problems are addressed. One of the key challenges being faced by data-intensive science efforts is that while the dataset sizes continue to grow rapidly, disk speeds, and both inter-cluster and wide-area bandwidths are not coping up. Thus, data movement is increasingly becoming the bottleneck.

There is often a need for complex analyses over datasets generated by scientific simulations. Such analyses can be classified into two categories: *individual variable analysis* and *correlation analysis*. Individual variable analysis involves analysis over each variable or attribute independently, and can take the form of data subsetting, data aggregation, data mining or visualization. Much of the existing work, especially in data visualization, has focused on individual variable analysis. However, more recently, several efforts [4, 26] have focused on studying the relationship among multiple variables and making interesting scientific discoveries based on such analysis.

This paper focuses on the problem of correlation analysis on massive scientific datasets in parallel and distributed settings. The "big data" problem of data movements being the constraint becomes even more severe for correlation analysis, because of several reasons. Even if different data files are stored on the same server, correlation calculation has a large memory cost and the entire calculation process is extremely time-consuming. Parallelizing such analysis is also hard, because of the possibility of large-scale data movement. Moreover, besides addressing the algorithmic challenges, there is a need for a system that can offer a high-level interface for such analysis. For example, in many cases, scientists may only be interested in performing correlation analysis over (*value-based* and/or *dimension-based*) subsets of the data, and a structured query interface is needed for such analysis. Because

scientific datasets are stored in formats like NetCDF or HDF5, and not in a database, support for such subsetting (especially, value-based subsetting) is not (efficiently) available.

Yet another problem arises from the fact that different variables may be stored in different servers. For example, consider Earth System Grid Federation (ESGF). Each *data node* can choose to download a subset of data available in the entire grid, and thus, it is possible that different attributes may be stored at different sites. Because of the volume of data, it is not feasible to simply download the data to be analyzed at one site, and a more intelligent approach is needed.

In our work, we propose a set of algorithms and a system for correlation analysis in parallel and distributed settings, starting from a high-level API, and with support for incremental and interactive analysis. From an algorithmic side, we present a series of methods for correlation analysis using bitmap indexing [8, 30, 31]. Bitmap indices, one of the popular indexing methods, preserve both the value distribution and the spatial locality of the data, and thus can be treated as a summary or profile of the original dataset, though much smaller in size. Moreover, because bitmap indices can help support basic database-like operations, e.g., data subsetting [8, 21], they do not have to be built exclusively for correlation analysis. Because of the data reduction associated with bitmap indices, the novel correlation calculation algorithms we have developed can incur much smaller network data transfer and memory accesses costs compared with the traditional method. We have designed two different indexing methods, which are *dynamic* and *static* indexing, to improve the efficiency. It turns out that bitmap indices or *bitvectors* also can help reduce the amount of communication during the computation of correlations in a parallel environment. We have developed two different partitioning methods, dimension-based partitioning and value-based partitioning, to perform parallel correlation analysis with bitvectors. These methods are also extended to a distributed setting, where datasets corresponding to different variables may be stored at geographically distributed locations.

Finally, with the help of bitmap indexing, we are able to generate *accurate* samples, which are then used to perform correlation analysis more efficiently. This allows us to trade some of the accuracy for improved response time. We have developed algorithms to use bitvectors for sampling and support correlation analysis based on these samples.

The algorithms we have developed have been incorporated in a flexible correlation analysis system, which also has several other desirable properties. First, correlation analysis is offered from a high-level API, where users can conveniently express *dimension-based* and *value-based* subsetting conditions (using an SQL-like syntax). These subsetting conditions are also supported on scientific datasets using bitmap indices. Moreover, we support *incremental* analysis. Users can add additional constraints on the top of the subset used for the last round of analysis. Finally, users can also perform an *undo* operation, which will allow them to build on top of not the most recent result, but an earlier result (and possibly further specialize on those subsets). With the help of bitmap indexing, we only need to keep track of bitvectors during the interactive querying process to support these features.

We have extensively evaluated our system. We compare the correlation analysis efficiency between the traditional method and our method in a stand-alone environment, and show that our dynamic indexing method can achieve a speedup from 1.78x to 3.61x and static indexing method can achieve a speedup from 11.4x to 15.35x. We show the scalability of the parallel method and compare the efficiency between two different partitioning methods. Next, we show that in an environment where computing resources and the data are geographically distributed, our method can further improve the efficiency compared with the traditional by 1.87x to 2.96x. We also show that if correlation analysis is performed over samples of the data (e.g., 25%), we can achieve a significant speedup (1.69x) with only a small accuracy loss (3.42%). Besides efficiency evaluation, we have also demonstrated the efficacy of the system, by case studies involving domain scientists at the Los Alamos National Laboratory, establishing how that our system is able to aid the data-driven discovery process.

2. MOTIVATING QUERIES AND PROPOSED SYSTEM INTERFACE

This section first introduces the popular correlation metrics, whose computations we are supporting. Next, we explain the query interface and the functionality we are intending to provide.

2.1 Correlation Metrics

This subsection introduces several correlation metrics from information theory [10, 24, 25, 5] that have lately been used in scientific data analysis.

2-D Histogram: In statistics, a histogram is a graphical representation of the distribution of the data, or in other words, an estimate of the probability distribution of a continuous variable. A 2-D histogram reflects the value distribution of one variable regarding to the value changes of another, and is a useful metric to indicate the value distribution relationship between variables.

Information Entropy: In information theory, the information content of a random variable can be quantified by Shannon's entropy [12]. Constant data (easily predictable) have low entropy, while apparently random data (uniform probability) have high entropy. Although entropy is normally applied to single variable, it can also be used to indicate correlations among variables (an example will be shown in Section 6).

Mutual Information: Mutual information is the metric for computing the dependence between two random variables, and shows the amount of shared information between two variables in the number of bits. If the mutual information is low, then the two variables are independent. Conversely if mutual information is high, one variable provides information about the other. Equation 1 shows the expression to calculate the mutual information. Here, we index the data in the variables or attributes A and B by j and k separately, and use x_j and y_k to represent each distinct value. N_A and N_B represent the number of distinct values of each attribute, and three probability distribution functions, P_A, P_B and P_{AB}, capture the probability of having each distinct value for A, for B, and for a pair of values of A and B, respectively.

$$I = \sum_{j=1}^{N_A} \sum_{k=1}^{N_B} P_{AB}(x_j, y_k) \times \log\left(\frac{P_{AB}(x_j, y_k)}{P_A(x_j) P_B(y_k)}\right) \quad (1)$$

2.2 Query Interface and Desired Functionality

We now show the functionality and the interface we intend to support, through an example shown in Figure 1. In the first step, the users are asked to input variable names for correlation analysis. In this example, users want to perform correlation analysis over ocean temperature (*TEMP*), salinity (*SALT*), and flow velocity (*UVEL*). In the following steps, users are able to input different queries that specify subsets of the data they want to perform correlation analysis on. Particularly, in the example shown, users want to first see the correlations among *TEMP*, *SALT* and *UVEL* when the ocean temperature is between 0 and 1 and the depth of the ocean is below 50 meters. After the query is submitted, the system is able to calculate different correlation metrics among the variables directly and return the results (histogram, entropy, mutual information) as the output. Users are able to see the correlations and may be interested in further analysis, For example, a user may find that the mutual information between *TEMP* and *SALT* is much higher than

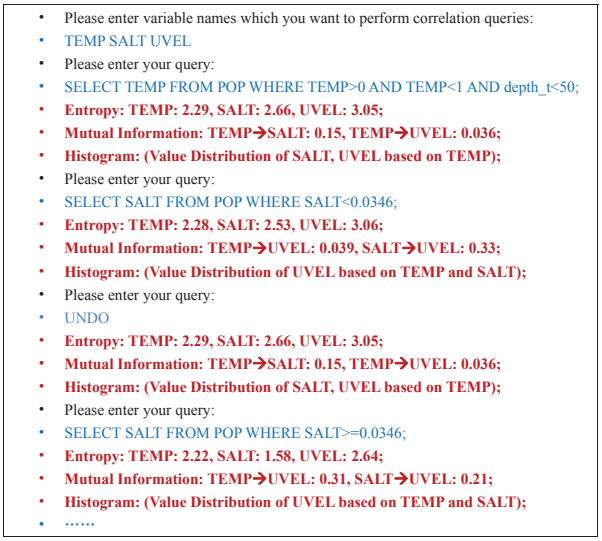

Figure 1: Example Showing Supported Query Interface

that between *TEMP* and *UVEL*, which means *TEMP* and *SALT* are high correlated for this data subset.

Then, in the next step, they may want to further explore a subset of values for *SALT*, i.e., further specialize on the previous query to generate more specific correlations. This is supported efficiently through *incremental analysis*, which we will describe later. If users are not satisfied with the current result, our approach supports an *undo* operation to go back to previous step. In this case, the system goes back to the second-last query. Thus, a new query can now specialize on top of the second-last query, and not the last query.

