An Efficient Block Sampling Strategy for Online Aggregation in the Cloud

Xiang Ci and Xiaofeng Meng(✉)

School of Information, Renmin University of China, Beijing, China
{cixiang,xfmeng}@ruc.edu.cn

Abstract. As the development of social network, mobile Internet, etc., an increasing amount of data are being generated, which beyond the processing ability of traditional data management tools. In many real-life applications, users can accept approximate answers accompanied by accuracy guarantees. One of the most commonly used approaches is online aggregation. Online aggregation responds aggregation queries against the random samples and refines the result as more samples are received. In the era of big data, more and more data analysis applications are migrated to the cloud, so online aggregation in the cloud has also attracted more attention. There can be a huge difference between the number of tuples in each group when dealing with group-by queries. As a result, answers of online aggregation based on uniform random sampling can result in poor accuracy for groups with very few tuples. Data in the cloud are usually organized into blocks and this data organization makes sampling more complex. In this paper, we propose an efficient block sampling which can exactly reflect the importance of different blocks for answering group-by queries. We implement our methods in a cloud online aggregation system called COLA and the experimental results demonstrate our method can get results with higher accuracy.

Keywords: Online aggregation · Block sampling · Cloud computing

1 Introduction

Data-driven activities are rapidly growing in various applications, including Web access logs, sensor data, scientific data, etc. Distilling the meaning from these data has never been in such urgent demand. All these big data have brought in great challenges to traditional data management in terms of both data size and significance. As the growth of sheer volume of data, performing analysis and delivering exact result on big data can be extremely expensive, sometimes even impossible. For example, suppose one petabyte data are stored in the database and we want to find a particular record from the database. If indices have been built on the attribute we want to query, the answer will be returned soon. But if we do not build indices, how long will it take? The only way we can do without indices is full scan. Now for the fastest Solid State Drives (SSD) with disk scanning speed of about 6GB/s, a linear scan of SSD with one petabyte data will take about 166666 seconds, i.e. 2777 minutes, 46 hours, or 1.9 days. It is definitely unacceptable. Fortunately, in many real-life applications, people just want to obtain a bird's eye view of the whole dataset. This situation has brought more attention to the already-active area of Approximate Query Processing (AQP).

OLA is one of the most widely used AQP techniques and our work also focuses on OLA. OLA is first proposed in the area of relational database management system (RDBMS). The basic idea behind OLA is to estimate the result by sampling data and the approximate answer should be given some accuracy guarantees. Usually, we use confidence to measure the accuracy. In many scenarios, group-by queries play an important role: data are divided into groups and aggregated within these groups. Uniform random sampling is appropriate only when the utility of the data to the users mirrors the data distribution and less effective for group-by queries because of the problem of "small group". For example, consider a column R with 1000 tuples of which 99% have value 1, while the remaining 1% of the tuples have value 100. If we want to estimate the Sum of the two groups individually by 100 samples, accurate results can be obtained only when there are 99 samples with the value of 1 and 1 sample with the values of 100. If two or more value of 100 are sampled, the estimation will be great error. Here, the group with value of 100 can be considered as "small group". The problem of "small group" can significantly influence the accuracy of the results.

In recent years, with the development of cloud computing, OLA in the cloud has drawn more attention. Especially it can reduce the economic cost of users on the typically pay-as-you-go cloud systems and increase the overall throughput of the cloud system. Although OLA techniques have been extensively studied for RDBMS and OLA is very suitable for cloud environments, there are still challenges to adapt them to the cloud when group-by queries are invloved. First, data in the cloud are usually organized and processed in blocks which may contain thousands of tuples. The accuracy of OLA highly depends on the sampling. When data are organized in blocks, the minimum sampling unit is block. So the layout of the data in block, i.e., the way by which tuples are grouped into blocks, will greatly influence the accuracy. Second, we need to change the sampling method to make the samples reflect the distribution of the original data. Third, many cloud systems, such as Hadoop and Hyracks, are MapReduce-oriented. The naive MapReduce framework is batch processing, which is opposite to OLA. The Reduce phase of MapReduce can not start until the completion of the Map phase while OLA needs pipelining between different phases.

