Taming Massive Distributed Datasets: Data Sampling Using Bitmap Indices

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

Kary Myers
Los Alamos National Laboratory
Los Alamos, NM 87544
kary@lanl.gov

Joanne Wendelberger
Los Alamos National Laboratory
Los Alamos, NM 87544
joanne@lanl.gov

James Ahrens
Los Alamos National Laboratory
Los Alamos, NM 87544
ahrens@lanl.gov

ABSTRACT

With growing computational capabilities of parallel machines, scientific simulations are being performed at finer spatial and temporal scales, leading to a data explosion. The growing sizes are making it extremely hard to store, manage, disseminate, analyze, and visualize these datasets, especially as neither the memory capacity of parallel machines, memory access speeds, nor disk bandwidths are increasing at the same rate as the computing power. Sampling can be an effective technique to address the above challenges, but it is extremely important to ensure that dataset characteristics are preserved, and the loss of accuracy is within acceptable levels.

In this paper, we address the data explosion problems by developing a novel sampling approach, and implementing it in a flexible system that supports server-side sampling and data subsetting. We observe that to allow subsetting over scientific datasets, data repositories are likely to use an indexing technique. Among these techniques, we see that bitmap indexing can not only effectively support subsetting over scientific datasets, but can also help create samples that preserve both value and spatial distributions over scientific datasets. We have developed algorithms for using bitmap indices to sample datasets. We have also shown how only a small amount of additional metadata stored with bitvectors can help assess loss of accuracy with a particular subsampling level. Some of the other properties of this novel approach include: 1) sampling can be flexibly applied to a subset of the original dataset, which may be specified using a value-based and/or a dimension-based subsetting predicate, and 2) no data reorganization is needed, once bitmap indices have been generated. We have extensively evaluated our method with different types of datasets and applications, and demonstrated the effectiveness of our approach.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

HPDC'13, June 17–21, 2013, New York, NY, USA.

Copyright 2013 ACM 978-1-4503-1910-2/13/06 ...$15.00.

Categories and Subject Descriptors

H.2.8 [**Information Systems**]: DATABASE MANAGEMENT—*Database Applications*

General Terms

Performance

Keywords

big data, bitmap indexing, data sampling

1. INTRODUCTION

Many of the 'big-data' challenges today are arising from increasing computing ability, as data collected from simulations has become extremely valuable for a variety of scientific endeavors. With growing computational capabilities of parallel machines, scientific simulations are being performed at finer spatial and temporal scales, leading to a data explosion. As a specific example, the Global Cloud-Resolving Model (GCRM) [24] currently has a grid-cell size of 4 km, and already produces 1 petabyte of data for a 10 day simulation. Future plans include simulations with a grid-cell size of 1 km, which will increase the data generation 64 fold.

Finer granularity of simulation data offers both an opportunity and a challenge. On one hand, it can allow understanding of underlying phenomena and features in a way that would not be possible with coarser granularity. On the other hand, larger datasets are extremely difficult to store, manage, disseminate, analyze, and visualize. Neither the memory capacity of parallel machines, memory access speeds, nor disk bandwidths are increasing at the same rate as the computing power, contributing to the difficulty in storing, managing, and analyzing these datasets. Simulation data is often disseminated widely, through portals like the Earth System Grid (ESG) [6], and downloaded by researchers all over the world. Such dissemination efforts are hampered by dataset size growth, as wide area data transfer bandwidths are growing at a much slower pace. Finally, while visualizing datasets, human perception is inherently limited relative to dataset sizes.

The above trends are leading to the following three problems:

1. Creating subsampled (lower-resolution) datasets from a high resolution simulation dataset, on demand and efficiently, while maintaining the characteristics of the original dataset.

2. Assessing the loss of quality (with respect to the key statistical measures) incurred with a particular level of resolution, on the given dataset, without having to take a pass through the entire high resolution dataset.

3. Providing the above functionality in a flexible system, which can support sampling at the server-side in response to requests from the client-side, and combine sampling with data subsetting.

1.1 Existing Sampling Techniques, Limitations, and Big Data Needs

Though, to the best of our knowledge, no system provides all of the above functionality, sampling itself has been extensively studied. Broadly, different statistical sampling methods [12, 30, 42, 44] have been proposed to find a representative subset of the entire dataset. Some popular techniques include *simple random sampling*, where we select a certain percent of elements randomly out of original dataset, and *stratified random sampling*, where we first divide the dataset into strata and then perform random sampling within each stratum. The latter method maintains certain spatial properties of the original dataset. To compare the accuracy between the sampled dataset and the original dataset, different error metrics [27, 43, 21] have also been used.

However, as we argue below, the existing work does not meet all the requirements, especially in the context of growing dataset sizes and the need for data dissemination and analysis in a distributed environment.

Sampling Accuracy: Two factors are extremely important while creating samples of scientific datasets so as to facilitate accurate analysis. The first is *value distribution*, i.e., the value distribution of the sampled dataset should be as close to the original dataset as possible. The second is *spatial distribution*, i.e., the data accuracy should be maintained not only for the entire dataset but also for various spatial sub-blocks. Most of the sampling methods [33, 43] developed in the context of scientific data management are focused on the second factor, but ignore the first one. On the other hand, value distribution based sampling is well studied and has been proven to be a good method in the database area [18, 35]. These methods, however, are not developed for scientific datasets, and do not even consider spatial distribution. Consideration of both value distribution and spatial locality is necessary for scientific datasets, and unfortunately, none of the existing work has included both.

Error Calculation without High Overheads: After sampling, it is also important to know how accurately the current sample is able to represent the original dataset. Different error metrics, such as mean, variance, histogram[1] and Q-Q plot[2] are used as diagnostics of the accuracy. With increasing dataset sizes and the distributed nature of analysis, there are several challenges in applying these methods. In particular, when the goal is to find the smallest sample that can achieve a satisfactory accuracy, the traditional sampling process involves the following (possibly iterative) process: 1) sample generation, and 2) error metrics calculation. If the error is too high, repeat with a larger sample, starting from step 1. The entire process can be extremely time consuming, especially if one needs to iterate multiple times. In particular, with the current methods, there is no way to know in advance what may be the smallest sample size at which acceptable accuracy levels can be achieved.

Flexible Data Analysis over Any Subset: In many cases, users are only interested in data analysis or visualization over a subset of the data. For example, only certain timestamps may be of interest, and/or only a particular spatial subarea needs to be analyzed. Even if server-side subsetting is available, the resulting dataset size may be very large. Thus, the sampling method should be such that it can be applied to any specified subset. Unfortunately, existing sampling methods cannot support such flexible data subset sampling.

Data Sampling without Data Reorganization: Certain sampling methods, such as KDTree-based stratified sampling[44], have been shown to be effective for scientific datasets. However, before sampling can be performed, data reorganization is necessary. This imposes huge memory and disk I/O costs. Moreover, it is not possible to maintain multiple copies of a massive dataset, and sampling is not the only operation to be performed at server-side. After reorganization, other data features that are necessary for other tasks could be lost. Thus, we need sampling methods which operate while maintaining the data in the original format.

[1] http://en.wikipedia.org/wiki/Histogram

[2] http://en.wikipedia.org/wiki/Q-Q_plot

1.2 Our Contributions

In this paper, we address the above limitations of existing work by developing a novel sampling approach. We observe that to allow subsetting over scientific datasets, data repositories are likely to use an indexing technique [39]. Among these techniques, we see that bitmap indexing can not only effectively support subsetting over scientific datasets, but can also help create samples that preserve both value and spatial distributions over scientific datasets. We have developed algorithms for using bitmap indices to sample datasets. We have also shown how only a small amount of additional metadata stored with bitvectors can help assess loss of accuracy with a particular subsampling level, i.e., we do not need to take a pass over the entire sampled dataset to calculate accuracy based on these metrics. Some of the other properties of this novel approach include: 1) value distribution as well as spatial distribution of the original dataset are preserved, 2) sampling can be flexibly applied to a subset of the dataset, which may be specified using a value-based and/or a dimension-based subsetting predicate, and 3) no data reorganization is needed, once bitmap indices have been generated.

We have extensively evaluated our method with different types of datasets and applications. First, considering two applications - visualization and clustering, we show that server-side sampling can drastically improve the efficiency of remote datasets analysis. Next, we show that our method has much better accuracy than simple random sampling and stratified random sampling methods, and with respect to different metrics, either better or comparable performance to KDTree-based sampling (which requires expensive data reorganization). Next, we show that our error pre-calculation methodology, a unique characteristic of our approach, gives very accurate estimation of error in sampled datasets. We also analyze the sample generation time with our approach, and show that when error calculation time and possibility of resampling to meet desired accuracy is included, our method outperforms other approaches. Finally, we show that we can combine our sampling method with value-based and/or dimension-based subsetting effectively.