3. ALGORITHMS FOR CORRELATION ANALYSIS

This section describes correlation analysis algorithms we have developed. We start with a simple algorithm, explain its limitations, and then introduce bitmap indexing. We show a series of methods based on bitmap indexing, and then describe methods for execution in parallel and distributed settings.

3.1 Initial Method

A default algorithm that can be used for calculating any of the correlation metrics we listed comprises four steps. First, we need to load the entire data for the variables involved, say, variables A and B, into the memory. Next, if the correlation required is for certain subsets of the data, we take a pass through the data for the variables A and B to generate the data subsets based on queries. As a next step, we generate *joint bins* for A and B, i.e., first divide data for A and B into bins based on values, and thus generate (A_1, A_2, …, A_m) and (B_1, B_2, …, B_n), and then generate joint bins based on individual bins and the dataset, which are $(A_1, B_1) \rightarrow count_0$, $(A_1, B_2) \rightarrow count_1$, …, $(A_m, B_n) \rightarrow count_{mn}$, where $count_i$ is the number of elements located within the joint bin i. Finally, these counts are used to calculate the correlation metrics.

As one can see, the algorithm has a very high memory requirement. If the data corresponding to the two attributes cannot fit in memory, we need to orchestrate complex data movements, which can be very expensive.

3.2 Bitmap Indexing

To speed up the above method, we consider indexing methods that have been proposed in the literature. Broadly, indexing provides an efficient way to support value-based queries and has been extensively researched and used in the context of relational databases. Bitmap indexing, which utilizes the fast bitwise operations supported by the computer hardware, has been shown to be an efficient approach, and has been widely used in scientific data management [20, 30]. In particular, recent work has shown that bitmap indexing can help support efficient querying of scientific datasets stored in native formats [8, 21].

ID	Value	e_0	e_1	e_2	e_3	i_0	i_1
		=1	=2	=3	=4	[1, 2]	[3, 4]
0	4	0	0	0	1	0	1
1	1	1	0	0	0	1	0
2	2	0	1	0	0	1	0
3	2	0	1	0	0	1	0
4	3	0	0	1	0	0	1
5	4	0	0	0	1	0	1
6	3	0	0	1	0	0	1
7	1	1	0	0	0	1	0
Dataset		Low Level Indices				High Level Indices	

Figure 2: An Example of Bitmap Indexing

Figure 2 shows an example of a bitmap index. In this simple example, the dataset contains a total of 8 elements with 4 distinct values. The *low-level* bitmap indices contain 4 bitvectors, where each bitvector corresponds to one value. The number of bits within each bitvector is the same as total number of elements in the dataset. In each bitvector, a bit is set to 1 if the value for the corresponding data element's attribute is equal to the *bitvector value*, i.e., the particular distinct value for which this vector is created. The *high-level* indices can be generated based on either the value intervals or value ranges. From Figure 2, we can see two *high-level* indices are built based on value intervals.

This simple example only contains integer values. Bitmap indexing also has been shown to be an efficient method for floating-point values [32]. For such datasets, instead of building a bitvector for each distinct value, we can first group a set of values together (*binning*) and build bitvectors for these bins. This way, the total number of bitvectors is kept at a manageable level.

From the example we can also see that the number of bits within each level of bitmap indices is $n \times m$, where n is the total number of elements and m is the total number of bitvectors. This can result in sizes even greater than the size of the original dataset, causing high time and space overheads for index creation, storage, and query processing. To solve this problem, *run-length compression* algorithms such as Byte-aligned Bitmap Code (BBC) [2] and Word-Aligned Hybrid (WAH) [31, 13] have been developed to reduce the bitmap size. The main idea of these approaches is that for long sequences of 0s and 1s within each bitvector, an encoding is used to count the number of continuous 0s or 1s. Such encoded counts are stored, requiring less space. Another property of the run-length compression methods is that it supports fast bitwise operations without decompressing the data.

3.3 Advanced Algorithm Using (Dynamic) Bitmap Indexing

We now describe an efficient method for computing correlations, which we refer to as *dynamic* bitmap indexing based method. This method requires single-variable indices (one bitmap index over one variable).

Correlation calculation between the variables A and B using bitmap indexing is shown in Figure 3. This method assumes that bitmaps have been constructed for each of the variables or attributes involved. The algorithm involves three steps. First, we directly find

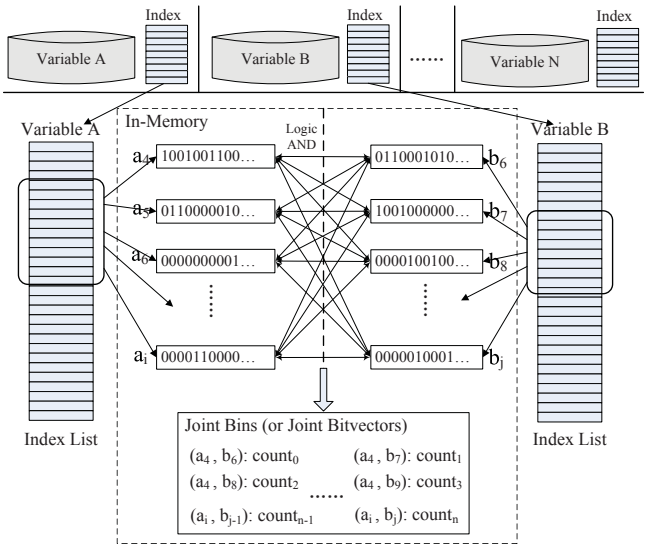

Figure 3: Dynamic Indexing based Correlation Computation

the subsets of bitvectors of the dataset for A (a_4 to a_i, in the example) and B (b_6 to b_j, in the example), which satisfy the current query and load them into the memory. Note that both *dimension-based* and *value-based* subsetting can be easily supported on bitmap indices, as they simply involve choosing certain rows and columns respectively from the bitvectors. In the second step, we generate *joint bins* for A and B. Because individual bins have been generated during the index generation phase and stored in the form of bitvectors, we only need to perform logic AND operations between bitvectors of A and B for this step. The total complexity of this step is $m \times n$, where m and n are the number of bitvectors (used in query) of A and B. In the final step, we calculate different correlation metrics based on the joint bins, just like the original method.

If we compare the indexing based method with the original method, we find that there are at least three advantages of using bitmap indices. First, our method only needs to load the indices (which are much smaller in size) instead of loading the dataset into the memory. Compared with the original method, we have much smaller memory requirements and the time to load data is also reduced. Second, data filtering, especially for a value-based filtering condition, is very costly with the original method, as one needs to examine each element. However, our method achieves this step by simply loading the bitvectors that satisfy the current value-based condition. Finally, a key step of calculating correlation metrics is to generate joint bins. Without indexing support, the joint bins have to be generated by scanning through each element in the data subset. However, with the help of bitmap indexing, the joint bins is based on logic AND operations, and is much simpler.

One issue, however, is the cost of creating bitmap index on the data, which can be expensive. However, in many cases, an index may be created for a variety of reasons, like supporting data subsetting [21] or sampling [22].

3.4 Using Static Bitmap Indexing

A further optimization of the above method is based on the following motivation. Within most of the scientific datasets, some variables are highly correlated while others are not. Meanwhile scientists also have preferences on correlation analysis over specific variable set. Static indexing, which builds multi-variable indices (i.e., one bitmap index over multiple variables) involves higher upfront cost and storage, but can be used to answer queries for specific combination of variables efficiently.

ID	A Value	B Value	jb_0 A [1,3] B [2,4]	jb_1 A [1,3] B [5,7]	jb_2 A [1,3] B [8,10]	jb_3 A [4,6] B [2,4]	jb_4 A [4,6] B [5,7]	jb_5 A [4,6] B [8,10]
0	4	3	0	0	0	1	0	0
1	1	2	1	0	0	0	0	0
2	4	5	0	0	0	0	1	0
3	1	9	0	0	1	0	0	0
4	2	3	1	0	0	0	0	0
5	6	10	0	0	0	0	0	1
6	5	3	0	0	0	1	0	0
7	4	7	0	0	0	0	1	0
8	1	9	0	0	1	0	0	0
9	2	5	0	1	0	0	0	0
Dataset			Multi-Variables Bitmap Indices					

Figure 4: Static Indexing

Specifically, we note that the (dynamic) indexing based method described above still requires bitwise operations between the datasets of variables A and B to generate the joint bins. The purpose of static indexing is to further reduce the cost of bitwise operations by generating a more involved index, over multiple variables, as shown in Figure 4. During the bitmap index generation phase, instead of generating two separate bins (A and B), we perform binning based on value subranges of both A and B (i.e., $A \cdot B$), and generate joint bitvectors based on the joint bins. In this example, the total number of joint bitvectors generated is 6. During the query process, we directly load the subset of joint bitvectors that satisfy the current query conditions into memory and generate joint bins by simply performing 1-bits counting operations over each bitvector. Based on that, the probability distribution of A, and B and $A \cdot B$ are generated and different correlation metrics are calculated. Moreover, static index can also be directly used to further calculate correlations between the current variable set and other variable or variable set, which is more efficient than dynamic indexing method

If we compare static indexing with dynamic indexing, static indexing method has larger index generation and storage costs, but the advantage is that for each query, we can directly find the subsets of joint bitvectors and calculate correlation metrics efficiently based on that. In comparison, with dynamic indexing, there is a need to perform bitwise operations between the bitvectors for different variables for each query. Hence, static indexing is more suitable for the cases where we know that certain variables are highly correlated, and/or it is known that the users will like to perform frequent correlation analysis on these variables.