Motivated by above requirements and challenges, we try to solve the problem of "small group" for online aggregation in the Cloud. The specific contributions we make in this paper are:

- We introduce a stage of preprocessing which can randomize the data blocks efficiently and obtain relative data information.
- We propose an adaptive block sampling method which can solve the problem of "small group".

– We implement our method in COLA, a system for Cloud Online Aggregation. And experiments of the proposed technique are reported in terms of performance and accuracy.

The remainder of the paper is organized as follows. In Section 2, we summarize the related work. We provide an overview of our problem in Section 3. In Section 4, we describe the stage of preprocessing. Our adaptive block sampling is presented in Section 5. In Section 6, we use COLA to implement our approach. Experimental results are presented in Section 7, followed by conclusions.

2 Related Work

In general, our work in this paper is related to two fields: online aggregation and data sampling. OLA was first introduced in RDBMS [1], which focuses on single-table queries involving "group by" aggregations. The work in [2] improves the approach in [1] by providing the large-sample and deterministic confidence interval computing methods in the case of single-table and multi-table queries. The query processing and estimate algorithms for OLA were studied in the context of joins over multi-tables [3]. A family of join algorithms called ripple joins were presented in [3]. Wu et al. [6] proposed a new OLA system called COSMOS to process multiple aggregation queries efficiently. All the work above is in the context of RDBMS. In fact, these centralized OLA methods or systems can not be extended to MapReduce-based cloud systems straightforward. Hadoop Online Prototye (HOP) [8] is a modified version of the original MapReduce framework, which is proposed to construct a pipeline between Map and Reduce. COLA [11,28] realizes the estimation of confidence interval based on HOP. A Bayesian framework based approach is used to implement OLA over MapReduce [13] based on the open source project Hyracks [9]. The approach's estimation method is complex, which is hard to be implemented in the MapReduce framework. In the work of [15], the authors focus on the optimization for running OLA over MapReduce-based cloud system.

Sampling has a long history in database. There are two levels of sampling unit in the existing sampling techniques: row-level sampling [19,21] and block-level sampling [20,22]. Row-level sampling provides true uniform-randomness, which is the basis of many approximate algorithms. The work in [20] proposes statistical estimators with block-level samplings. Several special sampling approaches for group-by queries are also proposed. A method called Outlier Indexing [23] is proposed to improve sampling-based approximations for aggregation queries. Congressional sampling [24] stratifies the database by considering the set of queries involving all possible combinations of grouping columns and provides general-purpose synopses. Another approach to solve the small group problem has been developed by Babcock et.al. [25] and is called small group sampling. It generates multiple sample tables and selects an adequate subset of them at query evaluation time. Philipp et.al. [26] proposed a novel sampling scheme for constructing memory-bounded group-aware sample synopses. All above sampling approaches for group-by queries are row-level sampling. In the context of online aggregation in the cloud, existing work of OLA over MapReduce adopts random sampling [8,13], and no special sampling techniques have been proposed.

3 Overview

In this section, we firstly formalize our problem that we study in the paper. This paper mainly focuses on online aggregation for single table, so we consider a relation R and queries like

$$\boxed{\text{SELECT } op(exp(t_{ij})),\ col \text{ FROM } R \text{ WHERE } predicate \text{ GROUP BY } col}$$

where *op* is the operation of Sum, other operator (Count, Avg) can be dealt similarly. *exp* is an arithmetic expression of the attributes in R, *predicate* is an arbitrary predicate involving the attributes, and *col* is one or more columns in R. Because all the data are stored in HDFS, the data unit is block while its counterpart in RDBMS is tuple. t_{ij} represents the j-th tuple in block i.

