DSS: A Scalable and Efficient Stratified Sampling Algorithm for Large-Scale Datasets

Minne Li, Dongsheng Li, Siqi Shen, Zhaoning Zhang, Xicheng Lu

DSS: A Scalable and Efficient Stratified Sampling Algorithm for Large-Scale Datasets

Minne Li, Dongsheng Li, Siqi Shen, Zhaoning Zhang, and Xicheng Lu

National Laboratory for Parallel and Distributed Processing,
National University of Defense Technology, Changsha 410073, China
litoeknee@gmail.com,dsli@nudt.edu.cn

Abstract. Statistical analysis of aggregated records is widely used in various domains such as market research, sociological investigation and network analysis, etc. Stratified sampling (SS), which samples the population divided into distinct groups separately, is preferred in the practice for its high effectiveness and accuracy. In this paper, we propose a scalable and efficient algorithm named DSS, for SS to process large datasets. DSS executes all the sampling operations in parallel by calculating the exact subsample size for each partition according to the data distribution. We implement DSS on Spark, a big-data processing system, and we show through large-scale experiments that it can achieve lower data-transmission cost and higher efficiency than state-of-the-art methods with high sample representativeness.

Keywords: stratified sampling, aggregation, distributed processing, Spark

1 Introduction

With the rapid advancement in the data collection and storage technology, burgeoning data size has brought both opportunities and challenges to driving business decisions. Valuable information and knowledge could be extracted from the ever-increasing datasets. To gain knowledge from the dataset, various computing techniques such as cloud computing [15, 23] and big-data processing [5, 22] have been proposed, which provide us the computing platform to process the data. However, it is time-consuming to process the ever-increasing and massive-scale data. Data sampling techniques can be used to gain statistical results, estimation and approximation of data in short time with slightly reduced accuracy. They are playing critical roles in areas such as social network analysis [8, 18] and business market research[4]. In this work, we focus on stratified sampling (SS), a widely adopted sampling technique [20] with high efficiency and accuracy. We implement a Distributed Stratified Sampling (DSS) algorithm, and the experimental results show that our algorithm can be 65% faster than a well-known distributed SS algorithm [14] with high-accuracy and high-scalability.

Basic steps of sampling consist of extracting a representative subset of the population, performing the estimation and experiment, and extrapolating the results back in order to understand characteristics of the overall population.

Specifically, stratified sampling is a sampling method involving the division of a population into distinct groups known as *strata* (homogeneous subgroups, in which the inner items are similar to each other). It assigns individuals of the surveyed population to *strata*, and then applies normal sampling methods (e.g., simple random sampling (SRS) or systematic sampling) without replacement to each *stratum* independently. Compared to SRS, stratified sampling is able to get higher statistical precision because the variability within subgroups sharing the same properties is lower than that of the entire population [20]. Thus stratified sampling improves the representativeness by reducing sampling error. In addition, having a higher statistical precision enables stratified sampling to tolerate smaller sample size than other methods, which helps to save time and effort of researchers. Consequently, stratified sampling outperforms other sampling methods both in efficiency and accuracy. Traditional implementation of stratified sampling, e.g., *reservoir sampling*, is not designed for distributed computing environment. Although several distributed implementations [14, 22] have been proposed, they are either not able to generate a statistical satisfied answer under certain conditions or not able to fully utilize the computing resources. We will further discuss this point in Section 2 and Section 4.

In this work, we propose a distributed algorithm for SS which is scalable and efficient. It has four main steps: (1) conducts a modified reservoir sampling with rough sampling size inside each data partition, (2) gathers the meta-data of intermediate results, (3) computes the exact sampling size for each partition and (4) performs a modified reservoir sampling in parallel to generate the final sampling results. Different from the method *Spark Single Query Evaluator* (Spark-SQE) proposed in [14], DSS reduces significantly the computational cost by conducting sampling process in a distributed manner. Moreover, the data transfer cost is reduced considerably because the computation phases are conducted in distributed nodes instead of in a master node, meaning that the volume of data transferred is much reduced.

The remainder of this paper is structured as follows. In Section 2, we compare our work with related research with a focus on scalable sampling techniques. In Section 3, we further discuss the definition and provide the syntax and semantics of stratified sampling queries, and present the sequential stand-alone algorithm of answering a single stratified sampling query. In Section 4, we describe the design of the algorithm. In Section 5, we show the experimental results, and in Section 6, we conclude this work.