3.5 Hierarchical Bitmap Indexing

Combining the benefits of both dynamic and static indexing, we have developed a hierarchical bitmap indexing framework to answer correlation analysis for scientific datasets. An example of this process is shown in Figure 5, where 11 variables are involved.

The entire process contains four steps. First, we build one bitmap index for each variable. These indices can be used for both individual data subsetting, as well as correlation analysis using dynamic indexing. Second, a simple clustering algorithm is applied over all variables to find and group those highly correlated variable pairs. In this step, an initial or approxiate correlation values between each variable pair is calculated and each pair whose correlation results is larger than a certain threshold is clustered into one group. From the figures, we can see that Var_1 and Var_2 are clustered into one group, and Var_5 and Var_6 are clustered into another. In this step, users are also able to manually build up groups if they want to perform frequent correlation analysis between certain variables, e.g., Var_9 and Var_{10}. After clustering, in the third step, static indices, as described earlier, are built over those clusters that contain two variables. These indices can be used to process frequent queries

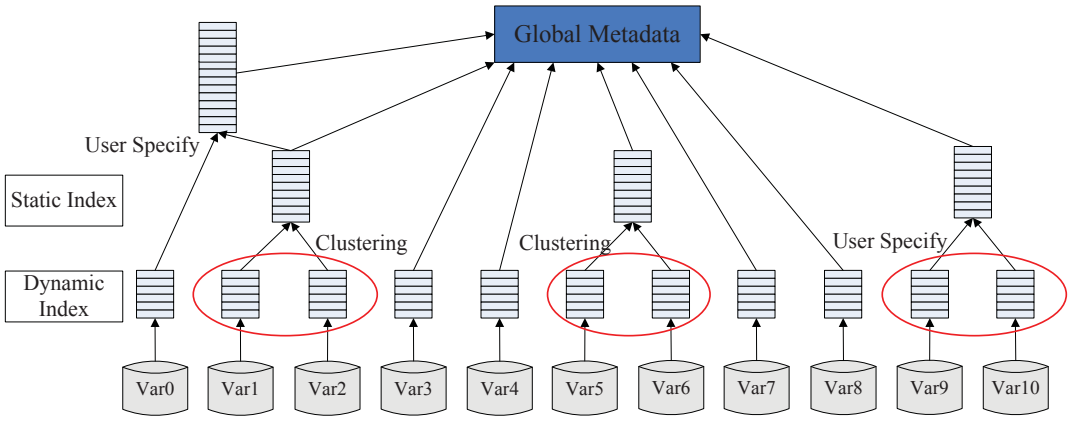

Figure 5: Hierarchical Bitmap Indexing

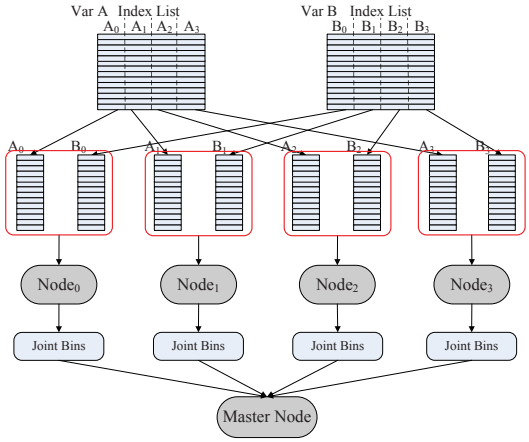

Figure 6: Dimension-based Partitioning Method

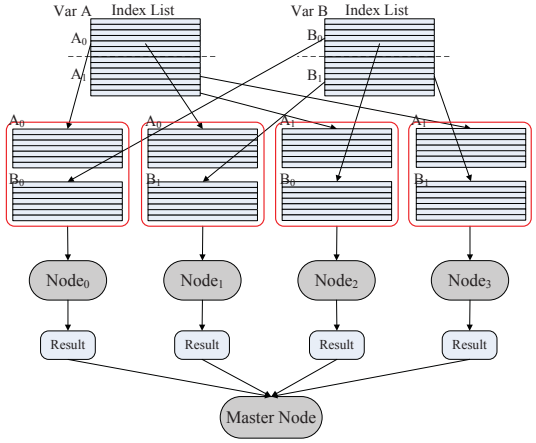

Figure 7: Value-based Partitioning Method

more efficiently. Moreover, our framework is flexible enough that if users want to add another variable into an existing cluster (support correlation analysis between one variable and one variable set), we can simply perform bitwise operations between the index of the single variable and static multi-variable index to generate a larger static index, such as the one involving Var_0, Var_1 and Var_2 in this figure. Finally, a global metadata is maintained to keep track of the entire hierarchy. When the system receives a query, the system can automatically find the right indexing method by looking up the global metadata.

3.6 Parallel Correlation Analysis

Our bitmap based sequential methods are the basis for the parallel and distributed algorithms we have developed. This section describes two parallelization methods (*dim-based partitioning* and *value-based partitioning*). We assume that the dynamic indexing method is being used - parallelizing the method based on static indexing is similar but also simpler.

Dim-based partitioning (referring to dimension-based partitioning) is the more straight-forward way of parallelization. Figure 6 shows the process of calculating correlation information between two variables in parallel using the *dim-based partitioning* method. The bitvectors for both variables A and B are first partitioned into 4 sub-index lists (A_0, A_1, A_2, and A_3; and B_0, B_1, B_2, and B_3) based on the dimensions. Each sub-index list corresponds to one data sub-block of the original dataset. Suppose there are 4 worker nodes ($Node_0$, $Node_1$, $Node_2$, and $Node_3$) in the parallel environment. Each node will be assigned with one sub-index of A (A_i) and one sub-index of B (B_i). For each correlation query, the bitwise operations between sub-index of A and B will be performed in parallel to generate the joint bins for data sub-blocks. After that, the joint bins will be sent to the master node, and the master node will calculate different correlation metrics based on that.

The advantage of this method is that it supports efficient indexing generation and analysis of each individual variable. In fact, during the index generation phase, instead of generating index-lists for the entire variable, the sub-index lists can be directly generated in parallel by different nodes. If subsetting conditions on individual variables are involved, they can also be easily applied. However, this method has a significant limitation during the stage when the correlations are computed. For several important correlation metrics such as the mutual information, the correlation results cannot be simply computed by taking counts from the sub-blocks of the data. Thus, each worker node must send the joint bins to the master node, which adds a large load on the network. Finally, master node has to perform an expensive global combination operation to calculate the metrics.

In view of this, we have designed a *value-based partitioning* method to support efficient parallel correlation analysis, as shown in Figure 7. Instead of generating sub-index lists based on dimensions, this method generates sub-index lists based on the partition of bitvectors (values). For example, suppose the total number of

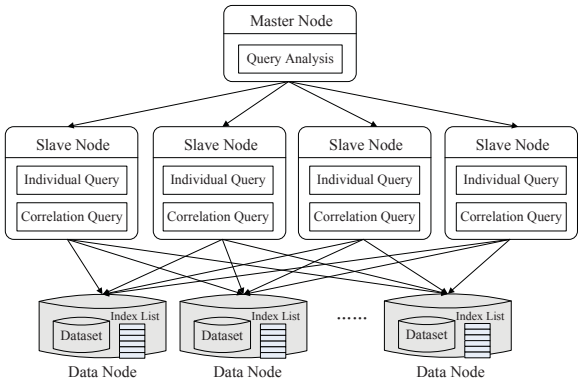

Figure 8: Correlation Analysis in a Distributed Environment

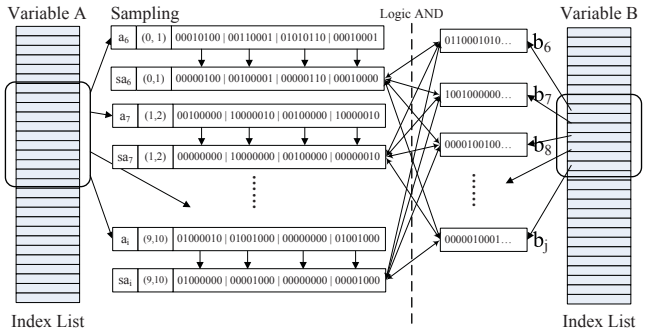

Figure 9: Sampling Using Bitvectors and Correlation Analysis

bitvectors for the variables A and B is 14 each. After partitioning, each sub-index list (A_0 and A_1; and B_0 and B_1) will have 7 bitvectors each. Now, each of the 4 worker nodes can be assigned a pair of sub-index, i.e., four sets (A_i,B_j) are created, where each of i and j can be 0 or 1.