Figure 1 shows the basic architecture of our system. Online aggregation in the cloud is that when users request an aggregation query, system returns an approximate result within the prescribed level of accuracy against random samples. The result is refined as more samples are received. In this way, users can terminate the running queries prematurely if an acceptable estimate arrives quickly. Usually, confidence interval and confidence level are adopted to measure the accuracy of current result. If users do not terminate the query actively, all the data will be scanned, and the processing is just the same as common aggregation query. The whole data processing is implemented in the cloud.

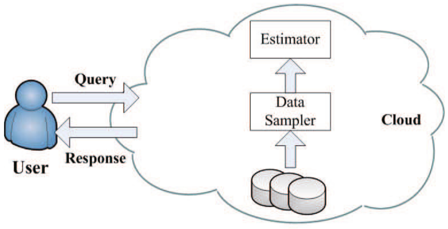


Fig. 1. Basic Architecture

As shown in Figure 1, there are two major issues we should consider when implementing online aggregation with group-by clause in the cloud: (1) How to sample data efficiently and, (2) How to implement the method in the cloud.

We will in turn detail our solutions for the two issues in subsequent sections.

4 Preprocessing

It is extremely difficult to execute the queries online without any assumption or preprocessing. We do not add constraints to the original data, so we need preprocessing to the data. During the stage of preprocessing, two important things should be completed: randomization of data blocks and acquisition of necessary data information. The performance of online aggregation highly depends on the sampling, and the accuracy of sampling depends on the data distribution. If the data are fully random, the result will be good. Unfortunately, this is not always true for real world data. For tuple-level sampling, there are several ways to achieve random sampling from disk. If the data are distributed randomly, we just need sequential access. Otherwise, we must do random disk access to obtain random sampling. For block-level sampling, the case can be worse. We still can not get random sampling by random disk access because the data layout inside the block may be not random. Besides, completely random disk access can be five orders of magnitude slower than sequential access [27]. If we want to ensure samples from different groups can match the data distribution, some additional data information must be obtained. In this section, we introduce one approach to randomize the blocks and obtain the data information simultaneously.

Usually, online aggregation assume the dataset is static, we also reserve the assumption in this paper. Here, static dataset does not mean we never update the dataset. In real-life applications, if the dataset is relatively static or update rarely, we can view it as a static dataset. The cost of preprocessing can be amortized by subsequent massive queries. This stage can be done by one MapReduce job and the basic procedure is as follows:

1) Get current number of different blocks on HDFS from namenode, denoted by N.

2) Initiate the MapReduce job and then Map tasks read data from HDFS.

3) Read value of the key-value pair and record the frequency of different values within each column. The concept of column here is similar to the column of RDBMS. In fact, there is no column in the data block, but different values in one line which are seperated by some predefined characters can be viewed as different columns. Then Map tasks generate a random number between 1 and N, and assign the random number to the header of the key of current key-value pair.

4) Reduce tasks receive the key-value pairs and write them to different blocks according to the header of the key. At the same time, reduce tasks merge the frequency of different values from all the map tasks. These frequencies and corresponding location (i.e. the data block they belong to) will be stored on the master as data information.

This method of data redistribution is easy to be realized and can be used in any MapReduce-based cloud system without any modification. Its performance will be demonstrated in the subsequent experiments.

5 Query Processing

Unlike query processing in RDBMS, OLA must return the estimated result continuously with accuracy guarantees. In this section, we will elaborate the query processing of OLA, which includes a new adaptive block sampling and the estimation of results.

5.1 Adaptive Block Sampling

Uniform random sampling is the most used sampling method of online aggregation. If the data is distributed evenly, random sampling is effective. Unfortunately, in the real world, many datasets are skew and the problem of "small group" is not uncommon. Block sampling is more complex than random sampling because the data within a block may be correlative. The basic processing unit of block sampling is block. If the block is inappropriate to a certain "small group", we still need to scan all the data in that block, this is really wasteful. In this section, we propose a new adaptive block sampling which can avoid above drawbacks.