2 Related Work

A lot of research effort has been made to design scalable algorithms for processing large-scale datasets. Boyd et al. [2] have investigated the alternating direction method of multipliers to solve distributed convex optimization problem, and Owen et al. [17] have introduced Mahout to apply machine learning algorithm against large datasets. However, many of these algorithms cannot generate results within an acceptable range of time without reducing data size[16].

In order to reduce the storage and computational cost as well as keep important statistical properties of the original data, researchers have proposed various data sampling algorithms. Gjoka et al. [7] have implemented a multi-graph sampling method for online social network datasets to generate representative samples for highly clustered individual social graphs. Kurant et al. [12] have utilized stratification to generate weighted graphs for efficient data crawling and metric estimation. However, these works are not designed for distributed computing environment [14].

In terms of stratified sampling, numbers of previous works have been proposed for the stand-alone environment. One of the many classical methods is the *reservoir sampling* algorithm [21], which requires a single pass over the whole dataset to generate representative results. However, the original *reservoir sampling* algorithm is not designed for distributed computing environments: data shuffling among clusters for a single query is required because the partition of data into clusters is mostly different to the partition of the population into strata. However, conducting data shuffling is unbearable in the big data environment. In order to design a scalable sampling algorithm and implement it on top of the data stored on distributed machines, the sampling process should be conducted in a parallel and distributed manner. Spark [22], a platform for large-scale datasets processing, provides the distributed stratified sampling API as one of its basic statistic functions, namely *sampleByKey* and *sampleByKeyExact*. These functions conduct sampling with given sampling probability. However, in order to draw a stratified sampling set by sample size in Spark, users need to provide the total count of records satisfied the stratum constraint, which is not possible in the practical, distributed computing environment. Even if the actual count is provided, the existing function could still fail the statistical requirement if the product of sampling fraction and total record count is not an integer. Levin et al. [14] have proposed a framework for stratified sampling queries, which will be further discussed in Section 3.

3 Stratified Sampling Queries

In this section we will provide the definition, syntax and semantics of stratified sampling queries, as well as present the sequential stand-alone algorithm (the *modified reservoir sampling algorithm*) of answering a single stratified sampling query. We will further discuss in Section 4 that our sequential algorithm can be conducted in a parallel and distributed manner effortlessly.

3.1 A Single Stratified Sampling Query

The notations used in this work is defined as follows. A single stratified sampling query in this paper is defined as a set of stratum constraints s_k, denoted by $Q = (s_1, s_2, \ldots, s_m)$. Each stratum constraint s_k is denoted by $s_k = (p_k, f_k)$, where p_k is a propositional formula and f_k is the required sample size. It is worth noting that for a qualified stratified sampling query, the strata must be

non-overlapping, in other words every individual should be assigned to only one stratum. Joint strata will result in nonprobability sampling since some individuals may have greater chances of being chosen. For example, if we define two stratum constraints s_1 and s_2 for a group of students, where $p_1 = (male)$ and $p_2 = (age > 10)$, conducting two simple random sampling separately in stratum defined by s_1 and s_2 will give males over 10 years old greater chances of being selected.

3.2 Sequential Answering Process

A valid answer to a query $Q = (s_1, s_2, \ldots, s_m)$ is the union of m disjoint sample sets where (1) every single individual in subset k ($k = 1, 2, 3, \ldots, m$) satisfies the propositional formula p_k and (2) there are exactly f_k individuals in subset k. In addition, a statistically representative answer set should guarantee that each subset k is a simple random sample of all the individuals in the population that satisfy p_k. We will further discuss this point in Section 4 and Section 5.

Algorithm 1 Reservoir sampling for stratum constraint s_k

1: Store the first f_k individuals satisfied p_k into a reservoir R_k
2: **for** j from $f_k + 1$ to n **do**
3: With probability f_k/j, randomly choose an individual from R_k and replace it with individual j
4: **end for**
5: **return** Individuals in R_k

After providing the definition of a single stratified sampling query, we present a sequential stand-alone algorithm for generating the subset satisfying a stratum constraint, which is listed in Algorithm 1. Algorithm 1 is derived from the *reservoir algorithm* [21] with almost the same procedure. The algorithm creates a *reservoir* array of size f_k with the first f_k items satisfied p_k from the population containing n individuals. Then the algorithm iterates through the remaining population. At the j^{th} iteration, the algorithm randomly chooses an individual from the *reservoir* and replace it by individual j with probability f_k/j. It can be proved that at the end of the iteration, every individual has equal probability (i.e. f_k/n) of being chosen for the *reservoir*.