Now, each worker node can perform bitwise operations on the pair that is assigned to it, generate joint bins, and calculate the correlation metrics results directly in parallel. The results of each worker nodes will be sent to the master node and master node only needs to combine the partial results together, which has very low cost. We can see that though there is some replication of data, the total number of bitwise operations for each worker node remains almost the same. Compared with the *dim-based partitioning* method, it has a very low network transfer cost and correlation calculation cost. The only disadvantage is that during the index generation phase, each worker node needs to scan the entire data block to generate indices for corresponding value sub-range.

3.7 Correlation Analysis in a Distributed Environment

Our bitvector based approach also forms the basis for correlation computations on datasets that are spread over geographically distributed repositories. The key advantage of our approach lies in the use of bitmaps as a space efficient summary of original dataset, which is also sufficient for calculating correlation metrics.

We further assume the following. The data over which correlations need to be computed are across multiple repositories, each of which could hold different variables or different dimensional partitions. We assume that bitvectors have been generated and stored within each repository together with the dataset, and further, they can be subset within the repository. We assume that no computational cycles are available at each repository, and thus, correlation computations can only be performed in a compute cluster. Figure 8 shows the environment.

The correlation query that needs to be processed is initially submitted to the master node. The master node decides how the computation and downloading of the data will be divided among the worker nodes. Accordingly, subsets of available bitvectors are downloaded from the data repositories. Subsequently, the rest of the processing is just like parallel computation of bitvectors, which we have already discussed. The key advantage of the approach is that the amount of data downloaded from the repositories is significantly smaller, leading to overall reduction in the time required for computing the correlations.

3.8 Correlation Analysis over Samples

As dataset sizes are growing rapidly, analyzing the entire dataset is often not feasible. It turns out that an added benefit of bitvectors is that they allow sampling to be performed, in a fashion that value distribution is preserved in the sample.

We describe index-based sampling method and its application to correlation analysis using an example, which is shown in Figure 9. In this example, we still want to generate the correlation results between variable A and variable B. Instead of correlation analysis over all data elements, we want to perform correlation analysis over only 50% of the data. Bitvectors of variable A and B that satisfy the current query are selected and loaded into the memory. From the figure, we can see that this small dataset contains 32 elements, so each bitvector has 32 bits. The bitvectors of A are $a_6(0,1)$, $a_7(1,2)$, $a_8(2,3)$, . . ., $a_i(9,10)$, and the bitvectors of B are b_6, b_7, b_8, . . ., b_j.

Though one can sample data corresponding to each variable and then perform correlation computation, obtaining representative samples for each variable (preserving the value-distributions) can be expensive in practice. Thus, we select only one variable, which is the variable with a smaller number of bitvectors, and generate samples based on it. This allows sampling to be efficient, and yet, we get a favorable reduction in the computation time, while preserving accuracy, for the correlation computation step. The bitvector based sampling we perform on each variable is as follows. Our goal is to preserve distribution of values in each spatial region. For this purpose, we divide bitmap indices into *spatial sectors*. In the figure, we can see that the variable A is selected as the variable for sampling, and every selected bitvector of A is divided into 4 sectors, such that there are 8 bits within each sector.

After creating these sectors, sampling is applied within each sector for each bitvector. Particularly, within each bitvector, a desired fraction (sampling rate) of bits that are 1 are chosen. This ensures that the value distribution within each sector is maintained. In Figure 9, we are generating 50% samples out of original dataset. We can see that sa_6, sa_7, . . ., sa_i are identifiers of data records that are in the sample generated, and only half of bits that have the value 1 are picked. For example, after sampling, the number of bits 1 in the sample bitvector sa_6 is 6, which is only half of that in original bitvector a_6. After that, the bitwise operations are performed between the sampled bitvectors of A and the original bitvectors of B, following the same steps as in the dynamic indexing method based on the entire data. This method still reduces the time spent on creating joint bins and computing correlations, as we will show through experimental results.

4. PUTTING IT TOGETHER: SYSTEM OVERVIEW

The approach and algorithms discussed in the previous section have been put together in an interactive system. The system supports a high-level query interface, and can provide incremental analysis by maintaining bitvectors for the last several queries.

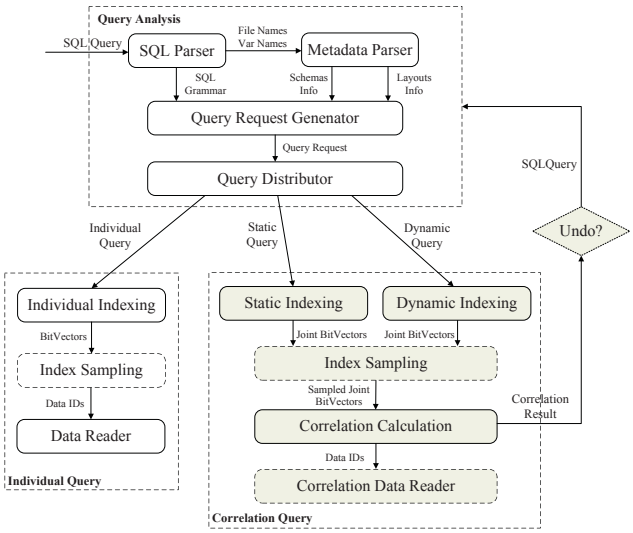

Figure 10: System Overview

Figure 10 shows a high-level overview of our system, which can be divided into three modules: *Query Analysis Module*, *Individual Query Module* and *Correlation Query Module*. The *Query Analysis Module* takes structured queries (using an SQL-like syntax) as the input, generates internal query requests by parsing the query and analyzing the corresponding metadata. *Individual Query Module* supports flexible data subsetting and sampling over each variable [22, 23]. *Correlation Query Analysis* supports the interactive correlation queries among multiple variables, which is the main contribution of this work. It contains five components:

1) Indexing Service: The bitmap indexing services, irrespective of whether they are static or dynamic indexing, take the query request as the input, perform indexing operations and generate joint bins across the variables involved as the output. The joint bins are used in the *Correlation Calculation* module to compute different correlation metrics.

2) Sampling Service: Sampling is used when the entire dataset cannot be analyzed in a timely fashion. This module performs data sampling directly over the bitvectors and outputs only a small sample of the data (in the form of *sampled joint bitvectors*). Then, the correlation calculation is performed based on this sample. As we described in Section 3.8, bitvector based sampling can generate "accurate" samples, and thus they provide the flexibility of accelerating correlation analysis with only a small sacrifice in the accuracy.

3) Correlation Calculation: This component calculates different correlation metrics (histogram, entropy, mutual information) based on the joint bitvectors (bins). One feature that is supported in our current implementation and has not been discussed so far is the use of *multi-level* indexing. A multi-level index uses a coarse-grained binning at the high-level level and a fine-grained binning at the lower-level. Subsequently, low-level bitvectors can be used to generate more accurate calculation of the correlations, though with additional time cost. In contrast, high-level bitvectors provide much faster response time, but generate correlation information in a coarse level.

4) Support for Incremental Analysis and "Undo": Two important features of the system for supporting interactive analysis are - incremental analysis and an "Undo" operation. After users have seen results on a particular subset of data, they can further specialize in the query by adding another condition. Users can also do an "Undo" to step back to the earlier results, and specify additional conditions for incremental analysis on top of them. This functionality is supported efficiently by keeping the query bitvectors at each step (or a certain k previous steps). On one hand, new subsetting conditions can be applied on top of the bitvectors stored from the previous steps. At the same time, because the size of bitvectors is much smaller than the dataset, we can store bitvectors from the last several steps without very high overheads.

5) Data Reader: Our system also allows users to view the actual data. Suppose during interactive correlation analysis, users specify a subset of data that turns out to be intriguing. Because our system records the bitvectors corresponding to this subset, users can obtain the original data corresponding to this subset efficiently.

5. EXPERIMENTAL RESULTS

In this section, we report results from a number of experiments conducted to evaluate our correlation analysis approach and algorithms. We designed experiments with the following goals: (1) We compare the correlation analysis efficiency among the original *no indexing*, *dynamic indexing*, and *static indexing* methods in a sequential environment, and show that correlation analysis with the help of bitmap indexing can improve the efficiency. (2) We show the scalability of our parallel indexing method with the increasing number of nodes and compare the *value-based partitioning* method with the *dim-based partitioning* method. (3) We show that in an environment where data is stored on geographically distributed repositories, our method is able to speed up the correlation analysis process compared to a simple method that does not use any indexing. (4) We show that if correlation analysis is performed over samples, and not the entire dataset, what kind of speedup we can achieve and how much accuracy is lost.