For a relation R with two columns A, B, let B be the column used for grouping and A be the aggregation attribute. Suppose B can be divided into n groups: $\{b_1, b_2, ..., b_n\}$. Since we have known the frequencies of values in different groups during the stage of preprocessing, it is easy to compute the ratio between these n groups, which can be described as follow:

$$b_1 : b_2 : ... : b_n = r_1 : r_2 : ... : r_n \tag{1}$$

The basic steps of our adaptive sampling is described as follows:

- **step1**: Map tasks read the data blocks sequentially. Data blocks have been randomized during the stage of preprocessing, so read blocks sequentially can get random blocks instead of randomly read;
- **step2**: Read key-value pair within the block one by one, and decide to accept or reject current key-value pair by using the method that is similar to acceptance/rejection sampling. The probability of acceptance is decided by column B. If current key-value pair belongs to group b_i, its probability of acceptance p_i is as follows:
$$p_i = \frac{r_i}{\sum_{i=1}^{n} r_i} \tag{2}$$
- **step3**: Compute the current ratio between different groups in the sample. If a certain group is unseen or its ratio is highly under the normal level, we set the probability of acceptance is 1. This means the group is accepted directly as soon as it appears next time.
- **step4**: Compute the terminal condition. We find it is not necessary to process the entire block to get the approximate answers. We define three parameters α, β and γ. Here α means the ratio of data we have processed within current block, β means the ratio of groups we have not seen yet, and γ means the upper bound of the amount of data which will be processed within a block. If α percent of the data have been processed and still β percent of the groups have not seen, this data block will be discarded; If the data block is not discarded, the processing of current data block will stop in two situations: 1) If all the groups have been seen and their ratios approximately match the true ratios, stop processing current block; 2) If γ percent of the data have been processed, we also stop. α, β and γ can be obtained by multiple tests and will be given in the subsequent experiments.

– **step5**: Repeat step 1 to 4 until users terminate the process actively. If users do not terminate, all the data block will be processed.

5.2 Estimation and Confidence Interval

After the sampling procedure, we can get a sampling S_k with size n_k for a particular group k, and set $exp_k = \sum_{i=1}^{n_k} exp_k(t_{ij})$. The estimation of aggregation result is given by

$$\mu = \frac{N}{n_k} * exp_k \tag{3}$$

According to the previous equation, the final aggregation result can also be considered as the average of Y, where $Y = N * exp_k$. The sampled data are retrieved in random order, so Y are identically distributed and independent to each other. According to the central limited theorem, the average of Y in samples obeys the normal distribution, so we can obtain the half-width interval with specified confidence level 100p%: $\varepsilon_n = z_p \sigma_n / \sqrt{n}$, where z_p is the p-quantile in the standard normal distribution, and σ_n is the standard deviation of n_k varibles in the sample. The final $100p\%$ confidence interval of the aggregation result is $[\mu - \varepsilon_n, \mu + \varepsilon_n]$.

6 Online Aggregation in the Cloud

Cloud is different from RDBMS, and the major problem of online aggregation in the cloud is that naive MapReduce does not support pipeline operations. Several online aggregation systems have tackled the problem of pipeline, e.g., HOP, COLA. We finally choose COLA, which is described in the paper [11], rather than HOP, to implement our method. Our method is easier to be implemented on COLA because it supports more operations for online aggregation. The basic steps of online aggregation with our method is described as follows:

– **step1**: The user sends a query to the master and the master determines the location (i.e. which slaves have relative data) which will involve the query;
– **step2**: Read data from relative slaves and get the initial samples using our adaptive block sampling;
– **step3**: Compute the estimation and confidence interval;
– **step4**: Output the result and continue sampling;
– **step5**: Repeat step 1 to 4 until users terminate the process actively. If users do not terminate, all the data block will be processed.

Our method primarily focuses on the query on a single table, so the implementation only involves one MapReduce job. In the Map function shown in Algorithm 1, tuples of every block are filtered out according to the predicate and transformed into key-value pairs.

Algorithm 1. Map function

```
Input:Object t;
Output:Text key, Text value;
1:if t satisfies the predicate then
2:calculate the probability of acceptance;
3:if accept then
2:    key.set(t.tuple.lang);
3:    value.set(t.tuple.size);
4:end if
5:output.collect(key,value);
```

The Reduce function is executed each time the estimate is invoked, so it is important to make the computing process incremental and make use of the results of the previous Reduce function. The Reduce function is described in Algorithm 2.