2. SYSTEM OVERVIEW

This section gives an overview of the system we have developed to support flexible server-side sampling of large datasets. Technical details of the sampling method will be given in the next Section.

Figure 1 shows a high-level overview of our system. In our previous work we designed a system to support flexible data subsetting (including both value-based and dimension-based predicates) using a standard SQL-like interface [38, 39]. The advantage of this approach is that a simplified *virtual* or high-level view of the dataset is presented to users. Thus, users downloading the data do not need to be familiar with the details of the data format. Instead, they can specify subsetting (and now sampling) requests with the high-level view.

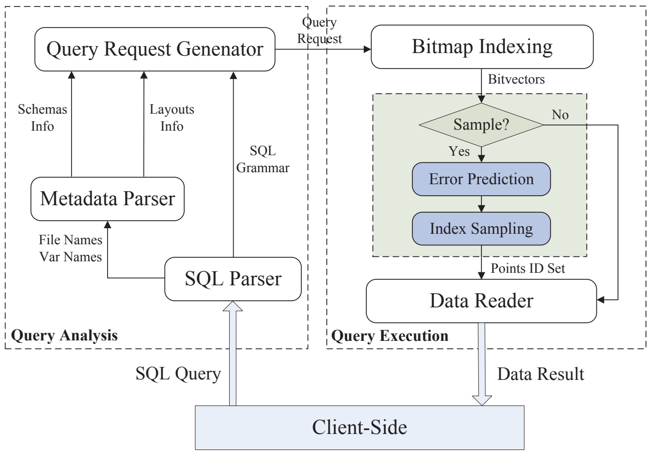


Figure 1: System Architecture

There are two main modules in the system, the *Query Analysis Module* and *Query Execution Module*. The *Query Analysis Module* takes an SQL query and corresponding metadata as input and generates a query request (in a specific format internal to the system) as the output. The *SQL Parser* is responsible for parsing the SQL query and generating a parse tree. We have implemented the parser by making certain modifications to the parser from SQLite[3], which is a lightweight open-source database engine. After the parse tree is generated, the *Metadata Parser* will take the data file names and variable names as input, look up the metadata files, find the data schema and data layout information, and load them into memory.

The second major module, *Query Execution Module*, takes the query request as input, performs data subsetting and sampling based on bitmap indices, and sends the data result back to the client. The *Bitmap Indexing* performs different indexing operations based on the query request and generates a collection of bitvectors which satisfies the current query as output. After that, we check to see if sampling is needed for the current query. If data sampling is not required here, the *Data Reader* will query the data subset based on the indexing information and return the result to the client. Otherwise, the *Data Sampling* sub-module will generate data samples based on bitmap indices.

There are two main components in the *Data Sampling* sub-module: *Error Prediction* and *Index-based Sampling*. Our approach includes a novel error prediction mechanism based on bitmap indices. With the help of this mechanism, we are able to pre-calculate approximation errors before actually sampling the data. Moreover, the error estimation can be performed based on indices instead of scanning through the entire sample. While the latter also reduces the error calculation time, our pre-calculation method allows a user to choose a sampling level which maintains a desired level of accuracy. Moreover, this alleviates the need for extracting a sample, calculating the error, and then resampling (likely with a different subsampling level), which can be very expensive in practice.

After error estimation, the *Index-based Sampling* component performs data sampling directly over bitmap indices and generates a set of data record identifiers as the result. Then the *Data Reader* will take the data record identifiers as the input, extract the data records, and return the results.

Besides error pre-calculation, which can improve the overall sampling efficiency significantly, and the overall effectiveness of our method, there are at least three other advantages for our system.

[3]http://www.sqlite.org

Small Preprocessing Costs: If the data repository already uses bitmap indices, or will like to use bitmap indices to efficiently obtain subsets of the original dataset, we can directly apply our sampling method without any preprocessing. For those applications without bitmap indexing support, the computational complexity of index generation is only $O(n\log(m))$ where n is the number of total elements and m is the number of bitvectors [47]. With the help of binning, m can be much smaller than n, so $\log(m)$ can be considered a constant number. Thus, our method is much faster compared with sampling methods with $O(n\log(n))$ preprocessing time, such as the KDTree-based method [44]. Another advantage of our method is that we do not need any modifications or reorganization of the original dataset. All sampling operations are performed using data in the original format and the bitmap indices.

Tradeoff between Accuracy and Sampling/Memory Costs: The bitmap indexing allows flexible multi-level indices over a given dataset. The *low-level* bitmap indices are able to reflect data features at a fine granularity, whereas the high-level indices improve the efficiency by binning a group of low-level bitmap indices together. By choosing to perform sampling using high-level or low-level bins, and even choosing the bin size at one or both levels, one can achieve the desired tradeoff between accuracy of sampling and time/memory costs of the sampling process.

Combining Sampling and Subsetting: Since our system is built on top of a data subsetting system, users can combine sampling with subsetting. Moreover, such queries can be executed efficiently because of the properties of bitmap indices.

3. SAMPLING USING BITMAP INDICES

This section first provides background on bitmap indexing and then introduces our data sampling method using bitmap indices. We also describe three enhancements of our sampling method, which are error prediction, sampling over a data subset, and sampling to support multi-attributes data analysis.

3.1 Background - Bitmap Indexing

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 [32, 47]. In particular, recent work has shown that bitmap indexing can help support efficient querying of scientific datasets stored in native formats [11, 39].

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 [46]. 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,

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

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) [4] and Word-Aligned Hybrid (WAH) [45] 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.

3.2 Stratified Random Sampling over Bitvectors

Consider data storage in a large-scale scientific repository. If we are using bitvectors to be able to retrieve subsets of the original dataset [11, 39], the question we want to focus on is "can the same bitvector be used to obtain accurate and representative samples, while also assessing the loss of accuracy with a particular sampling level". It turns out that bitvectors can not only be used in this fashion, but they also provide several advantages over existing and popularly used sampling techniques.

We now describe the bitvector based sampling method we have developed. The basic idea in our method is to perform *random stratified sampling* over each bitvector, which corresponds to a particular value or, more likely, a bin of values. Specifically, we extract the same percent of samples out of each bitvector. By sampling over bins with equal probability, we are able to keep value distribution in the sampled dataset close to that of the original dataset. In fact, as we will show below, this approach preserves *entropy* of the original dataset, a highly desired property of samples in many applications.

Within each bitvector, we first divide the bitvector into *sectors* of a certain size, and choose the same percent of samples out of each sector. This way, we can also preserve the value distribution within each spatial region. Furthermore, when multi-level bitvectors are created (such as the example earlier in Figure 2) this method can be applied to either the low-level or the high-level index. This choice allows a tradeoff between efficiency and accuracy.

We now explain the steps of our method in more detail, using an example in Figure 3. There are three main steps:

Building bitmap indices: In this example, the small dataset contains 32 elements, so each bitvector has 32 bits. The number of distinct values is 4. The low-level bitmap indices contain 4 bitvectors: $e_0(=1)$, $e_1(=2)$, $e_2(=3)$, $e_3(=4)$, and the high-level bitmap indices include 2 bitvectors: $i_0([1,2])$, $i_1([3,4])$. In this simple example, all values are integers, though as we mentioned in Section 3.1, bitmap indices can be (and have been) used for floating-point values by generating bins with value ranges.

Dividing bitvectors into sectors: In order to preserve distribution of values in each spatial region, bitmap indices should be logically

Sampling over Low Level Indices:

e_0	=1	00010100 | 00110001 | 01010110 | 00010001
se_0	=1	00000100 | 00100001 | 00000110 | 00010000
e_1	=2	00100000 | 10000010 | 00100000 | 10000010
se_1	=2	00000000 | 10000000 | 00100000 | 00000010
e_2	=3	10001001 | 00000100 | 10001001 | 00100100
se_2	=3	00001000 | 00000000 | 10000001 | 00100000
e_3	=4	01000010 | 01001000 | 00000000 | 01001000
se_3	=4	01000000 | 00001000 | 00000000 | 00001000

Sampling over High Level Indices:

i_0	[1, 2]	00110100 | 10110011 | 01110110 | 10010011
si_0	[1, 2]	00100100 | 10000010 | 00010100 | 10010000
i_1	[3, 4]	11001011 | 01001100 | 10001001 | 01101100
si_1	[3, 4]	10001001 | 00001000 | 10001000 | 01000100

Figure 3: Our Proposed Sampling Method: Stratified Random Sampling over Bitmap Indices

divided into spatial sectors. In the figure, we can see that for both the low-level and the high-level bitmap indices, every bitvector is divided into 4 sectors, and there are 8 bits within each sector.