Thanks to the disjoint property of strata, answering a query $Q = (s_1, s_2, \ldots, s_m)$ requires only single pass over the population set sequentially if we maintain m *reservoirs* R_k and m element indexes j_k. Each *reservoir* contains f_k individuals satisfied p_k respectively and thus holds a simple random sample of the processed individuals at any step during the execution.

4 Distributed Sampling Design

This section commences by focusing on the detailed design of our algorithm *Distributed Stratified Sampling* DSS. We first present the algorithm which reduces

significantly the data transmission cost among distributed nodes by sending the abstract of intermediate result rather than the result itself. In addition, DSS considerately reduces the computational cost compared to existing alternatives by conducting all the sampling phases in a distributed manner. The experimental comparison will be covered in Section 5.

4.1 Sampling Representativeness in Distributed Environment

We now give the definition of representativeness in distributed environment can be seen as a cluster of connected computers (the nodes) or virtual machines provided by cloud computing services including Amazon EC2, Microsoft Azure, etc. Note that in the production environment, data is already stored in the file system and distributed separately to nodes. We are required to answer the sampling query without changing the original position of data. To generate a statistically representative answer set to a single stratified sampling query, the sample result should be unbiased in the first place. Because the partition of the population into strata is always different to the partition of data into clusters, data shuffling is required to response a sampling query. However, this is unrealistic in industrial and practical environment. Consequently, the proportion of satisfied elements in each data partition must be taken into account.

For example, we are asked to generate a sample of 10 male students playing basketball from a population of 50 students. The satisfied population (all the male students playing basketball in the population) data is distributed among two separate nodes, 20 in node O_1 and 30 in node O_2. A simple solution is to generate an intermediate sample of 10 male students playing basketball for both of nodes, and then conduct a unification process to produce the final answer set. However, this approach will trigger a biased, statistically invalid sample. The above approach produces a male student playing basketball selected for intermediate a probability of $1/2$ and $1/3$, for O_1 and O_2 respectively. After conducting a simple random sampling to select 10 individuals among intermediate results, the probability of individuals from O_1 and O_2 appearing in the final answer set will be $1/4$ and $1/6$ respectively. This answer, however, is biased and statistically invalid since each individual should have a chance of $1/5$ to be selected for the query. Consequently, in order to have a uniform and unbiased sample, the proportion of satisfied elements in each data partition located in each node must be taken into account for deciding the selection ratio from each intermediate sample. In this case specifically, the intermediate results from O_1 and O_2 should be selected with probability of $1/2$ and $1/3$ respectively.

4.2 Distributed Algorithm Spark-SQE

In Section 3.2, we present the *modified reservoir algorithm* (Algorithm 1) for answering sequentially a single stratified sampling query. By combining this algorithm with the sampling method discussed in Section 4.1, we further describe a

distributed version of Algorithm 1, namely *Spark Single Query Evaluator* (Spark-SQE), derived from the *Map Reduce Single Query Evaluator* provided by [14]. The procedure of Spark-SQE is depicted in Figure 1.

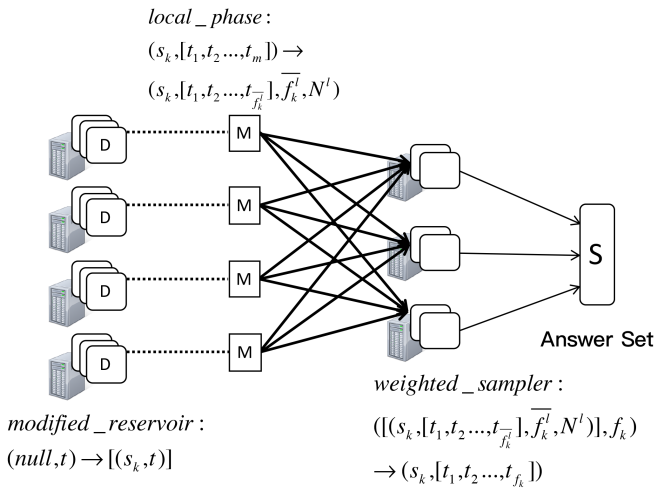

Fig. 1. Process of Spark-SQE, including the map phase (*Algorithm 1*), the *local phase* and the *weighted sampler*. The symbolic tuples besides the arrow in each phase represents input and output for this phase respectively. The local transformation and computation between the raw data and the intermediate sample is connected by *dotted line*. Data transmission among clusters is represented by *solid lines*. *Thick lines* indicate large data-transmission while the *thin ones* indicate the opposite. Each D, M and S stands for data partition, intermediate sample, and final result respectively.