Algorithm 2. Reduce function

```
Input:Text key,Iterator(Text)values;
Output:aggregation estimate $\mu$, confidence interval $\varepsilon$;
1://$n_k$: number of tuples processed by the reducer;
2://$sum_i$: sum of the variables in the last iteration;
3://$quadratisum_i$: quadratic sum of the variables in the last iteration
4:while values.hasNext() do
5:    Text it=values.getNext();
6:    $sum$+=it.get_first();
7:    $quadraticsum$+=it.get_second();
8:end
9:sum+=$sum_i$;
10:quadraticsum+=$quadratisum_i$
11:variance=quadraticsum/n-sum*sum/n*n;
12:$\mu$=sum/n;
13:$\varepsilon = z_p sqrt\,(variance)\,/sqrt\,(n)$
```

7 Performance Evaluation

7.1 Experiment Overview

The testbed is established on a cluster of 11 nodes connected by a 1Gbit Ethernet switch. One node serves as master, and the remaining 10 nodes act as slave nodes. Each node has a 2.33GHz quad-core CPU and 7GB of RAM, and the disk size of each node is 1.8TB. We set the block size of HDFS to 64MB.

Naive MapReduce dose not support pipeline operations, so we use COLA, an online aggregation system which is developed based on HOP. We implement our methods described above for COLA, and all the following experiments are conducted on the improved COLA.

In the experiment, we evaluate our approach on a synthetic dataset. We constrcut a relation R which contains three attributes A, B and C. C is used for grouping. We generate about 20GB data and C contains 10 groups. Three of the ten groups are relatively small, and they are defined as "smalll group" in this paper. Table 1 summarizes settings used in the experiments:

Table 1. Settings used in the experiments

Parameter	**Values**
Number of Computer Node	11
Data size	20G
Platform	COLA
Map task num. per Computer Node	4
Reduce task num. per Computer Node	2

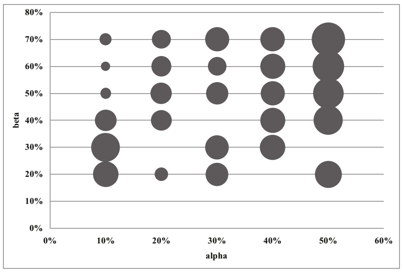


Fig. 2. Parameters: α and β

All the data are stored in HDFS, and we test online aggregation queries of Sum with following example queries Q.

Q= SELECT Sum(A), C FROM R GROUP BY C

In subsquent experiments, we set the confidence level to 95%. The accuracy of estimated aggregation result is measured by *relative_error*, which is computed by following equation:

$$relative\_error = \frac{|estimateValue - actualValue|}{actualValue} \tag{4}$$

7.2 α, β and γ

These three parameters are very important for our sampling. We want to make our samlping faster, so α can not be too large. At the same time, we need to ensure the accuary of the result, so β can not be too small. We set α to 10%, 20%, 30%, 40% and 50% respectively. For each α, we set β to 20%, 30%, 40%, 50%, 60% and 70%, so we have thirty kinds of combination. We record the time when the relative_error is no more than 5%. In order to choose the parameters accurately, one small group in C is used to show the result. Figure 2 illustrates the result. X-axis means α, Y-axis means β. The area of the bubbles in the figure represents the time. The smaller the bubble is, the shorter the time it takes. We