Random sampling over each sector: After creating sectors, random sampling can be performed within each sector, and for each bitvector, to generate data samples. Within each bitvector, random sampling is only applied to 1-bits. To preserve value distribution within each region, we need to make sure sample percentages over each sector are the same. One advantage of using bitmap indexing is that its implementations help us locate all 1-bits efficiently. In Figure 3, we are generating 50% samples out of the original dataset. We can see that se_0, se_1, se_2, se_3 are identifiers of data records that are in the sample generated using the low-level bitvectors, whereas si_0, si_1 are the data records for the sample using the high-level bitvectors. For both low-level and high-level bitmap indices, within each sector, only half of the 1-bits are picked. For example, after sampling, the number of 1-bits in the sample bitvector se_0 is 6, which is only half of that in original bitvector e_0.

From the figure, we can also see that although low-level bitmap indices have more bitvectors, each bitvector has fewer 1-bits. On the other hand, the number of bitvectors in the high-level bitmap indices is smaller, but more 1-bits exist in each bitvector. Hence, both methods generate sampled datasets of the same size. Low-level bitmap indexing is able to achieve better accuracy because it reflects the value distribution at a finer granularity. However, it also has an additional time cost, because of higher indices loading time and bitvector striding time.

Finally, we point out the property of this method with respect to preserving entropy. Information theory and *entropy* have been extensively used while sampling data (or even selecting angles, streamlines, or other features) in graphics and visualization, as also summarized by Xu *et al.* [48].

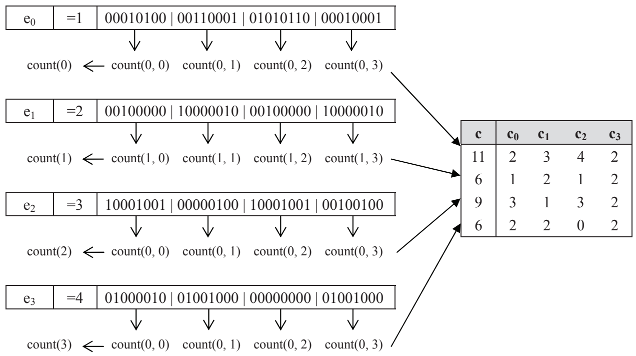


Figure 4: Metadata Generation for Error Prediction

Formally, if X is a random variable with a series of possible outcomes x, where $x \in \{x_1, x_2, \ldots, x_n\}$, and if the probability for the random variable to have the outcome x_i is $p(x_i)$, then Shannon's entropy is defined as

$$H(X) = \sum_i p(x_i) \times \log(1/p(x_i)).$$

Assuming no binning is performed, and sector sizes are large enough that precisely the same fraction of values can be chosen, we can see that the sampled dataset using bitvectors will have the same distribution of values, or the same entropy.

3.3 Error Prediction

After sampling, it is also important to know how accurate the sampled dataset is compared with the original dataset. Traditional sampling methods can only calculate error metrics after samples are generated, and if the error is too high, the entire sampling process has to be repeated with another sample percentage. As we will show now, with bitvectors we are able to pre-calculate error metrics based on bins. Thus, we can perform error predictions analysis to find a sample percentage which will give desired accuracy levels, and then can perform data sampling only once. This is a significant advantage, since the error calculation method only takes at most $O(m)$ time, where m is the number of bitvectors. In comparison, sample generation normally takes $O(n)$ time, where n is the number of data records in the original dataset, and $n >> m$.

As we stated earlier, while evaluating quality of a sampled dataset, different error metrics, like mean, variance, histogram and Q-Q plot are used. In particular, for our discussion we consider error metrics of two types: 1) mean, variance, histogram, and Q-Q plot for each variable, and 2) mean and variance for each sector.

We need to calculate and store some additional information during bitmap index generation. Figure 4 shows the metadata generation over bitmap indices. For dataset or variable level error calculation, the only additional information we need is the total number of 1-bits within each bitvector. From the figure, we can see that $count(0)$, $count(1)$, $count(2)$ and $count(3)$ record the total number of 1-bits for each bitvector. The results are stored in the first column of the 2-dimensional count matrix c. The metadata we need for sector-level mean and variance calculation is the number of 1-bits within each sector. From the figure, we can see that for bitvector $e0(=1)$, $count(0,0)$, $count(0,1)$, $count(0,2)$ and $count(0,3)$ record the number of 1-bits within each sector. The result is stored in columns $c0$, $c1$, $c2$ and $c3$ of the count matrix.

Now we elaborate on calculation of specific metrics. Our approach can also be referred to as *error pre-calculation*, which is in contrast to *error post-calculation* normally done with the traditional sampling methods.

Mean, Variance, Sector Means, and Sector Variances: We now show how to pre-calculate *mean* and *variance* of the sampled dataset based on bins and the count matrix. The input is the representative value ($value$) of each bin, which we determined at the time of index generation, and the total number of elements ($count$) within each bin, which we can find from the count matrix. Besides that, each time we also set a sample percentage to decide the size of the sample result, denoted as $SamplePercent$. Equation 1 computes the number of samples selected from each bitvector($scount_i$) based on $count_i$ and $SamplePercent$:

$$scount_i = count_i \times SamplePercent. \tag{1}$$

Our method fetches the same percent samples out of each bitvector, which is equal to $SamplePercent$. Hence, by multiplying $count_i$ with $SamplePercent$, we are able to compute the approximate number of samples within each bitvector. Now, Equation 2 calculates the mean value of the sampled dataset:

$$Mean = \frac{\sum_{i=1}^{m}(scount_i \times value_i)}{\sum_{i=1}^{m}(scount_i)}. \tag{2}$$

Within each bitvector, we know both the representative $value_i$ and sample size $scount_i$. By multiplying these two factors together, we can get the sum value of samples in the current bitvector. Based on that, we can calculate the total value by adding the sum value of each bitvector together. We are also able to count the total number of sample elements by adding $scount_i$ of each bitvector together. Based on the sum value and total sample elements count, we can get the *mean* value.

Equation 3 calculates the *variance* of the sampled dataset. We first compute the value differences within each bitvector based on $mean$ and $value_i$, then add all value differences together and finally divide by the total number of sample elements:

$$Variance = \frac{\sum_{i=1}^{m}\left(scount_i \times (Mean - value_i)^2\right)}{\sum_{i=1}^{m}(scount_i)}. \tag{3}$$

The method of calculating *sector means* and *sector variances* is similar. We simply need to apply the Equations 2 and 3 for each sector.

We can see that our approach, error pre-calculation, can calculate *mean* and *variance* within $O(m)$ where m is the total number of bitvectors. Note that in contrast, the error post-calculation method will have to scan the entire sampled dataset twice to compute the mean and the variance. The time complexity is $O(s)$, where s is the sample size.

Histogram: The input is still $value$, $count$ and $SamplePercent$. Based on Equation 1, we can obtain the number of sampled elements for each bitvector ($scount_i$). Now,

$$Prob_i = \frac{scount_i}{\sum_{i=1}^{m}(scount_i)}. \tag{4}$$

Equation 4 calculates each value $Prob_i$ in the histogram by simply dividing the sample size of each bitvector $scount_i$ by the total sample size. This way, we obtain the element probability of each bitvector. By calculating probabilities over all bitvectors, we are able to generate a histogram.

This method can compute the *histogram* within $O(m)$, where m is the number of bitvectors. In comparison, error post-calculation has to first perform a *Radix Sort*[4] over the entire sampled dataset. After that, it needs to count the number of elements within each bucket and then divide this number by the total sample size. The time complexity is $O(s)$ where s is the sample size.

[4]http://en.wikipedia.org/wiki/Radix_sort

Q-Q Plot: We first recap the definition of a Q-Q plot. Viewing the original dataset and the sampled dataset as two distributions, we compare them by plotting their quantiles against each other.

Algorithm 1 shows how to calculate a Q-Q plot using bitvectors. The input is s , which indicates the total number of sample elements; m, the total number of bitvectors; q, the total number of quantiles; $count$, the number of elements within each bitvector; and $value$, the *representative value* of each bitvector (calculation described below). In line 1, we define a variable $curCount$ to record the total number of elements that are smaller than the value of the current bitvector. The variable pos indicates each quantile position identifier in the sampled dataset. It can be computed based on total sample size(s), multiplying it with the quantile percentage, as shown in line 8. Lines 3 to 12 compute the quantile value based on each quantile position. We iterate from the bitvector with the smallest value to the bitvector with the largest value. If the current quantile position pos is larger than $curCount$, we update the $curCount$ and go to the next bitvector, as shown in line 4 and line 10. If pos becomes smaller than $curCount$, it means the current quantile is located within the current bitvector. Then we can record the representative value of the current bitvector as the quantile value and go to the next quantile, as captured by lines 5 through 8. We keep performing this calculation until we find the value of all the desired quantile positions.