In each data partition, similar process as Algorithm 1 is conducted to generate local intermediate result, of the form $[(s_k, t)]$ which is a list of tuple (s_k, t) if individual t satisfies s_k. A parallel execution of Algorithm 1 can be seen as a *map* phase specified by Spark, which deals with key-value pairs to generate a set of intermediate key-value pairs of the form $[(s_k, t)]$.

To gather the intermediate results, one of the naive implementations will copy all the satisfied individuals from the whole cluster to the master node. However, this procedure cannot fully utilize the computing resources.

Instead of the naive method above, during the *local* phase of Spark-SQE, we can merely collect a certain amount of intermediate sample. For example, we can construct a simple random selection of $\overline{f_k^l}$ individuals for each data partition l before passing them to the *weighted sampler* ($\overline{f_k^l}$ could be less than f_k if not enough satisfied individuals exist in some certain data partitions). The *local* phase generates the local sample set satisfied s_k, the sampling size $\overline{f_k^l}$ and N^l, the total number of individuals satisfied s_k in this partition. This approach could considerably reduce the data sent over the network and increase concurrency.

Lastly, a *weighted sampler* merges all intermediate results, which are already shrunk by the *local* process in each data node, and conducts a weighted random selection of size f_k. This approach will generate a representative answer because the final sampling size is proportional to N^l of each intermediate result as we described in Section 4.1.

4.3 Improved Distributed Algorithm DSS

By sending the meta-data rather than original data to the *reduce* function after conducting the *local* phase in Spark-SQE, we can further reduce the amount of data transmitted over the network. Figure 2 illustrates the process of *Distributed Stratified Sampling* (DSS) which is an improved version of Spark-SQE.

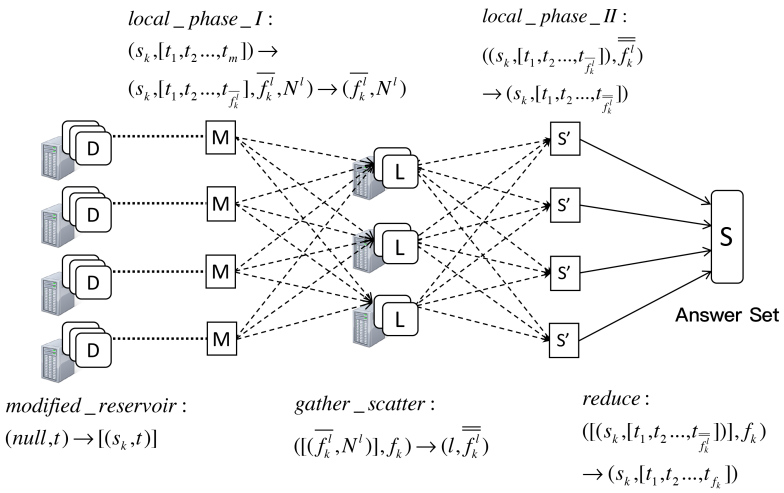

Fig. 2. Process of DSS, including the map phase (*Algorithm 1*), the *local phase I*, the *gather scatter* phase, the *local phase II* and the *reduce phase*. The *dotted lines* represent local transformation and computation between the raw data and the intermediate sample. In addition, the data and meta-data transmission among clusters are represented by *solid lines* and *dashed lines* respectively. Each *D*, *M*, *L*, *S'* and *S* stands for data partition, intermediate sample, required sampling frequency list, local weighted result and final result respectively.

The *map* phase of DSS is derived from Spark-SQE and generates the same result. In the *local phase I*, DSS has already generated an intermediate sample of $\overline{f_k^l}$ from N^l individuals satisfied p_k in data partition l as the Spark-SQE. To reduce the data transfer cost, we can simply send $\overline{f_k^l}$ and N^l instead of the whole intermediate result to the master node which will execute the *gather-scatter* function. Then in the *gather-scatter* phase, DSS computes the required sampling frequency $\overline{\overline{f_k^l}}$ of s_k for each data partition using weighted sampling.