The dataset we used here is generated by the Parallel Ocean Program (POP) [15], which is an ocean circulation model. The simulation we used has a grid resolution of approximately 10 km (horizontally), and vertically it has a grid spacing close to 10 m near the surface, increasing up to 250 m in the deep ocean. POP generates 1.4 GB output for each variable per time-slice. The total number of variables in the dataset is 26, and each variable is modeled with either two dimensions (longitude and latitude) or three dimensions (longitude, latitude, and depth). The data is stored in the NetCDF format. The size of bitmap indices ranges from 12.1% to 26.8% compared to the size of its corresponding variable. The total number of bitvectors of each variable is from 203 to 431, and the bitvectors generation time ranges from 112 to 187 seconds for each variable per time-slice, depending on the value ranges of variables. For bigger data size, parallel index generation can be applied to improve the efficiency. Also index generation can be treated as a preprocessing step, since once these indices have been calculated, they can be used for a variety of queries, not limited to correlation analysis (for example, subsetting [21] and sampling [22]). We chose the same binning scales for the *no indexing*, *dynamic indexing* and *static indexing* methods so that the correlation results of all methods are same. All of our experiments were conducted on the Glenn cluster from Ohio Supercomputing Center, where every node has 8 cores, 2.6 GHz AMD Opteron(TM) processors, with 64 GB RAM and 1.9 TB local disk space.

5.1 Efficiency Improvement Using Bitmap Indexing

The experiments in this subsection compare the correlation analysis time among the original or the *no indexing* method, the *dynamic indexing* method, and the *static indexing* method. Two variables were chosen and we calculate entropy of each variable and the 2D histogram and the mutual information between them.

Figure 11 shows the correlation analysis time among these three methods based on different queries. We selected 1000 queries with different dimension-based and value-based subsetting conditions,

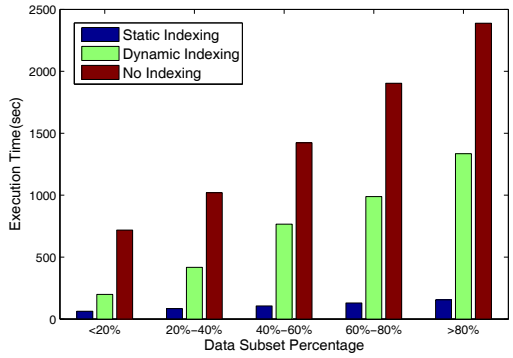

Figure 11: Comparison of Correlation Analysis Time with Queries with Different Subsetting Levels

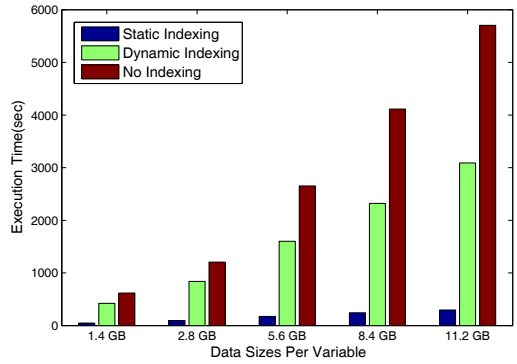

Figure 12: Comparison of Correlation Analysis Time with Queries with Different Dataset Sizes

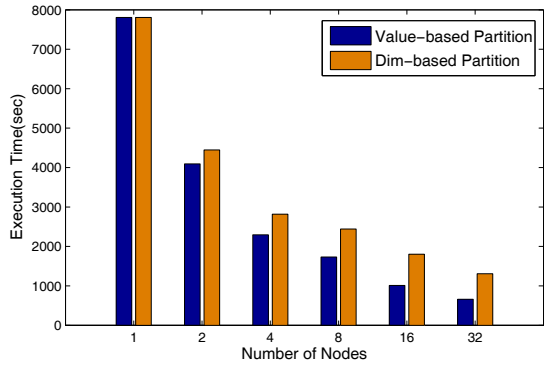

Figure 13: Parallel Correlation Analysis - Varying Number of Nodes

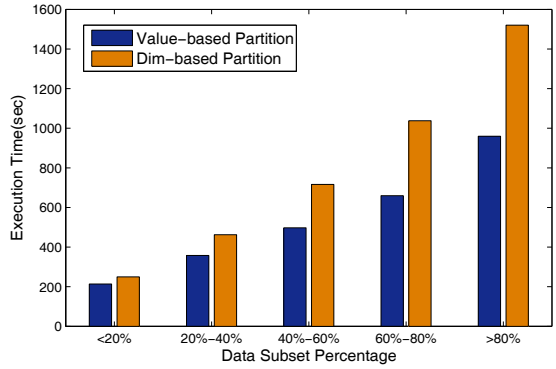

Figure 14: Parallel Correlation Analysis - Queries with Different Subsetting Levels

and then divide them into five categories based on data subsetting percentage (<20%, 20%-40%, 40%-60%, 60%-80%, >80%). Each variable is 5.6 GB (4 timestamps merged together) in size. The time cost of the original or the *no indexing method* includes the data loading time, the data filtering time (where subsetting conditions are applied), the joint bins generation time, and finally, the correlation metrics calculation time. Without indexing support, the entire datasets for the two variables involved have to be loaded into the memory, and subsequently, filtering conditions are applied to examine each data element. The joint bins are generated by binning over each element within the data subset, which is also time consuming.

The time cost of *dynamic indexing* includes bitvectors subset loading time, bitwise operation time to generate joint bins, and the correlation metrics calculation time. Compared with *no indexing method*, *dynamic indexing* method has much lower data loading time, as the size of the bitvectors is much smaller than the size of the data block, and only a subset of bitvectors that satisfy the current query conditions need to be loaded into the memory. Moreover, the joint bins are generated based on fast bitwise operations. This is reflected in the experiments, and from the figure, we can see that, irrespective of the data subsetting percentage, the *dynamic indexing* method always achieves better efficiency than the *no indexing method*. The speedup factor varies from 1.78x to 3.61x, becoming smaller as data subset percentage increases. This is because the larger the subsetting percentage is, a larger fraction of bitvectors have to be loaded into the memory, and more bitwise operations need to be performed. The time cost of the *static indexing* method includes only the joint bitvectors loading time and the correlation metrics calculation time. Compared with *no indexing* method, the speedup is from 11.4x to 15.35x.

Figure 12 shows the correlation analysis time among three methods over different sizes of the data. Here the correlation metrics calculation is over the entire data blocks of two variables, without any subsetting. The size of the dataset for each variable ranges from 1.4 GB to 11.2 GB. From the figure we can see that even without data subsetting, for all different cases, both *dynamic indexing* and *static indexing* methods perform well, with their advantage even increasing as the dataset sizes increase. This is because our indexing based methods require less memory.

5.2 Scalability of Parallel Indexing

The experiments in this subsection show the speedup of parallel correlation analysis using multiple nodes. During this evaluation, we also compare the efficiency of *value-based partitioning* method with the *dim-based partitioning* method. Correlation analysis is performed over two variables, and the data sizes for each is 28 GB (20 timestamps merged together). We use between 1 and 32 nodes for our experiments, and because the method is memory-intensive, only 1 core per node is used.

Figure 13 shows the scalability of our parallel correlation analysis method with different number of nodes. The X axis shows number of nodes (from 1 to 32) used. The correlation calculation here is over the entire data blocks (all elements), i.e., no subsetting condition is involved. From the figure we can see that both *value-based partitioning* and *dim-based partitioning* methods show good speedup as number of nodes increases. For the *value-based partitioning* method, the speedup using 2, 4, 8, 16, and 32 nodes is 1.87x, 3.4x, 4.53x, 7.68x, and 11.79x, respectively. For *dim-based*

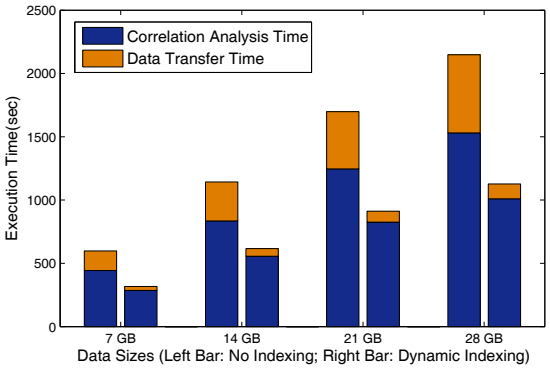

Figure 15: Distributed Analysis - Data Downloaded from a "Local" Data Server with 1 Gb/sec Bandwidth

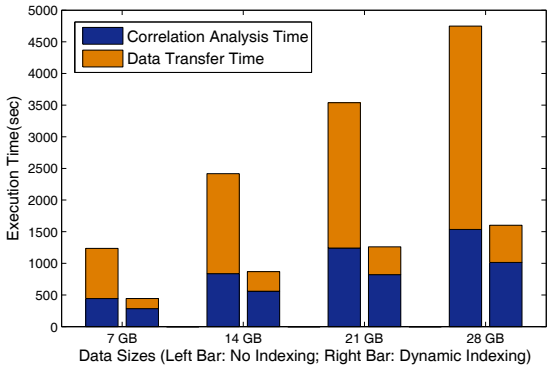

Figure 16: Distributed Analysis - Data Downloaded from a "Remote" Data Server with 200 Mb/sec Bandwidth

partitioning method, the speedup using 2, 4, 8, 16, and 32 nodes is 1.73x, 2.77x, 3.18x, 4.32x, and 5.96x, respectively. The reason for higher efficiency of *value-based partitioning* is because the master node is a bottleneck for *dim-based partitioning*. Note that the speedups for *value-based partitioning* are still not close to linear. This is because different bitvectors can have different number of 1s, which leads to different amounts of time for the bitvector operations.