Algorithm 1: Compute_QQPlot(s, m, q, $count$, $value$)

```
1: curCount ← 0, pos ← 0
2: i ← 0, j ← 0
3: while i < m&&j < q do
4:    curCount ← curCount + count_i
5:    if curCount > pos then
6:       QQPlotArray_j ← value_i
7:       j ← j + 1
8:       pos ← s * j/100
9:    else
10:      i ← i + 1
11:   end if
12: end while
```

Our method is able to calculate the Q-Q plot with $O(q)$ in the best case and $O(q+m)$ in the worst case, where q is the total number of selected quantiles. In comparison, the error post-calculation method has to first perform a quick sort over the entire sampled dataset to calculate the Q-Q plot. After that, certain quantiles need to be selected out of the sorted dataset as Q-Q plot values. For example, we can fetch the data elements located at 1%, 2%, ... ,100% positions out of the sorted sample dataset as the result. The time complexity is $O(s \times \log(s))$ where s is the sample size.

Now, we describe how we calculate $value$, the representative value of a bitvector, when we have multi-level bitmap indices. For low-level bitmap indices, we can simply use the mean or the median value as the representative value of each bin. For high-level bitmap indices, each bitvector indicates a relatively larger value range. In our work, we use three indicators to predict errors for high-level bitmap indices. In high-level bitmap indices, each bin indicates a value range which has both a lower-bound and an upper-bound. By using lower-bound and upper-bound values during the error prediction process, we are able to calculate a boundary on the actual error metric results. Besides, each high-level bin is built by combining a group of low-level bins together. Hence, we are able to calculate the value distribution of each high-level bin by looking at corresponding low-level bins and finding an estimated value to represent each high-level bin. This way, we are able to find the actual error boundaries and also generate a relatively accurate error prediction. In some cases, when the data range of the dataset is large, the bin size of low-level bitmap indices can be big. We can also apply this three indicators method to low-level bitmap indices.

3.4 Sampling Only a Subset of Data

When a data repository is disseminating data, a particular user might only be interested in a certain subset of data, based on spatio-temporal ranges (*dimension subsetting*) and/or specific values for attributes (*value-based subsetting*). However, as the dataset size for the subset may still be too large, sampling may still be needed.

Traditional sampling methods cannot efficiently support data sampling over a user-specified subset of data that includes value-based subsetting. For example, simple random sampling, stratified random sampling and KDTree stratified random sampling methods can all handle dimension-based subsetting, but when value-based subsetting is involved, they have to first generate data samples over the entire dataset and then perform post-filtering, which is clearly not efficient.

Suppose we need to sample datasets at a certain level, in conjunction with a subsetting condition, which includes both dimension-based and value-based subsetting conditions. We will proceed as follows. We first focus on the value subsetting conditions and search the (possibly) multi-level bitmap indices to find corresponding bitvectors. Only these bitvectors need to be loaded. Next we perform dimension subsetting over the retrieved bitvectors. Finally, we apply the stratified sampling only over this bitset.

3.5 Data Subsetting and Sampling over Multiple Attributes

In a typical scientific dataset, certain attributes can be *stand-alone*, i.e., can be analyzed separately. On the other hand, certain attributes can be closely connected with each other, and it is better to study them together. Suppose we consider the output from the cosmology data described in Section 4 below. Each record in the dataset corresponds to one particle and includes multiple attributes. For example, the attribute *mass* indicates the field value related to the current particle, and *VX*, *VY*, *VZ* indicate the particle velocity in each of the three spatial dimensions. *mass* can be analyzed separately, as it does not have a strong connection with the other attributes. For *VX*, *VY*, *VZ*, however, scientists prefer to analyze them together to find the relationships among them.

The techniques we have described so far build indices over each attribute separately, which does not fit the second scenario very well. We now describe an extension to support sampling to ensure a preserved distribution over multiple attributes.

Suppose we need to sample with respect to two attributes, X and Y. The entire process can be divided into 3 steps: (1) Divide the value range of each attribute into *one-attribute* bins, say, (X_1, X_2, ... ,X_{m1}) and (Y_1, Y_2, ... ,Y_{m2}). (2) Form *multiple attributes bins* (or *mbins*) (X_1, Y_1), (X_1, Y_2), ... ,(X_{m1}, Y_{m2}) based on the one-attribute bins generated in the previous step. For each *mbin*, generate a bitvector and initially set all bits to 0. (3) Scan through the dataset. For each record, find its X and Y value, classify it into the corresponding *mbin* and set the corresponding bit to 1. Repeat this process until all records are mapped to related *mbins*.

4. EXPERIMENTAL RESULTS

In this section, we report results from a number of experiments conducted to evaluate our sampling approach. We designed experiments with the following goals: (1) To show how data sampling is able to improve data analysis efficiency in a distributed environment (where data source and resources for data analysis are geographically separated), (2) To examine the accuracy of our bitmap indices sampling method and compare it with a number of other sampling methods, (3) To evaluate the accuracy of error pre-calculation, by comparing predicted errors with the actual errors,

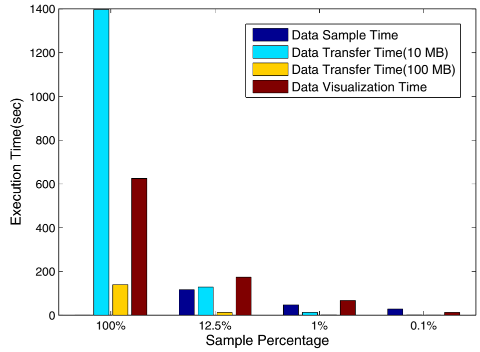


Figure 5: Visualizing a Remote Dataset: Execution Time with and without Sampling

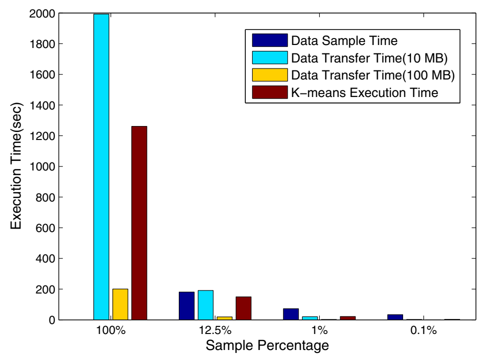


Figure 6: Clustering a Remote Dataset: Execution Time with and without sampling

(4) To compare the efficiency of our method against other sampling methods, in particular in view of error pre-calculation, and (5) To show how sampling over data subsets improves the efficiency.

We used two different scientific datasets. The *ocean* dataset is generated by the Parallel Ocean Program (POP) [22], which is an ocean circulation model. The execution 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, and each variable is modeled with three dimensions: longitude, latitude, and depth. The data is stored in the NetCDF format. The *cosmology* dataset is generated by the Road-Runner Universe MC^3, which is a large N-body cosmology simulation of dark matter physics. An MC^3 time step of 4000^3 (64 billion) particles with 36 bytes per particle takes 2.3 TB per time-slice. The particles generated per time-slice are split into a collection of data files based on the spatial information. Each particle within the file corresponds to one record, which is formed by 8 attributes (X, Y, Z, VX, VY, VZ, MASS, TAG). The data is stored in binary format.

In our experiments, the data repository and the server-side data sampling are on the Darwin Cluster at Los Alamos National Laboratory. Darwin consists of 120 compute nodes with 48 core(12-core by 4 socket) 2GHz AMD Opteron 6168 and 64 GB memory. The client-side data analysis is performed on one compute node which has 8 cores Intel(R) Xeon(R) CPU 2.53GHz and 32 GB memory.

4.1 Improving Efficiency of Distributed Data Analysis with Sampling

In this experiment, we consider the following scenario. The entire dataset is located on a remote server, and any analysis must be done after the data is downloaded to the client-side. We consider two distinct applications: data visualization and data mining. In the data visualization scenario, we visualize the sampled dataset using Paraview [2], a widely used data analysis and visualization application. In the data mining scenario, we take data samples as input and perform K-means clustering using MATE [20], a map-reduce like system. With these two applications, we compare the efficiency of data analysis (including data downloading time), when using the original dataset against the cases where different subsampling levels are used. In particular, we divide the data processing time into three parts: 1) Server-side data sampling time, 2) Data transfer time between the server and the client, and 3) Client-side data analysis time. The second factor above varies with the wide-area data transfer bandwidths one might have. For our experiments, we used two different networks, one with 10 MB/s bandwidth and the other with 100 MB/s bandwidth.