These required sampling frequency $\overline{\overline{f_k^l}}$ of each partition is delivered to each data node separately. Thirdly, a local simple random selection shown as the *local phase II* in DSS is conducted to generate local parts of the final answer set. Compared to Spark-SQE, the computational cost is reduced considerately as well since we conduct the weighted sampling phase in different nodes separately rather than running a sequential process in a single reduce node. Finally, the *reduce* phase in DSS gathers every part of the final result from each data node together to compose a correct, unbiased and uniform answer to the single stratified sampling query.

The *gather-scatter algorithm* listed in Algorithm 2 receives L meta-data tuples of intermediate samples for s_k from L data partitions. Each tuple consists of the actual intermediate sample size $\overline{f_k^l}$ and N^l, the total number of satisfied individuals in data partition l. At first, the algorithm checks if the sum of intermediate sample size is enough for the required sample size f_k (Line 1). If not, the algorithm simply returns $\overline{f_k^l}$ as the weighted sample size for each data partition l to generate the final answer (Line 2). Otherwise, the algorithm continues by iterating over 1 through L to compute an index list for determining the exact sample size of each data partition (Line 8 to 13). Because the index list is constructed based on the count of satisfied individuals N^l in data partition l, it is thus probabilistically proportional to N^l.

Algorithm 2 Gather-scatter $([(\overline{f_k^l}, N^l)], f_k) \rightarrow [(l, \overline{\overline{f_k^l}})]$

1: **if** $\sum_{l=1}^{L} \overline{f_k^l} < f_k$ **then**
2: **return** $[(l, \overline{f_k^l})], l \in [1, L]$
3: **end if**
4: $N \leftarrow \sum_{l=1}^{L} N^l$
5: $I \leftarrow$ *Randomly select n indexes from* $[0, N]$
6: $low\_bound \leftarrow 0$
7: $up\_bound \leftarrow N^1$
8: $\overline{\overline{f_k^l}} \leftarrow 0, l \in [1, L]$
9: **for** l from 1 to L **do**
10: $\overline{\overline{f_k^l}} \leftarrow |I \cap [low\_bound, up\_bound]|$
11: $low\_bound \leftarrow low\_bound + N^l$
12: **if** $l < L$ **then**
13: $up\_bound \leftarrow up\_bound + N^{l+1}$
14: **end if**
15: **end for**
16: **return** $[(l, \overline{\overline{f_k^l}})], l \in [1, L]$

The *weighted sampler* phase in Spark-SQE is similar as the *gather-scatter algorithm*. Instead of the meta-data, it receives the intermediate sample and then generates the final result, thus it requires significantly larger amount of data transfer compared to the *gather-scatter algorithm*.

It is trivial to prove the correctness (a simple random sample) of the *gather-scatter algorithm* by induction. The proof can be drawn inductively over the size n of the final sample. Basically, we need to prove that every subset of the population of size n has equal probability of selection, for every n.

5 Experimental Evaluation

In this section, we conduct large-scale and systematic experiments to evaluate the proposed DSS and compare it with Spark-SQE, which is the Spark implementation of a well-known stratified method provided in [14]. We describe the experimental setup in Section 5.1, and demonstrate the efficiency and scalability of the algorithms by examining the running times and transmitted data size. We also discuss how the sampled data generated by DSS precisely represent the original data.

5.1 Experimental Setup

Dataset. In this experiment, we use two real-world datasets and one synthetic dataset shown in Table 1. The two real-world datasets are the LiveJournal social network dataset [13] and the Twitter-2010 social network dataset [1]. These datasets are natural graphs following the *power-law* [6] distribution, where only a few vertices have large numbers of neighbors while most of the vertices have relatively few neighbors. We pre-process the datasets by changing the graph storage structure from edge list to adjacency list. The synthetic graph dataset is generated through Power Graph API [9]. By adjusting the *power-law* exponent constant α we can control the skewness of the degree distribution, where a lower α implies a higher graph density and larger number of high-degree vertices. Because the adjacency list stores all neighbors of a vertex in a single record, a lower α will result in a larger storage size of high degree vertices records.