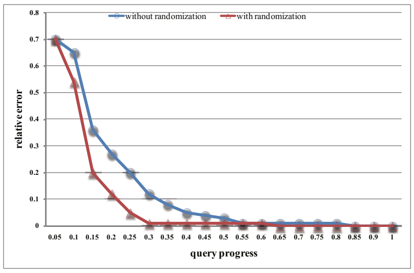
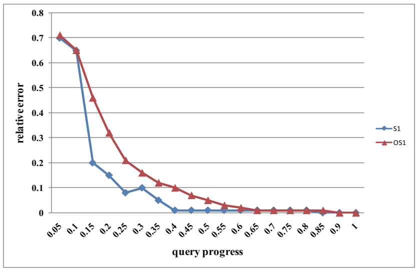
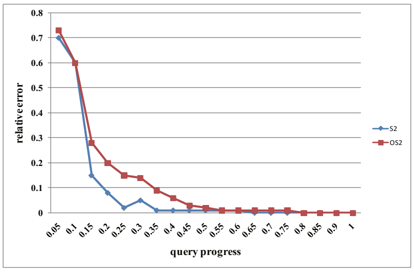


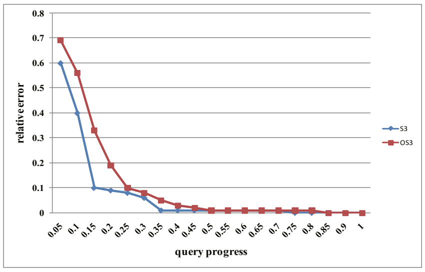
Fig. 3. Effect of Data Randomization



(a) Small Group 1



(b) Small Group 2



(c) Small Group 3

Fig. 4. Query Error of Small Group

can know that when the α is 10% and β is 60%, we can get the shortest running time when the accuracy of relative_error is also satisfied.

γ can not be too small or too large. In our multiple experiments, we found that 55% is a good choice. If γ is smaller than 55%, it is hard to get accurate results. If this value is larger than 55%, longer time will be taken and it is wasteful.

Subsequent experiments all adopt above α, β and γ.

7.3 Effect of Data Randomization

We compare the trends of relative_error between original data and data after randomization. We only choose one large group from the ten groups in C to show the result due to limited space. Figure 3 shows that our method of data randomization really takes effect. We can get the accurate result earlier than the dataset without data randomization.

7.4 Query Error of Small Group

Our method mainly focuses on the problem of small group, so we compare the trends of query error of the three small groups in our experiment. Because there is no work of block sampling for the problem of small group, we just compare our method with simple random sampling. S1, S2 and S3 are the results of small group 1, 2 and 3 by using our adaptive block samping. OS1, OS2 and OS3 are the results of small group 1, 2 and 3 by using simple random sampling. Group 1 is the samllest one of the three groups while group 3 is the largest.

Figure 4 displays the query results of group 1, group 2 and group 3 respectively. From all these figures, our method can get high accuary with less time. We can also find that the samller the group is, the better our method performs.

8 Conclusion

"small group" is a very important problem for online aggregation. Uniform random sampling is not suitable for this problem. We propose a new adaptive block sampling to solve the problem of "small group". This method can adjust the probability of acceptance dynamically and the experiments have shown its effieciency.

Acknowledgments. This research was partially supported by the grants from the Natural Science Foundation of China (No. 61379050,91224008); the National 863 High-tech Program (No. 2013AA013204); Specialized Research Fund for the Doctoral Program of Higher Education(No. 20130004130001), and the Fundamental Research Funds for the Central Universities, and the Research Funds of Renmin University(No. 11XNL010).

References

1. Hellerstein, J.M., Haas, P.J., Wang, H.J.: Online aggregation. In: SIGMOD Conference, pp. 171–182 (1997)
2. Haas, P.J.: Large-sample and deterministic confidence intervals for online aggregation. In: 9th IEEE International Conference on Scientific and Statistical Database Management, pp. 51–62. IEEE Press, New York (1997)
3. Haas, P.J., Hellerstein, J.M.: Ripple Joins for online aggregation. In: SIGMOD Conference, pp. 287–298 (1999)