Figure 5 compares the efficiency of the data visualization using different subsampling levels: 100%, which means that we are using the original dataset without sampling, 12.5%, 1%, and 0.1%. The dataset without sampling is 11.2 GB in size and is from the POP application. From the figure, we can see that although our method incurs extra sampling costs compared to the case when the original dataset is analyzed, both the data transfer and analysis time is much lower, and more than compensates for the sampling time. Specifically, we find that compared to visualization over the original dataset, if network bandwidth is 10 MB/s, the speedup with 12.5% sampling rate, 1% sampling rate, and 0.1% sampling rate is 4.82, 15.91, and 47.59, respectively. If network bandwidth is 100 MB/s, the corresponding speedups are 2.61, 6.72, and 19.02, respectively. Another consideration with sampling is the accuracy of the analysis, which we will focus on in the next subsection.

Figure 6 compares the efficiency of K-means clustering (data mining) execution, using the original dataset and the three sampling levels (12.5%, 1%, and 0.1%). The dataset is from cosmology, and is 16 GB in size. The number of K-means cluster centers is 10 and the number of iterations is 50. The number of threads is 4. From the figure, we can see that, similar to data visualization, with the help of sampling, the speedup with 10 MB/s network bandwidth ranges from 5.25 to 84.24, and the speedup with 100 MB/s network bandwidth ranges from 3.26 to 39.8. Again, accuracy is another consideration, which we will analyze next.

4.2 Accuracy Comparison with Different Sampling Methods

As we stated above, besides efficiency, accuracy is a very important consideration for a sampling method. Using visualization and clustering as representative data analysis applications, we not only evaluate the absolute accuracy of our method, but also compare the accuracy against three other methods.

The sampling methods we compare against are as follows. Simple random sampling involves randomly selecting a data subset out of the original dataset without focusing on any features. Stratified random sampling [12] performs random sampling within each *stratum*. Normally, the way these strata are formed can preserve spatial distribution of samples, but not the value distribution. KDTree-based sampling [44] has been proven to be a good method for visualization, and has also been applied to the cosmology dataset. It divides data into strata by building a k-dimensional tree over the dataset. The tree construction method is primarily based on spatial dimension(s) but can also consider data values as one dimension. Random sampling is performed within each stratum to generate a data sample. Because both data values and spatial distribution are considered in forming the strata, KDTree-based sampling has led to better accuracy than stratified random sampling.

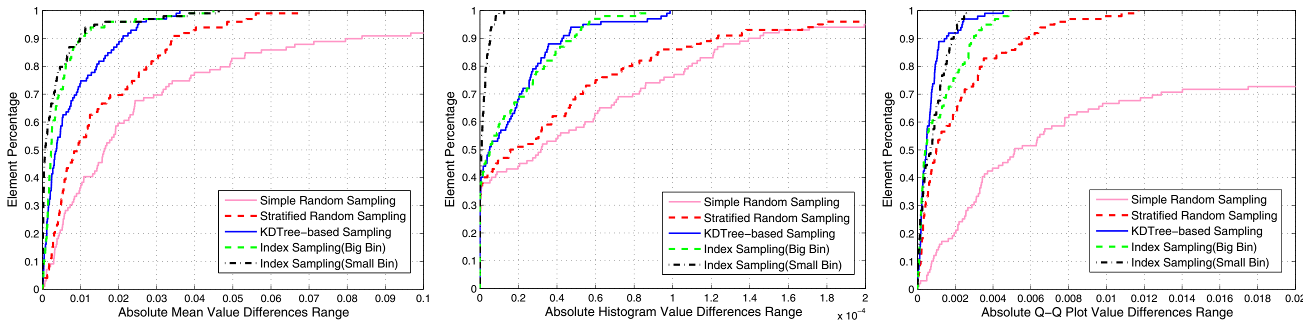


Figure 7: Error (Means, Histogram, and Q-Q Plot) Comparison Using Cumulative Frequency Plots: TEMP from POP Dataset

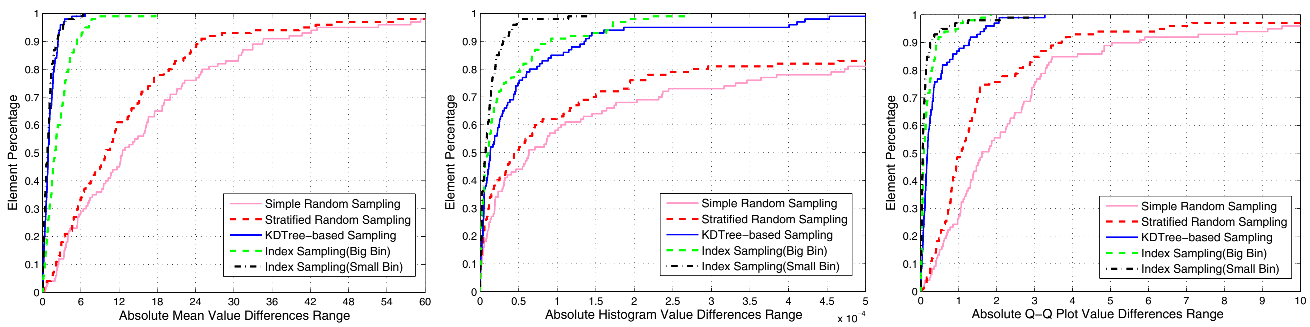


Figure 8: Error (Means, Histogram, and Q-Q Plot) Comparison Using Cumulative Frequency Plots: VX from Cosmology Dataset

In our method, which we will refer to as *index sampling*, we chose two bitmap indexing levels. The method we will denote as *small bin* corresponds to the use of low-level bitmap indices, which indicates fine-grained value distribution. The method we will denote as *big bin* corresponds to the use of high-level bitmap indices. Here, we groups 10 small bins into a big bin, and thus, value distributions are preserved only at a coarser level. The datasets and the variables used here are the same as the previous experiment: *TEMP* from the POP dataset and (*VX*, *VY*, *VZ*) from the cosmology dataset. The sample percentage is 0.1% of the original dataset.

It turns out that the appropriate error metrics for visualization and clustering are very distinct. Now we discuss the accuracy of the two applications separately.

4.2.1 Accuracy for Visualization

Characterizing the impact of sampling on visualization is hard, since human perception plays a role in how a dataset is viewed. Based on the existing literature from visualization [44], we used the following metrics: means of the value over 200 separate sectors, histogram using 200 value intervals, and Q-Q plot with 200 quantiles. To make the results more obvious, we calculated the sector means, histogram, and Q-Q plot value of both the original dataset and each sample dataset, and computed the absolute value differences between the original dataset and the sample dataset. To represent these charts, we use a *Cumulative Frequency Plot*(CFP). In our plots (Figure 7 for example), a point (x, y) indicates that the fraction y of all calculated absolute value differences are less than x. Since the error metric 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. For the bitmap index sampling method, the total number of small bins of *TEMP* is 442, and the total number of small bins of *VX* is 670. Each 10 small bins are grouped into a big bin.

The left subfigures of Figures 7 and 8 show the absolute value differences of sector means using the five sampling methods (including two versions of our approach). The simple random sampling shows the worst accuracy. The stratified random sampling, which considers spatial distribution, achieves better accuracy than simple random sampling. However, as it does not consider value distribution, the results are still worse than KDTree-based sampling and index sampling. If we compare KDTree-based sampling with index sampling, we can see that for POP data, index sampling(both small bin and big bin) achieves better accuracy than KDTree-based sampling. For cosmology data, KDTree-based sampling shows better accuracy than index sampling(big bin). However, index sampling(small bin) method still achieves the best accuracy.

The middle subfigures of Figures 7 and 8 show the absolute value differences for histogram entries, comparing the five sampling methods. KDTree-based sampling considers value distribution by treating variable value as one dimension during the KDTree sorting process. This method is more focused on spatial partitions and only considers value distribution at a very coarse level. Thus, as we can also see from the figures, for the cosmology dataset, the histogram results with KDTree-based sampling are not as good as our method. For the POP dataset, KDTree-based sampling and index sampling with big bin achieve a similar accuracy. Index sampling with small bin achieves a better accuracy than all the other methods.

The right subfigures of Figures 7 and 8 show the absolute value differences of Q-Q plot values among the five sampling methods. If we compare KDTree-based sampling with index sampling, we can see that for the POP dataset, KDTree-based sampling achieves the

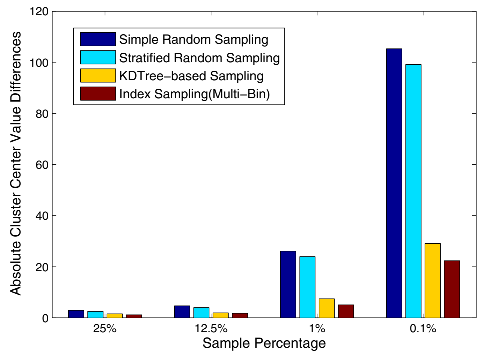


Figure 9: K-means: Accuracy Differences with Different Sampling Levels and Sampling Methods

best accuracy, but for the cosmology dataset, index sampling(both small bin and big bin) shows better accuracy. On the whole, the Q-Q plot value differences between KDTree-based sampling and index sampling are small.