Stratified Sampling Query. We generate the sampling query by considering the out-degree of vertices. The strata are created by partitioning the out-degree of vertices into sub-ranges, which are represented by propositional formulas, e.g., $(\textit{out-degree} < 20)$. We generate valid sampling query by randomly selecting non-overlapping degree ranges. For example, $Q = (s_1, s_2)$ where $s_1 = (\textit{out-degree} < 20, 2000)$ and $s_2 = (\textit{out-degree} > 50, 1000)$ represents a query with two distinct strata where the out-degree of vertices is lower than 20 and higher than 50.

Environment. We have implemented the algorithms in Apache Spark framework. The Spark environment is built on top of a cluster consisting four nodes, with one serving as the master and three as worker. Each node is configured with Ubuntu 14.04 LTS, 47 GB RAM, 8 Intel Xeon E5-1620 CPUs, and 2.7TB storage. All the data is stored in HDFS.

Table 1. Properties of the dataset.

Dataset	$\|V\|$	$\|E\|$	Size
LiveJournal	4,847,571	68,993,773	514M
alpha1.8	9,999,999	641,383,778	4.6G
Twitter-2010	41,652,230	1,468,365,182	13G

5.2 Results

In the experiment, we evaluate the efficiency and scalability of the algorithms in terms of the runtime and transmitted data size, respectively. The runtime is counted from receiving a query to generating a final result. The transmitted data size is measured by calculating the data received in *weighted sampler* phase for Spark-SQE and *gather-scatter* phase for DSS. The result has eliminated the time of loading the raw data into memory from HDFS. Moreover, we perform experiments to evaluate the representativeness and quality of the stratified sampled data generated by DSS. For the social network datasets, we calculate their degree distributions and check if the sampled data have the same degree distribution as the original data.

Efficiency. Figure 3 illustrates the relative runtime of the algorithms grouped by different datasets. The type of query has been classified by different scales, which is the total number of records satisfied the stratum constraint. We generate the query by defining the large group as $p_{large} = (\textit{out-degree} < 50)$ and the small group as $p_{small} = (\textit{out-degree} > 200)$ and use the same sampling frequency as 2000. The small group indicates the "long tail" of the *power-law* graph, which represents vertices with many neighbors. On the contrary, vertices with few neighbors comprise the large group. According to the *power-law* we discussed in Section 5.1, the number of records in large group is significantly bigger than the small group.

As is shown in Figure 3, the runtime of DSS can be only 35% of Spark-SQE in the best case. Even in the worst case, the runtime of DSS is about 65% of that of Spark-SQE. This achievement is mostly attributed to our distributed sampling process in the *weighted sampling* phase. The enhancement is not directly related to the data size. The most time-consuming phases of the two algorithms are the *local* phase and the *reduce* phase (*weighted sampler* phase for Spark-SQE). For the *local* and *map* phase both algorithms are totally conducted in parallel. For the *reduce* phase, only DSS runs in parallel and the time used in this phase is directly related to the sampling frequency. Because we use the same sampling frequency and a growing data size, according to the result shown in Figure 3, we can conclude that the *local* and *map* phase contributes more to the running time. This conclusion is theoretically explainable as the *local* and *map* phase needs to conduct an iteration over all records while the *reduce* phase only needs an iteration of 2000 records in our experiment.

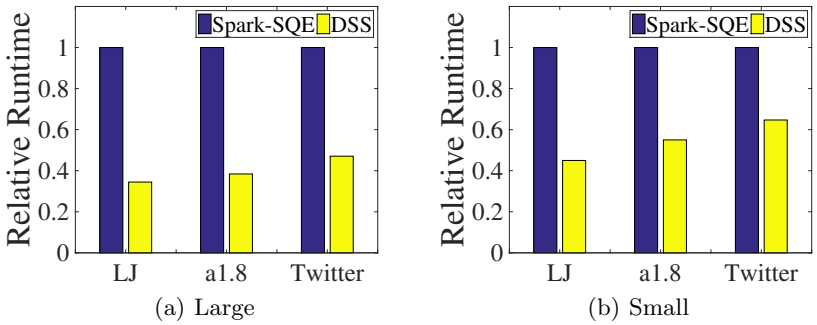

Fig. 3. The running time of DSS divided by Spark-SQE for different query group: (left) large group and (right) small group.