Figure 14 shows the efficiency of both partitioning methods with different queries, which are then classified with respect to the subsetting percentage involved. The number of nodes here is 16. From the figure we can see that for both methods, the execution time increases as data subset percentage increases, as we expect. However, if we compare the two partitioning methods, we can see that the relative improvement from the *value-based partitioning* method becomes more significant than the *dim-based partitioning* method as data subsetting percentage increases (relative improvement ranges between 1.17x to 1.58x). The reason is that as data subsetting percentage increases, the number of joint bins generated by each process also increases, which imposes a more significant network overhead for the *dim-based partitioning* method.

5.3 Efficiency Improvement in Distributed Environment

The experiments in this subsection analyze the efficiency of performing correlation analysis in a distributed environment, where data is stored over geographically separated data servers, and analysis is performed over a single cluster. In such a case, without bitmap indexing, datasets corresponding to variables involved need to be downloaded to the cluster used for the computations. Instead, if the indices have been generated and stored together with the dataset, only the index files need to be downloaded to the cluster. In our experiment, we use 16 compute nodes for parallel correlation calculation. We use a *local* data server (1 Gb/sec bandwidth connection to the compute-cluster) and a *remote* data server (200 Mb/sec bandwidth connection to the compute-cluster) in separate sets of experiments.

Figure 15 compares the total time between the *dynamic indexing* method and the *no indexing* method using local data server. The data size of each variable ranges from 7 GB to 28 GB. No subsetting is involved. The total execution time with either of the methods can be divided into two parts: the network data transfer time and the parallel correlation analysis time. As expected, our method reduces the data transfer time, because only bitvectors are being downloaded, and not the full dataset. For parallel correlation analysis, to make a fair comparison, both methods used *value-based partitioning*, i.e., without indexing, the full dataset is partitioned on the basis of the values. While partitioning bitvectors based on values is trivial, there is a processing cost associated with partitioning the dataset based on values. We do not report the extra value partitioning time for the *no indexing* method. We can see that our method is still able to achieve much better efficiency than the *no indexing* method, because our method has much smaller memory data loading cost and the computation costs for joint bins are also lower. The overall improvement ranges from 1.87x to 1.91x.

Figure 16 compares the total time between the *dynamic indexing* method and the *no indexing* method using a remote data server, with lower (200 Mb/sec) bandwidth. From the figure, we can see that while the parallel correlation analysis time of both methods is similar (as in the previous experiment), our method further improves the efficiency by saving the data transfer time. The overall speedup ranges from 2.78x to 2.96x.

It should be further noted that our experiments did not include any data subsetting. If correlation queries involve any subsetting condition, the size of the bitvectors involved can be reduced further. In comparison, a data repository without any indexing support may not allow any subsetting, and users may have to download all the data for each variable being analyzed.

5.4 Efficiency and Accuracy Comparison with Sampling

Our last set of experiments compares the efficiency and accuracy of performing correlation analysis with different sampling levels (sampling applied to bitvectors). We selected 10 variables (each has three dimensions with 1.4 GB in size) from the POP dataset, and calculated the mutual information between each distinct pair of them. The total number of such pairs is 45. Dynamic indexing is used in these experiments.

Figure 17 shows the efficiency of calculating mutual information for 45 variable pairs with different samples. The X axis shows different sample percentages (100%, 50%, 25%, 10%, 5%, 1%) compared with original data size, and the Y axis shows the entire correlation analysis time. Overall, we can see that the entire analysis efficiency greatly improves as sample size becomes smaller. Compared with correlation analysis over the original dataset (100%), the speedup using 50%, 25%, 10% 5%, and 1% samples is 1.34x, 1.69x, 2.17x, 2.93x, and 6.84x, respectively. While the steps for constructing joint bins and calculating the metrics are accelerated by almost the same factor as the sampling level, the cost of reading the original bitvectors remains almost the same, and an additional cost of sampling is introduced.

Figure 18 shows the accuracy of mutual information results using different sampling levels. Here we generated 45 mutual information results (45 pairs) for each sampling level. We compute

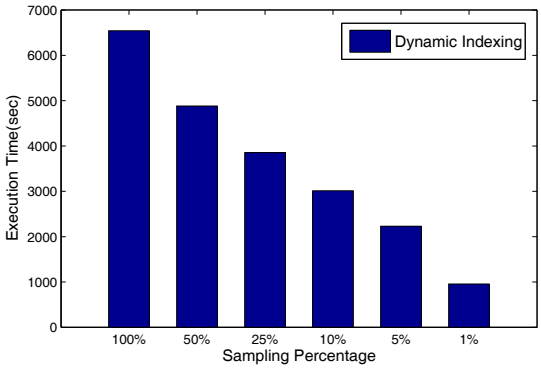

Figure 17: Efficiency Comparison with Different Sampling Levels

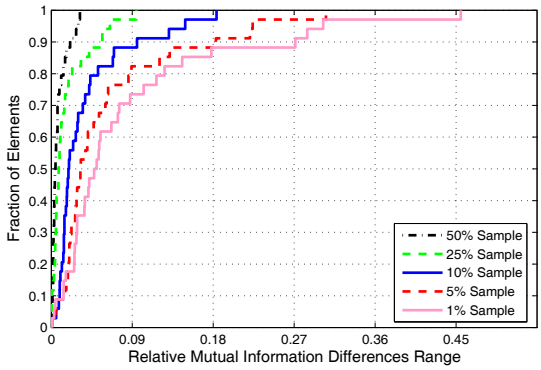

Figure 18: Cumulative Frequency Plot to Show Accuracy Comparison with Different Sampling Levels

the relative value differences using expression $(original\_result - sample\_result)/original\_result$ for each pair. Then, we use a *Cumulative Frequency Plot* (CFP) to represent the relative mutual information differences for all pairs. In CFP, a point (x, y) indicates that the fraction y of all calculated relative value differences are less than x. Because the value differences should be as small as possible, it implies that a method with the curve to the left has a better accuracy than the method with the curve to the right. From the figure we can see that the accuracy using 50% samples is very close to the original dataset, as errors are very close to 0 for almost all points. Not surprisingly, accuracy becomes worse as sparser samples are taken. If we calculate the average accuracy lost based on the CFP, we find that the average accuracy loss with 50%, 25%, 10%, 5% and 1% sample is 1.53%, 3.42%, 7.91%, 12.57%, and 18.32%, respectively. Because our bitvector based sampling method preserves distribution of values, reasonably high accuracies are maintained even with a small sample of data. Overall, our sampling method provides a way to accelerate the computations while obtaining reasonably accurate results. Sampling can also provide a way to further reduce the data transfer volumes when the data is stored in geographically distributed servers. In fact, as our previous work has shown [22], bitmap indexing can be used to perform server-side sampling of the data.

6. CASE STUDIES DEMONSTRATING EFFICACY OF TOOL

All results presented in the previous section focused on demonstrating the efficiency of the methods. In this section, our goal is to demonstrate how the tool is useful to the scientists. Particularly, we show that the ability to compute correlations over various subsets through an intuitive and high-level query interface makes our tools extremely valuable for scientific discoveries.

The studies were conducted at the Los Alamos National Laboratory (LANL), using datasets generated from the POP simulation. Among the two scientists, one is a physical chemist interested in studying bio-geochemistry data, such as the production ("PROD") and sinking ("FLUX-IN") of Silicate (SiO2), Calcium Carbonate (CaCO3), and Particulate Organic Carbon (POC), as well as Oxygen (O2) production ("PROD") and consumption ("CONSUMP"). The second scientist focused on studying the relationship between Temperature (TEMP) and Salinity (SALT), which are the two basic elements of the ocean data. Using our system, they are able to explore different correlations over different areas or value ranges by submitting different queries, and our system returns correlation metrics results (histogram, entropy, and mutual information) as the output.