4.2.2 Accuracy for Clustering

The error metric here is the difference between cluster centers, using the original and the sampled dataset. Specifically, we first calculate cluster center values for the original dataset, then calculate cluster center values for the sampled dataset, and finally compute the Euclidean distance between cluster centers in the the original dataset and the sampled dataset. The dataset we used here is the cosmology data and the indices are built over the three attributes *VX*, *VY* and *VZ*, i.e., the multiple attribute sampling method summarized in Section 3.5 is used here. The total number of multiple bins for *VX*, *VY*, *VZ* is 2000.

Figure 9 shows the accuracy using four sampling methods. The X axis shows different sampling percentages (25%, 12.5%, 1%, 0.1%), and Y axis shows the average cluster center value differences. KDTree-based sampling considers sorting based on spatial information first and then values. In this case, this method sorts the data based on *X*, *Y*, *Z* and then *VX*, *VY* and *VZ*. It achieves better accuracy compared with simple random sampling and stratified random sampling. Indices sampling, which considers binning over *VX*, *VY* and *VZ* first and then spatial locality, achieves better accuracy than all the other methods. As sampling percentage decreases, the advantage of our method becomes even more prominent.

To summarize our discussion, we can observe the following. Traditional methods from statistics, i.e., simple random and stratified random sampling, cannot get accurate samples as they are not considering enough features of the data. KDTree-based sampling, which is more focused on spatial locality, achieves good accuracy on sector means and Q-Q plots. However, the histogram result is not as good as for bitmap index sampling. Our method, which considers the value distribution first and then spatial locality, is able to generate a better histogram, while at the same time achieving good accuracy for sector means and Q-Q plots compared to KDTree-based sampling. It also achieves a better result than all the other methods when multiple attributes need to be considered while sampling. Furthermore, our method allows flexibility in choosing bin levels, and thus, users can adjust the bin size and level to get the desired tradeoff between accuracy and efficiency. Finally, as we will elaborate later, another advantage of our method lies in its ability to pre-calculate error levels.

4.3 Error Prediction Accuracy

As we have stated throughout, an important and distinct feature of our approach is the ability to pre-calculate error levels. However, we need to verify if the predicted error results are close to the actual error results. We now describe results from an experiment designed for this purpose using the POP dataset. The sampling percentage is 0.1%.

In this experiment, we first calculate predicted error metrics with the methods described earlier in Section 3.3, then compute the actual error metrics by scanning over the entire sample dataset and compare the two sets of results. Figure 10 compares the predicted and actual errors for sector mean values, histogram and Q-Q plots, using the index sampling(small bin) method. The two sets of lines are either always or almost always identical, which shows that for index sampling(small bin) method, our error pre-calculation is able to accurately reflect actual error results.

Figure 11 compares the predicted and actual errors for sector mean values, histogram, and Q-Q plots, now using the index sampling(big bin) method. Here, we use the mean value as the representative value for each big bin. In the left figure (means), if we compare the predicted errors with the actual errors, we can see that there are only small value differences between the 60th sector and the 85th sector. In most cases, these two lines are identical. In the middle figure (histogram), we can see that there is some variation. This is because the index sampling with big bin method represents value distributions at a relatively coarse granularity. Each big bin can only be classified into one value interval in a histogram, but each bin contains a value range and some values may belong to the neighboring intervals. In the right figure (Q-Q Plot), again the differences are very small.

4.4 Efficiency Comparison with Different Sampling Methods

Earlier we have shown the benefits of sampling for improving the execution time when datasets are remote. However, so far we have not compared efficiency of our method against other methods. We now report such a comparison. Since a key feature of our approach is error pre-calculation, we focus on a scenario where the samples must be generated so as to meet certain accuracy requirements. Thus, the total sampling time can be divided into two components: *sample generation time* and *error calculation time*. Moreover, with other methods, one may need to sample multiple times to obtain the right accuracy levels. The variable we use here is *TEMP* from the POP simulation, and the data size is 1.4 GB.

Figure 12(a) compares the sample generation time among the five sampling methods. The X axis shows different sampling percentages, (3.13%, 6.25%, 12.5%, 25%), and the Y axis shows the execution time in seconds. We can see that simple random sampling takes the least time, which is not surprising. Stratified random sampling and KDTree-based sampling have similar sample generation time, each being somewhat slower than simple random sampling because of the time needed for generating strata. Another difference between stratified random sampling and KDTree-based sampling is that the latter requires $n \log(n)$ preprocessing time, which is not included here. In our method, the random sampling must be applied to each bitvector, which leads to higher time cost than the other three methods. This time depends upon the number of bins used. We can see that with the big bin method, which has one-tenth the number of bins compared to the small bin method, the time cost is only marginally higher than other methods. However, the index sampling(small bin) method has 1.19 to 3.98 times slowdown over KDTree-based sampling.

Figure 12(b) compares the error calculation time among the five sampling methods. With simple random sampling, stratified random sampling, and KDTree-based methods, we have to take a pass over the entire sampled dataset to perform error calculations. This is not only a high cost, but one that also increases with the size of the sampled dataset. In comparison, our method is able to pre-

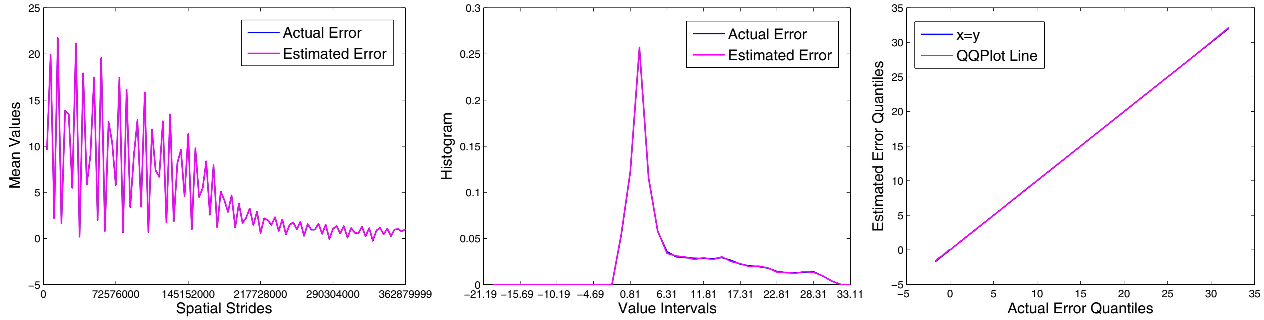


Figure 10: Predicted and Actual Errors (Means, Histogram, and Q-QPlot): Small Bin Method

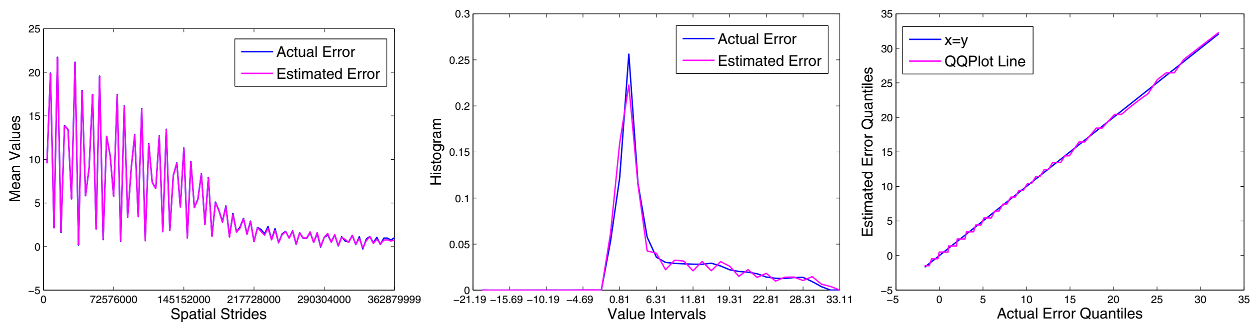
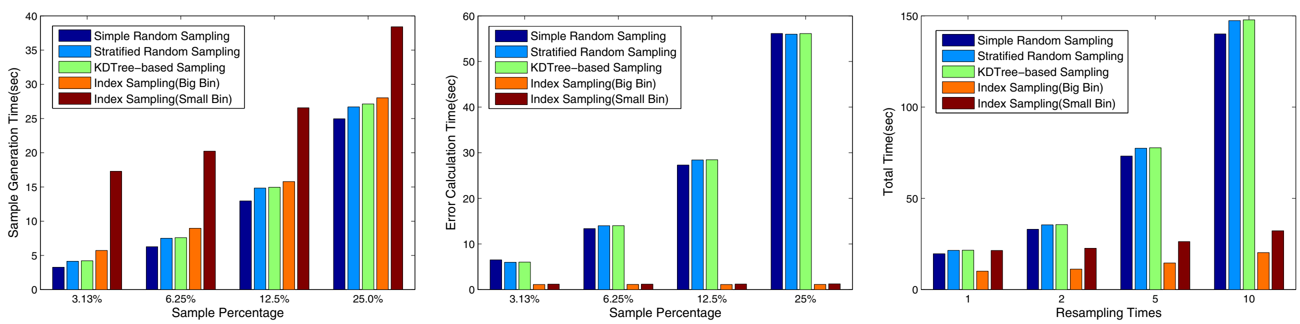