Scalability. As we mentioned above, the enhancement of our algorithm is directly related to the sampling frequency. To evaluate the scalability of our algorithm, we create a group of stratum constraints, which ask for sampling sizes ranging from 1000 to 4000. The running time for these groups of stratum constraints is shown in Figure 4.

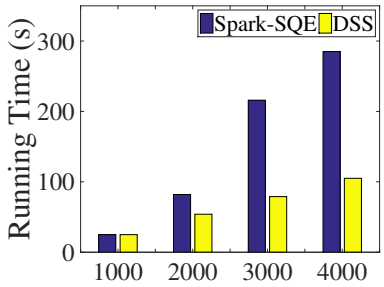

Fig. 4. The running time of Spark-SQE and DSS under different sampling frequencies for the LiveJournal dataset.

Figure 4 illustrates the runtime of DSS under different sampling frequencies can reach a linear improvement. Compared to Spark-SQE, a relative enhancement of 65% is drawn in the best case. Note that we almost touch the upper bound of optimization in our experiment environment since we have only three executors.

Transmission Cost. Table 2 shows the comparison of network transmission cost for the two algorithms. We focus on the intermediate data collection phase and use the LiveJournal dataset. As is depicted in Table 2, the size of the trans-

Table 2. The size of data transmitted in bytes during the intermediate data collection phase of the algorithms using LiveJournal dataset under different workloads, which represents by a stratum constraint $s_k = (p_k, f_k)$ as described in Section 3.1.

Workload	Spark-SQE	DSS
$s_1 = (\textit{out-degree} < 10, 1000)$	2,017,416	112
$s_2 = (\textit{out-degree} < 10, 2000)$	4,028,816	112
$s_3 = (\textit{out-degree} < 10, 3000)$	6,048,232	112
$s_4 = (\textit{out-degree} < 10, 4000)$	8,064,840	112

mitted data of DSS is much smaller than that of Spark-SQE. The reason is that the data transfer cost of Spark-SQE is proportional to sampling frequency, while that of DSS remains the same across different frequencies. This is due to the fact that DSS only transmits the metadata (a tuple of two integers per partition), whose size increases linearly with the number of partitions in clusters. In contrast to DSS, Spark-SQE sends an array of all intermediate records for each partition, thus performs poorly when the sampling size increases.

Sample Representativeness. Degree distribution is an important property of social network and has frequently been analyzed in much social network research [3, 10]. We evaluate the representativeness of the sampled datasets through evaluating the difference between the degree distributions of the stratified sampled datasets and that of the original datasets using *Kolmogorov–Smirnov test* (K–S test) [11, 19]. The K–S test works as follows. It firstly generates cumulative probability plots for the two data and calculate each vertical distance for a given x values between the two curves. Then the maximum distance (called K–S statistic) is searched from all the vertical distances. In the end, the *probability value* (p-value) is calculated by plugging this maximum distance into K–S probability function. The closer to 1 the p-value is the more likely the two distributions are similar, and vice versa.

Table 3. The K–S statistic and p-values of K–S tests for the three datasets used in this work. The stratum constraint is defined as $s_k = (\textit{out-degree} < 50, 1000)$.

Dataset	K–S statistic	p-value
LiveJournal	0.020336	0.798941
alpha1.8	0.017253	0.925117
Twitter-2010	0.016218	0.953649

As is shown in Table 3, the p-values of all the datasets are high (79.9%, 92.5%, 95.4%), thus we cannot reject the null hypothesis that the degree distributions

of stratified sampled data by DSS and that of original data are the same. In other words, the degree distributions of sampled data and original data share great similarities, which means the sampled data precisely represent the original data.

6 Conclusion

This paper proposed a distributed algorithm DSS for applying stratified sampling to large-scale, distributed datasets. DSS significantly reduces the computational cost of selecting stratified sample by implementing a *modified reservoir algorithm* inside each stratum in a distributed manner. Moreover, the data transfer cost is reduced significantly as well, by transmitting the meta-data instead of the data records. We implement DSS on Spark, a big-data processing platform, and evaluate the algorithm using two large-scale real-world datasets. The experiment results show that DSS performs well in terms of efficiency, scalability and representativity. Compared to Spark-SQE, which is a Spark implementation of state-of-the-art method, DSS reaches a relative enhancement of 65%. In addition, DSS requires extremely smaller amount (less than 0.05% in our experiment) of network resources in clusters than that of Spark-SQE.