Figure 19 shows different correlation results our system presented to the scientists. The description below focuses on showing (in intuitive terms) the observations that can be made from the data.

Subfigure 19(a) shows the histogram of $SALT$ based on different value ranges of $TEMP$. Let us first look at the $SALT$ value along the X axis. Here we define p as the value on the X axis where the curve has the highest Y value. When $TEMP < 5$, the p value of *SALT* is around 0.0349. However, as the temperature increases, the salinity decreases. When $TEMP >= 5$ and $TEMP < 10$, the p value decreases to 0.0342. After that, the salinity increases as the temperature increases. When $TEMP >= 10$ and $TEMP < 15$, the p value increases to 0.0345, and then when $TEMP >= 15$, the p value of $SALT$ increases to 0.035. Thus, salinity is high when the temperature is either low or high. After talking to the scientists, we find that this is because on one hand, the deeper into the ocean, the colder the water gets, and the water also gets denser, which implies a higher salinity. On the other hand, the surface salinity also increases in areas close to the equator, because of hotter air. This happens because water evaporates faster, leaving more salt to a smaller amount of water. Moreover, if we look at the Y axis, we are able to see the diversity of the salinity within different temperature ranges. For example, when $TEMP < 5$, over 23% of the data elements' value is around 0.035. However, when $TEMP >= 15$, only around 6% of the data elements' value is around 0.035.

Subfigure 19(b) shows the histogram of $SALT$ based on different areas of the ocean (dimension subsets). From the figure we can see that the p value of $SALT$ within the Atlantic Ocean and the Mediterranean Sea areas is larger than the p value within the Pacific Ocean and the Indian Ocean, which reflects the actual features of these areas.

Subfigure 19(c) shows the histogram of $SALT$ based on different value ranges of $TEMP$ inside the Mediterranean Sea. This figure is used to show that our system is able to perform correlation analysis over flexible combinations of dimension subsets and value subsets. From the figure we can see that although the value subrange of $TEMP$ is the same as in subfigure 19(a), the relationship between $TEMP$ and $SALT$ has changed: particularly, now salinity increases as the temperature increases. This is because this area is warm.

Subfigure 19(d) shows the histogram of sink volumes of Silicate $SiO2\_FLUX\_IN$ based on different sink volumes of Calcium Carbonate $CaCO3\_FLUX\_IN$. We can see that the p value of $CaCO3\_FLUX\_IN$ increases as $SiO2\_FLUX\_IN$ increases. This is because more sinks of Silicate implies more plankton in this area, and more plankton implies more Calcium Carbonate generated and more sinks of Calcium Carbonate. Hence, 2D histogram is able to help scientists find different value distributions of one variable based on value changes of another.

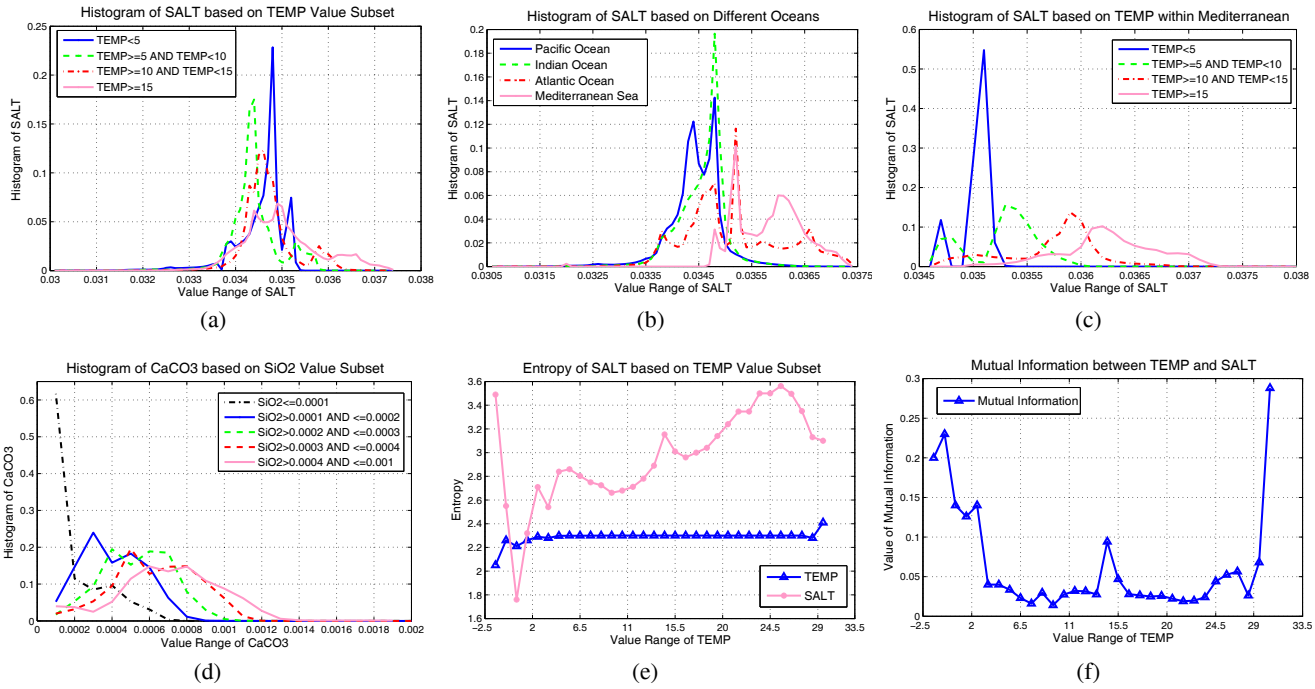

Figure 19: Correlation Metrics Results over Data Subsets

Subfigure 19(e) shows an example of using entropy for correlation analysis. The goal here is to see the changes of entropies of $SALT$ with respect to the different value subsets of $TEMP$. A total of 30 queries are submitted and each query specifies a $TEMP$ subset with similar value intervals, which makes the entropy values of $TEMP$ similar. From the figure we can see that while the entropy values of $TEMP$ are similar, the entropy values of $SALT$ have large differences. When the $TEMP$ value is around one centigrade, entropy of $SALT$ is the lowest, and it increases as the $TEMP$ value increases. In another word, the value of salinity is more constant and predicable as temperature is around one centigrade, whereas one sees a diversity of salinity values as the temperature increases.

Finally, subfigure 19(f) shows the mutual information between $TEMP$ and $SALT$ based on the value distribution of $TEMP$. From the figure, we can see that mutual information between $TEMP$ and $SALT$ is high when the temperature values are either low or high. This implies that $TEMP$ and $SALT$ are highly correlated within these two value ranges. For the other value ranges, where the mutual information is close to 0, correlation is very small between the two variables.

Overall, the representative results above show that correlation analysis over flexible subsets of data can help scientists confirm known facts, and even make new observations.

7. RELATED WORK

Our work has some similarities to a number of efforts from high performance data management area, as well as visualization.

Closely related to our work, Fastbit [30] and FastQuery [8] apply bitmap indexing and parallel indexing to support efficient value-based subsetting (for individual variables). Our work builds on top of these, but is unique in applying bitvectors for correlation analysis. There has also been a growing trend towards building database-like functionality on top of native storage of the data. Examples include the NoDB approach [1] and automatic data virtualization [28]. Our work is an example of this approach, but, again, unique in its focus on correlation analysis. In recent years, many Array DBMSs, including SciDB [6] and RasDaMan [3] have been designed, and are gaining popularity. None of these systems have provided support for correlation analysis across variables.

In scientific data analysis, much of the recent focus has been on *in-situ* analysis, with ADIOS project providing a mature implementation of this approach [16]. DIRAQ [17] provides a parallel *in-situ*, in network data encoding and reorganization technique that enables the transformation of simulation output into a query-efficient form. In the future, we will like to develop methods for *in-situ* correlation analysis. Other tools for scientific data analysis include OPeNDAP [9], which provides data virtualization through a data access protocol and data representation. SciHadoop [7] and SciMATE [27] integrate map-reduce and its variant with scientific library to enable map-reduce tasks over scientific data. Scientific Data Manager (SDM) [19] employs the Metadata Management System (MDMS) and provides a programming model to abstract low-level parallel I/O operations for complex scientific processing. FASM [18] utilizes statistical metadata with various subsetting schemes to perform efficient analyses on large datasets. None of these efforts have considered correlation analysis.

Analysis of multiple variables and their relationships in scientific simulation outputs has been an ongoing topic of research. In a very recent work, Biswas *et al.* [4] have presented an information theoretic framework for exploring multivariate datasets in a "top-down" manner, where they divide the variables into groups based on their information overlap, and then identify representative variables from each group to conduct further relationship analysis. Wang *et al.* [26] used information theory for exploring the causal relationship among the variables of a time-varying multivariate dataset. In an earlier work, Jänicke *et al.* [14] applied the dimensionality reduction on the high dimensional data where each

dimension was analogous to a variable in the multivariate context. Another well-known multivariate exploration technique was developed by Di Yang *et al.* [33]. They use a Nugget Management System (NMS), where the nuggets represent the information that the users are interested in. Several authors have surveyed existing multivariate data analysis techniques [29, 11]. Almost none of these efforts have focused on scalability limitations, especially, what happens when the data does not fit into memory, or if the data is stored in geographically distributed repositories. Parallelization of the methods is another challenge that has not been addressed.