Figure 11: Predicted and Actual Errors (Means, Histogram, and Q-QPlot): Big Bin Method



(a) Sample Generation Time (b) Error Calculation Time (c) Total Time

Figure 12: Time Cost Comparison across Sampling Methods

calculate error metrics based on bins(quite accurately, as we established earlier) before sampling. And the cost of performing the pre-calculation is not related to the sample size. From the figure, we can see that our method achieves at least 28x speedup compared with the other three methods while creating a 25% sample of the dataset. Note that these results are for a 1.4 GB dataset, and the advantage of our method will increase for larger sized datasets.

Figure 12(c) compares the overall efficiency among the five sampling methods. The X axis shows the resampling times, and the Y axis shows the total time cost in seconds. The sampling percentage is 6.25%. Because the first three methods cannot support error prediction, the sample generation and error calculation process may have to be repeated multiple times until a satisfactory accuracy level is found. However, using index sampling, we can perform multiple error pre-calculations first (with different sampling levels) and then need only one round of sample generation. If we look at the first set of bars which correspond to the case where we sample only once, we can see that index sampling(small bin) method has a similar total cost compared with the other three methods, whereas the index sampling(big bin) method is significantly faster. However, if the sampling process needs to be repeated, both big bin and small bin methods are much faster than any of the other methods.

4.5 Data Sampling over Data Subsets

Another advantage of bitmap indexing is that it supports efficient subsetting over subsets of the original dataset, where these subsets

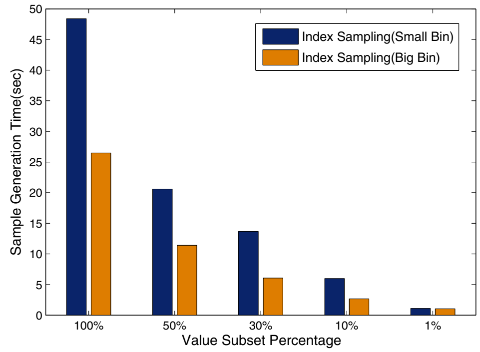


Figure 13: Sampling over Value Subsets

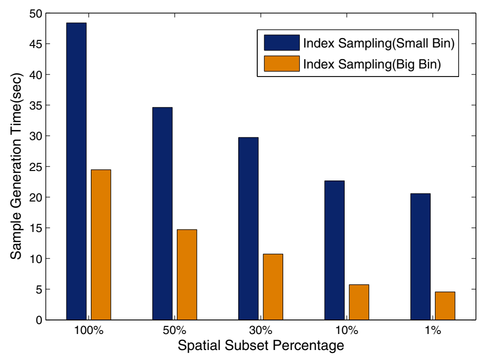


Figure 14: Sampling over Spatial Subsets

may involve spatial (dimension-based) and/or value-based conditions. In this subsection, we show how our method is effective, i.e. data sampling efficiency improves if sampling is performed over a subset of values or spaces. Here we discuss value subsetting and spatial subsetting separately, although our method is able to support a combination of the two.

Figure 13 shows the time incurred while sampling over different value-based subsets. The X axis shows the subsetting percentage, i.e. the fraction of the original dataset that meet the conditional predicate. The Y axis shows the sampling time, which includes both the index loading time and the sample generation time. The sampling rate is 25% in all cases, i.e. 25% of the data records that meet the conditional predicate are returned. From the figure, we can see that for both the small bin and big bin methods, the efficiency improves as the subsetting percentage decreases. Smaller value-based subset implies not only smaller index loading time but also smaller sample generation time. Take the index sampling(small bin) method for example, sampling over 10% of the data takes 6.95 times less time than sampling over 100% of the data.

Figure 14 shows the time cost of sampling with different spatial subsets. The X axis shows the spatial subsetting percentage and the Y axis shows the indices sampling time. The sampling percentage is still 25%. From the figure, we can see that the time cost decreases as the spatial subsetting percentage decreases, though the improvement is not as obvious as in the case of value subsetting. This is because for spatial subsetting, all indices still have to be loaded, so the only speedups are on the sample generation time.

5. RELATED WORK

Sampling of datasets has been widely studied, including work specific to scientific datasets and/or visualization.

Traditional statistical sampling methods [12], including simple random sampling and stratified random sampling, have been used often. We have performed a detailed comparison against these methods and demonstrated how our approach is more effective. KDTree-based sampling [44] uses a KDTree to divide data into sub-blocks and performs random sampling within each block. It needs to reorganize the entire dataset, with a time complexity of $O(n\log(n))$. We have also compared our method with this method, and shown that our approach outperforms this method in several aspects, and is comparable in other ways. The Z-curve order sampling method [33] involves a hierarchical indexing framework that uses a Z-order curve. However, it can only be applied to regular array-based datasets. Among the datasets we have used, this method will not even be applicable to the cosmology dataset. The WTSP Tree method [43] builds a wavelet-based time-space partitioning tree over large-scale time-varying datasets and supports multi-level data sampling on that. The entire dataset has to be reorganized and the WTSP Tree building process is time consuming.

Sampling has also been studied in the context of databases. One area of emphasis has been online aggregation, with initial work by Hellerstein *et al* [17]. Jermaine *et al* [19] proposed an online aggregation method for the DBO engine. Histograms [34] and wavelets [8] can be pre-computed and used. Chaudhuri *et al* [9] have conducted extensive studies on executing approximate aggregation queries using workload information and biased samples. More recent work in the database community has been in the context of speeding up map-reduce jobs with sampling. One initial study [16] proposed a framework to support incremental data sampling. EARL [28] involves a new sampling strategy with support for early error approximation based on bootstrapping, which has been widely employed in statistics and can be applied to arbitrary functions and data distributions. This method is able to decrease the resampling times and achieve good efficiency and accuracy. However, resampling is still needed to generate a satisfying sampling result.

Dissemination and analysis of large-scale and distributed datasets has been the focus of other studies as well. Some of the popular directions have been replica services [7, 10], reliable and predictable data transfers [3, 41], and constructing workflows [1, 13]. Chimera is a system for supporting virtual data views and demand-driven data derivation [15]. Metadata cataloging and metadata services have also been developed [14, 36]. The Metadata Catalog Service (MCS) [37] and Artemis [40] are collaborative components used to access and query repositories based on metadata attributes. Many middleware efforts have specifically focused on the needs of data-driven sciences [5], and enhancing and optimizing data transfer frameworks has been a popular topic [3, 25, 26, 29, 31, 23]. Our sampling techniques can work in conjunction with these efforts to make it feasible to analyze large-scale datasets.

6. CONCLUSIONS

This paper has described a novel sampling method for massive scientific simulation datasets. We utilize the value distribution and spatial locality features of bitmap indices and have developed an accurate sampling method over multi-level bitmap indices. We also developed an error prediction mechanism to pre-calculate error metrics before sampling the data. Moreover, with the help of bitmap indexing, our method is able to support data sampling over any combination of value subset and dimension subset.

7. ACKNOWLEDGMENTS

This work was supported by the Department of Energy (DOE) Office of Science (OSC) Advanced Scientific Computing Research (ASCR) and NSF award IIS-0916196 to the Ohio State University.

8. REFERENCES

[1] David Abramson and Jagan Kommineni. A Flexible IO Scheme for Grid Workflows . In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*, April 2004.

[2] James Ahrens, Berk Geveci, and Charles Law. Paraview: An end user tool for large data visualization. *the Visualization Handbook. Edited by CD Hansen and CR Johnson. Elsevier*, 2005.

[3] W. E. Allcock, I. Foster, and R. Madduri. Reliable Data Transport: A Critical Service for the Grid. In *Proceedings of the Workshop on Building Service Based Grids*, 2004.

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

[5] Andrew Baranovski, Keith Beattie, Shishir Bharathi, Joshua Boverhof, John Bresnahan, Ann Chervenak, Ian Foster, Tim Freeman, Dan Gunter, Kate Keahey, Carl Kesselman, Rajkumar Kettimuthu, Nick Leroy, Michael Link, Miron Livny, Ravi Madduri, Gene Oleynik, Laura Pearlman, Robert Schuler, and Brian Tierney. Enabling petascale science: Data management, troubleshooting, and scalable science services. *Journal of Physics: Conference Series*, 125, 2008.