Acknowledgments. This work is sponsored in part by the National Basic Research Program of China (973) under Grant No. 2014CB340303, the National Natural Science Foundation of China under Grant No. 61222205, the Program for New Century Excellent Talents in University, and the Fok Ying-Tong Education Foundation under Grant No. 141066.

References

1. Boldi, P., Vigna, S.: The webgraph framework i: compression techniques. In: Proceedings of the 13th international conference on World Wide Web. pp. 595–602. ACM (2004)
2. Boyd, S., Parikh, N., Chu, E., Peleato, B., Eckstein, J.: Distributed optimization and statistical learning via the alternating direction method of multipliers. Foundations and Trends® in Machine Learning 3(1), 1–122 (2011)
3. Clauset, A., Shalizi, C.R., Newman, M.E.J.: Power-law distributions in empirical data. SIAM Review 51(4), 661–703 (2009), http://dx.doi.org/10.1137/070710111
4. Cooper, D.R., Schindler, P.S., Sun, J.: Business research methods (2006)
5. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. Communications of the ACM 51(1), 107–113 (2008)
6. Faloutsos, M., Faloutsos, P., Faloutsos, C.: On power-law relationships of the internet topology. In: ACM SIGCOMM computer communication review. vol. 29, pp. 251–262. ACM (1999)
7. Gjoka, M., Butts, C.T., Kurant, M., Markopoulou, A.: Multigraph sampling of online social networks. Selected Areas in Communications, IEEE Journal on 29(9), 1893–1905 (2011)

8. Gjoka, M., Kurant, M., Butts, C.T., Markopoulou, A.: A walk in facebook: Uniform sampling of users in online social networks. arXiv preprint arXiv:0906.0060 (2009)
9. Gonzalez, J.E., Low, Y., Gu, H., Bickson, D., Guestrin, C.: Powergraph: Distributed graph-parallel computation on natural graphs. In: Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12). pp. 17–30 (2012)
10. Jia, A.L., Shen, S., van de Bovenkamp, R., Iosup, A., Kuipers, F.A., Epema, D.H.J.: Socializing by gaming: Revealing social relationships in multiplayer online games. TKDD 10(2), 11 (2015), http://doi.acm.org/10.1145/2736698
11. Kolmogorov, A.N.: Sulla determinazione empirica di una legge di distribuzione. na (1933)
12. Kurant, M., Gjoka, M., Butts, C.T., Markopoulou, A.: Walking on a graph with a magnifying glass: stratified sampling via weighted random walks. In: Proceedings of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems. pp. 281–292. ACM (2011)
13. Leskovec, J., Krevl, A.: SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data (Jun 2014)
14. Levin, R., Kanza, Y.: Stratified-sampling over social networks using mapreduce. In: Proceedings of the 2014 ACM SIGMOD international conference on Management of data. pp. 863–874. ACM (2014)
15. Lu, X., Wang, H., Wang, J., Xu, J., Li, D.: Internet-based virtual computing environment: beyond the data center as a computer. Future Generation Computer Systems 29(1), 309–322 (2013)
16. Meng, X.: Scalable simple random sampling and stratified sampling. In: Proceedings of the 30th International Conference on Machine Learning (ICML-13). pp. 531–539 (2013)
17. Owen, S., Anil, R., Dunning, T., Friedman, E.: Mahout in action. greenwich, ct (2011)
18. Papagelis, M., Das, G., Koudas, N.: Sampling online social networks. Knowledge and Data Engineering, IEEE Transactions on 25(3), 662–676 (2013)
19. Smirnov, N.: Table for estimating the goodness of fit of empirical distributions. The annals of mathematical statistics 19(2), 279–281 (1948)
20. Thompson, S.K.: Stratified Sampling, pp. 139–156. John Wiley & Sons, Inc. (2012), http://dx.doi.org/10.1002/9781118162934.ch11
21. Vitter, J.S.: Random sampling with a reservoir. ACM Transactions on Mathematical Software (TOMS) 11(1), 37–57 (1985)
22. Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M.J., Shenker, S., Stoica, I.: Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In: Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation. pp. 2–2. USENIX Association (2012)
23. Zhang, Z., Li, D., Wu, K.: Large-scale virtual machines provisioning in clouds: challenges and approaches. Frontiers of Computer Science 10(1), 2–18 (2016)