8. CONCLUSIONS

While the potential of data-driven discoveries for scientific advances is being increasingly recognized, several trends are making interactive and efficient data analysis hard. On one hand, the amount of data generated by scientific simulation (or instruments) is rapidly increasing. On the other hand, computing environments are becoming more and more constrained with respect to data movement (at all levels).

This paper has focused on the problem of correlation analysis in parallel and distributed settings. We have developed a series of techniques, with the main underlying idea that bitmap indices can serve as a concise and representative summary of the original dataset, allowing computation of the correlation metrics more efficiently. Our algorithms have been incorporated in a system that offers a high-level API, where users can interactively choose subsets of the data to be analyzed, and sampling can be combined with correlation analysis. We have extensively evaluated our system and shown the efficiency improvement using our method. We also conducted a user-evaluation with domain scientists and demonstrated how our system is able to aid the data-driven discovery process.

9. ACKNOWLEDGMENTS

This work was partially funded by the Department of Energy (DOE) Office of Science (OSC) Advanced Scientific Computing Research (ASCR) and also partially funded by NSF award ACI-1339757 to the Ohio State University.

10. REFERENCES

[1] Ioannis Alagiannis, Renata Borovica, Miguel Branco, Stratos Idreos, and Anastasia Ailamaki. NoDB: efficient query execution on raw data files. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 241–252, 2012.

[2] G. Antoshenkov. Byte-aligned bitmap compression. In *Data Compression Conference, 1995. DCC'95. Proceedings*, page 476. IEEE, 1995.

[3] P. Baumann, A. Dehmel, P. Furtado, R. Ritsch, and N. Widmann. The Multidimensional Database System RasDaMan. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, pages 575–577, 1998.

[4] Ayan Biswas, Soumya Dutta, Han-Wei Shen, and Jonathan Woodring. An information-aware framework for exploring multivariate data sets. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2683–2692, 2013.

[5] Udeepta D Bordoloi and H-W Shen. View selection for volume rendering. In *Visualization, 2005. VIS 05. IEEE*, pages 487–494. IEEE, 2005.

[6] Paul G. Brown. Overview of SciDB: large scale array storage, processing and analysis. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, pages 963–968, 2010.

[7] J. Buck, N. Watkins, J. LeFevre, K. Ioannidou, C. Maltzahn, N. Polyzotis, and S. Brandt. Scihadoop: Array-based query processing in hadoop. In *SC*, 2011.

[8] J. Chou, K. Wu, O. Rübel, M.H.J.Q. Prabhat, B. Austin, E.W. Bethel, R.D. Ryne, and A. Shoshani. Parallel index and query for large scale data analysis. In *SC*, 2011.

[9] P. Cornillon, J. Gallagher, and T. Sgouros. Opendap: Accessing data in a distributed, heterogeneous environment. *Data Science Journal*, 2(0):164–174, 2003.

[10] Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[11] M.C.F. de Oliveira and H. Levkowitz. From visual data exploration to visual data mining: a survey. *Visualization and Computer Graphics, IEEE Transactions on*, 9(3):378–394, 2003.

[12] Stefan Gumhold. Maximum entropy light source placement. In *Visualization, 2002. VIS 2002. IEEE*, pages 275–282. IEEE, 2002.

[13] Gheorghi Guzun, Guadalupe Canahuate, David Chiu, and Jason Sawin. A tunable compression framework for bitmap indices. In *Proceedings of the 30th international conference on data engineering (ICDE)*. IEEE, 2014.

[14] H. Jänicke, M. Bottinger, and G. Scheuermann. Brushing of attribute clouds for the visualization of multivariate data. *Visualization and Computer Graphics, IEEE Transactions on*, 14(6):1459–1466, 2008.

[15] PW Jones, PH Worley, Y. Yoshida, JB White III, and J. Levesque. Practical performance portability in the parallel ocean program (pop). *Concurrency and Computation: Practice and Experience*, 17(10):1317–1327, 2005.

[16] Scott Klasky, Hasan Abbasi, Jeremy Logan, Manish Parashar, Karsten Schwan, Arie Shoshani, Matthew Wolf, Sean Ahern, Ilkay Altintas, Wes Bethel, et al. In situ data processing for extreme-scale computing. In *Proc. Conf. Scientific Discovery through Advanced Computing Program (SciDACÂŠ11)*, 2011.

[17] Sriram Lakshminarasimhan, David A Boyuka, Saurabh V Pendse, Xiaocheng Zou, John Jenkins, Venkatram Vishwanath, Michael E Papka, and Nagiza F Samatova. Scalable in situ scientific data encoding for analytical query processing. In *Proceedings of the 22nd international symposium on High-performance parallel and distributed computing*, pages 1–12. ACM, 2013.

[18] Jialin Liu and Yong Chen. Fast data analysis with integrated statistical metadata in scientific datasets. In *2013 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 1–8. IEEE, 2013.

[19] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E.A. Lee, J. Tao, and Y. Zhao. Scientific workflow management and the kepler system. *Concurrency and Computation: Practice and Experience*, 18(10):1039–1065, 2006.

[20] P. O'Neil and D. Quass. Improved query performance with variant indexes. In *ACM Sigmod Record*, volume 26, pages 38–49. ACM, 1997.

[21] Y. Su, G. Agrawal, and J. Woodring. Indexing and parallel query processing support for visualizing climate datasets. In *2012 41th IEEE/ACM International Conference on Parallel Processing (ICPP)*, pages 249–258. IEEE, 2012.

[22] Yu Su, Gagan Agrawal, Jonathan Woodring, Kary Myers, Joanne Wendelberger, and James Ahrens. Taming massive distributed datasets: data sampling using bitmap indices. In *Proceedings of the 22nd international symposium on High-performance parallel and distributed computing*, pages 13–24. ACM, 2013.

[23] Yu Su, Yi Wang, Gagan Agrawal, and Rajkumar Kettimuthu. Sdquery dsi: integrating data management support with a wide area data transfer protocol. In *Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis*, page 47. ACM, 2013.

[24] Pere-Pau Vázquez, Miquel Feixas, Mateu Sbert, and Wolfgang Heidrich. Automatic view selection using viewpoint entropy and its application to image-based modelling. In *Computer Graphics Forum*, volume 22, pages 689–700. Wiley Online Library, 2003.

[25] Ivan Viola, Miquel Feixas, Mateu Sbert, and Meister Eduard Groller. Importance-driven focus of attention. *Visualization and Computer Graphics, IEEE Transactions on*, 12(5):933–940, 2006.

[26] Chaoli Wang, Hongfeng Yu, Ray W Grout, Kwan-Liu Ma, and Jacqueline H Chen. Analyzing information transfer in time-varying multivariate data. In *Pacific Visualization Symposium (PacificVis), 2011 IEEE*, pages 99–106. IEEE, 2011.

[27] Yi Wang, Wei Jiang, and Gagan Agrawal. Scimate: A novel mapreduce-like framework for multiple scientific data formats. In *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*, pages 443–450. IEEE, 2012.

[28] Li Weng, Gagan Agrawal, Umit Catalyurek, Tahsin Kurc, Sivaramakrishnan Narayanan, and Joel Saltz. An Approach for Automatic Data Virtualization. In *Proceedings of the Conference on High Performance Distributed Computing (HPDC)*, June 2004.

[29] Pak Chung Wong and R. Daniel Bergeron. 30 years of multidimensional multivariate visualization. In *Scientific Visualization, Overviews, Methodologies, and Techniques*, pages 3–33, Washington, DC, USA, 1997. IEEE Computer Society.

[30] K. Wu, W. Koegler, J. Chen, and A. Shoshani. Using bitmap index for interactive exploration of large datasets. In *15th International Conference on Scientific and Statistical Database Management*, pages 65–74. IEEE, 2003.

[31] K. Wu, E.J. Otoo, and A. Shoshani. Compressing bitmap indexes for faster search operations. In *Scientific and Statistical Database Management, 2002. Proceedings. 14th International Conference on*, pages 99–108. IEEE, 2002.

[32] K. Wu, K. Stockinger, and A. Shoshani. Breaking the curse of cardinality on bitmap indexes. In *Scientific and Statistical Database Management*, pages 348–365. Springer, 2008.

[33] Di Yang, E.A. Rundensteiner, and M.O. Ward. Analysis guided visual exploration of multivariate data. In *Visual Analytics Science and Technology, 2007. VAST 2007. IEEE Symposium on*, pages 83–90, 2007.