[6] D. Bernholdt, S. Bharathi, D. Brown, K. Chanchio, M. Chen, A. Chervenak, L. Cinquini, B. Drach, I. Foster, P. Fox, et al. The earth system grid: Supporting the next generation of climate modeling research. *Proceedings of the IEEE*, 93(3):485–495, 2005.

[7] M. Cai, A. Chervenak, and M. Frank. A Peer-to-Peer Replica Location Service Based on A Distributed Hash Table. In *Proceedings of SC 2004*, November 2004.

[8] Kaushik Chakrabarti, Minos Garofalakis, Rajeev Rastogi, and Kyuseok Shim. Approximate query processing using wavelets. *VLDB Journal*, 10:199–223, 2001.

[9] Surajit Chaudhuri, Gautam Das, Mayur Datar, Rajeev Motwani, and Vivek Narasayya. Overcoming limitations of sampling for aggregation queries. In *Proceedings of ICDE 1999*, pages 534–542, 1999.

[10] A.L. Chervenak, N. Palavalli, S. Bharathi, C. Kesselman, and R. Schwartzkopf. Performance and Scalability of a Replica Location Service. In *Proceedings of the Conference on High Performance Distributed Computing (HPDC)*, June 2004.

[11] 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.

[12] W.G. Cochran. *Sampling techniques*. Wiley-India, 2007.

[13] Ewa Deelman, Jim Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, Albert Lazzarini, Adam Arbree, Richard Cavanaugh, and Scott Koranda. Mapping Abstract Complex Workflows onto Grid Environments. In *Journal of Grid Computing*, pages 9–23, 2003.

[14] Ewa Deelman, G. Singh, M.P. Atkinson, A. Chervenak, N.P. Chue Hong, C. Kesselman, S. Patil, L. Pearlman, and M. Su. Grid-Based Metadata Services. In *Proceedings of the 16th International Conference on Scientific and Statistical Database Management (SSDBM04)*, 2004.

[15] I. Foster, J. Voeckler, M. Wilde, and Y. Zhao. Chimera: A Virtual Data System for Representing, Querying and Automating Data Derivation. In *Proceedings of the Conference on Scientific and Statistical Data Management*, July 2002.

[16] R. Grover and M.J. Carey. Extending map-reduce for efficient predicate-based sampling. In *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*, pages 486–497. IEEE, 2012.

[17] Joseph M. Hellerstein, Peter J. Haas, and Helen J. Wang. Online aggregation. In *Proceedings of SIGMOD 1997*, 1997.

[18] Y. Ioannidis and V. Poosala. Histogram-based approximation of set-valued query-answers. In *Proceedings of the International Conference on Very Large Data Bases*, pages 174–185, 1999.

[19] Christopher Jermaine, Subramaniam Arumugam, Abhijit Pol, and Alin Dobra. Scalable approximate query processing with the dbo engine. In *Proceedings of SIGMOD 2007*, pages 725–736, 2007.

[20] W. Jiang, V.T. Ravi, and G. Agrawal. A map-reduce system with an alternate api for multi-core environments. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pages 84–93. IEEE Computer Society, 2010.

[21] C.R. Johnson and A.R. Sanderson. A next step: Visualizing errors and uncertainty. *Computer Graphics and Applications, IEEE*, 23(5):6–10, 2003.

[22] 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.

[23] Rajkumar Kettimuthu, Alex Sim, Dan Gunter, Bill Allcock, Peer-Timo Bremer, John Bresnahan, Andrew Cherry, Lisa Childers, Eli Dart, Ian Foster, Kevin Harms, Jason Hick, Jason Lee, Michael Link, Jeff Long, Keith Miller, Vijaya Natarajan, Valerio Pascucci, Ken Raffenetti, David Ressman, Dean Williams, Loren Wilson, and Linda Winkler. Lessons learned from moving earth system grid data sets over a 20 gbps wide-area network. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing (HPDC 2010)*, Jun 2010.

[24] M.F. Khairoutdinov and D.A. Randall. A cloud resolving model as a cloud parameterization in the ncar community climate system model: Preliminary results. *Geophys. Res. Lett*, 28(18):36173620, 2001.

[25] Ezra Kissel, D. Martin Swany, and Aaron Brown. Improving GridFTP performance using the Phoebus session layer. In *Proceedings of SC*, November 2009.

[26] T. Kosar and M. Livny. Stork: Making Data Placement a First Class Citizen in the Grid. In *Proceedings of International Conference on Distributed Computing Systems (ICDCS)*, 2004.

[27] E.C. LaMar, B. Hamann, and K.I. Joy. Efficient error calculation for multiresolution texture-based volume visualization. *Hierachical and Geometrical Methods in Scientific Visualization*, pages 51–62, 2003.

[28] N. Laptev, K. Zeng, and C. Zaniolo. Early accurate results for advanced analytics on mapreduce. *Proceedings of the VLDB Endowment*, 5(10):1028–1039, 2012.

[29] Wantao Liu, Brian Tieman, Rajkumar Kettimuthu, and Ian Foster. A data transfer framework for large-scale science experiments. In *3rd International Workshop on Data Intensive Distributed Computing (DIDC 2010) in conjunction with 19th International Symposium on High Performance Distributed Computing (HPDC 2010*, 2010.

[30] S.L. Lohr. *Sampling: design and analysis*. Thomson, 2009.

[31] D. Lu, Y. Qiao, P. A. Dinda, and F. E. Bustamante. Modeling and Taming Parallel TCP on Wide Area Networks. In *Proceedings of the 12th International Parallel and Distributed Processing Symposium (IPDPS)*, April 2005.

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

[33] V. Pascucci and R.J. Frank. Global static indexing for real-time exploration of very large regular grids. In *Supercomputing, ACM/IEEE 2001 Conference*, pages 45–45. IEEE, 2001.

[34] V. Poosala and V. Ganti. Fast approximate query answering using precomputed statistics. In *Proceedings of ICDE 1999*, page 252, 1999.

[35] V. Poosala, P.J. Haas, Y.E. Ioannidis, and E.J. Shekita. Improved histograms for selectivity estimation of range predicates. *ACM SIGMOD Record*, 25(2):294–305, 1996.

[36] G. Singh, S. Bharathi, A. Chervenak, E. Deelman, C. Kesselman, M. Mahohar, S. Pail, and L. Pearlman. A Metadata Catalog Service for Data Intensive Applications. In *Proceedings of Supercomputing 2003 (SC2003)*, November 2003.

[37] Gurmeet Singh, Shishir Bharathi, Ann Chervenak, Ewa Deelman, Carl Kesselman, Mary Manohar, Sonal Patil, and Laura Pearlman. A metadata catalog service for data intensive applications. In *SC '03: Proceedings of the 2003 ACM/IEEE Conference on Supercomputing*, page 33, Washington, DC, USA, 2003. IEEE Computer Society.

[38] Y. Su and G. Agrawal. Supporting user-defined subsetting and aggregation over parallel netcdf datasets. In *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 212–219. IEEE, 2012.

[39] 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.

[40] Rattapoom Tuchinda, Snehal Thakkar, A Gil, and Ewa Deelman. Artemis: Integrating scientific data on the grid. In *Proceedings of the 16th Conference on Innovative Applications of Artificial Intelligence (IAAI*, pages 25–29, 2004.

[41] S. Vazhkudai and J. Schopf. Using disk throughput data in predictions of end-to-end grid transfers. In *Proceedings of the Third Workshop on Grid Computing (Grid 2002)*, November 2002.

[42] J.S. Vitter. An efficient algorithm for sequential random sampling. *ACM transactions on mathematical software (TOMS)*, 13(1):58–67, 1987.

[43] C. Wang, A. Garcia, and H.W. Shen. Interactive level-of-detail selection using image-based quality metric for large volume visualization. *Visualization and Computer Graphics, IEEE Transactions on*, 13(1):122–134, 2007.

[44] J. Woodring, J. Ahrens, J. Figg, J. Wendelberger, S. Habib, and K. Heitmann. In-situ sampling of a large-scale particle simulation for interactive visualization and analysis. In *Computer Graphics Forum*, volume 30, pages 1151–1160. Wiley Online Library, 2011.

[45] 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.

[46] 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.

[47] Kesheng 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, 2003*, pages 65– 74. IEEE, July 2003.

[48] L. Xu, T.Y. Lee, and H.W. Shen. An information-theoretic framework for flow visualization. *Visualization and Computer Graphics, IEEE Transactions on*, 16(6):1216–1224, 2010.