4. Luo, G., Ellmann, C.J., Haas, P.J., Naughton, J.F.: A scalable hash ripple join algorithm. In: SIGMOD Conference, pp. 252–262 (2002)
5. Jermaine, C., Dobra, A., Arumugam, S., Joshi, S., Pol, A.: A disk-based join with probabilistic guarantees. In: SIGMOD Conference, pp. 563–574 (2005)
6. Wu, S., Ooi, B.C., Tan, K.: Continuous sampling for online aggregation over multiple queries. In: SIGMOD Conference, pp. 651–662 (2010)
7. Wu, S., Jiang, S., Ooi, B.C., Tan, K.: Distributed online aggregation. presented at PVLDB, pp. 443–454 (2009)
8. Condie, T., Conway, N., Alvaro, P., Hellerstein, J.M., Gerth, J., Talbot, J., Elmeleegy, K., Sears, R.: Online aggregation and continuous query support in MapReduce. In: SIGMOD Conference, pp. 1115–1118 (2010)
9. Borkar, V.R., Carey, M.J., Grover, R., Onose, N., Vernica, R.: Hyracks: A flexible and extensible foundation for data-intensive computing. In: ICDE, pp. 1151–1162 (2011)
10. Böse, J.-H., Andrzejak, A., Högqvist, M.: Beyond online aggregation: parallel and incremental data mining with online Map-Reduce. In: 2010 Workshop on Massive Data Analytics on the Cloud, pp. 1–6 (2010)
11. Shi, Y., Meng, X., Wang, F., Gan, Y.: You can stop early with COLA: online processing of aggregate queries in the cloud. In: CIKM, pp. 1223–1232 (2012)
12. Gan, Y., Meng, X., Shi, Y.: COLA: A cloud-based system for online aggregation. In: ICDE, pp. 1368–1371 (2013)
13. Pansare, N., Borkar, V.R., Jermaine, C., Condie, T.: Online aggregation for large mapreduce jobs. In: PVLDB, pp. 1135–1145 (2011)
14. HKalavri, V., Brundza, V., Vlassov, V.: Block sampling: efficient accurate online aggregation in mapreduce. In: CloudCom, vol. (1), pp. 250–257 (2013)
15. Wang, Y., Luo, J., Song, A., Dong, F.: Partition-Based Online Aggregation with Shared Sampling in the Cloud. J. Comput. Sci. Technol., 989–1011 (2013)
16. Qin, C., Rusu, F.: Parallel online aggregation in action. In: SSDBM, p. 46 (2013)
17. Wang, Y., Luo, J., Song, A., Dong, F.: OATS: online aggregation with two-level sharing strategy in cloud. In: Distributed and Parallel Databases, pp. 1–39 (2014)
18. Wu, M., Jermaine, C.: Guessing the extreme values in a data set: a bayesian method and its applications. VLDB J., 571–597 (2009)
19. Antoshenkov, G.: Random sampling from pseudo-ranked B+ trees. In: VLDB, pp. 375–382 (1992)
20. Chaudhuri, S., Das, G., Srivastava, U.: Effective use of block-level sampling in statistics estimation. In: SIGMOD Conference, pp. 287–298 (2004)
21. Olken, F., Rotem, D.: Random sampling from database files: a survey. In: SSDBM, pp. 92–111 (1990)
22. Haas, P.J., Koenig, C.: A Bi-level bernoulli scheme for database sampling. In: SIGMOD Conference, pp. 275–286 (2004)
23. Chaudhuri, S., Das, G., Datar, M., Motwani, R., Narasayya, V.R.: Overcoming limitations of sampling for aggregation queries. In: ICDE, pp. 534–542 (2001)
24. Acharya, S., Gibbons, P.B., Poosala, V.: Congressional samples for approximate answering of group-by queries. In: SIGMOD Conference, pp. 487–498 (2000)
25. Babcock, B., Chaudhuri, S., Das, G.: Dynamic sample selection for approximate query processing. In: SIGMOD Conference, pp. 539–550 (2003)
26. Rsch, P., Lehner, W.: Sample synopses for approximate answering of group-by queries. In: EDBT, pp. 403–414 (2009)
27. Jacobs, A.: The pathologies of big data. Commun. ACM, 36–44 (2009)
28. COLA, http://idke.ruc.edu.cn/COLA/