Efficient Skew Handling in Online Aggregation in the Cloud

Xiang Ci
School of Information
Renmin University of China
Beijing, China
Email: cixiang@ruc.edu.cn

Fengming Wang
School of Information
Renmin University of China
Beijing, China
Email: rucwfm1991@ruc.edu.cn

Yantao Gan
School of Information
Renmin University of China
Beijing, China
Email: ganyantao@ruc.edu.cn

Xiaofeng Meng
School of Information
Renmin University of China
Beijing, China
Email: xfmeng@ruc.edu.cn

***Abstract*—As the development of social network, mobile Internet, etc., an increasing amount of data are being generated, which beyonds 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, since more and more data analysis applications are migrated to the cloud, online aggregation in the cloud has also drawn more attention. The problem of data skew can greatly impact the results of online aggregation in the cloud. In fact, there exist two special types of data skew in online aggregation in the cloud. In this paper, we propose two methods to deal with the two types of data skew respectively. We implement our methods in a cloud online aggregation system called COLA and the experimental results demonstrate our methods can remarkably eliminate negative effect of data skew and get better results.**

I. Introduction

Big data has brought great challenges to traditional data management, as the growth of sheer volume of data, techniques of data management and analytics based on the relational database system cannot work in many fields. Value of stored data can only be explored by data analysis. Extracting knowledge from data is usually an interactive process, with a user issuing a query, seeing the result, and using the result to formulate the next query, in an iterative fashion. During this procedure, users are more concerned about the response time rather than data accuracy. They can accept a little sacrifice of data accuracy in return for shorter response time.

Aggregation query of massive datasets has been noticed by researchers of relational database management system, and online aggregation (OLA) was proposed to cope with this problem. The basic idea behind online aggregation is to estimate the result by sampling data and the approximate answer should be given certain accuracy guarantee. Generally, we use confidence to measure the accuracy. In recent years, as the development of cloud computing and big data, online aggregation in the cloud has attracted more attention. Although OLA is very suitable for cloud environment, there are still some limitations constraining its extensive use, and one of the important considerations is data skew. Data skew is not a new issue, but it has brought new challenges when we implement OLA in the cloud. Our contributions include:

– We point out two special types of data skew in OLA and analyze how they influence the results of OLA.
– We propose two approaches to solve the problem of data skew in OLA.
– We implement our method in COLA, and conduct extensive experiments to demonstrate that our method can deliver reasonable precise online estimates within a time period much shorter than that used to produce exact answers.

The remainder of the paper is organized as follows. In Section 2, we present the related work. In Section 3, we describe material required for the remainder of the paper. In Section 4, two methods are proposed to handle the problem described in section 3. We use COLA to implement the proposed methods, and the experiment results are presented in Section 5, followed by conclusions.

II. Related Work

Online aggregation was proposed as one commonly-used approximate query processing technique to provide a time-accuracy trade off for aggregation queries. It was first introduced in the field of relational database management system [1], which focuses on single-table queries involving "group by" aggregations. Work [6] proposed a new OLA system called COSMOS to process multiple aggregation queries efficiently. COSMOS organizes queries into a dissemination graph to exploit the dependencies across queries, and the partial answers can be reused by the linked queries. All the work above is in the context of traditional databases. Nowadays, online aggregation research is renewed in the context of cloud computing, and some studies have been conducted based on MapReduce. 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 so that the reduce task could start immediately as long as any Map output is generated. HOP can provide the original snapshots of the MapReduce jobs at data dependent intervals, and it supports OLA by scaling up the snapshots with the job progress without any confidence bounds of the query estimate. COLA [9], [10], [23] realizes the estimation of confidence interval based on the HOP. COLA allows different task to adopt different data granularity and provides progressive approximate answers for both single tables and joined multiple tables.

Data is not always so uniform in reality, so the work in [12] focuses on the problem of data skew, and a preprocessing stage is introduced. This stage can make the skewed data into random data. Work [15] improves performance of OLA for skewed data distribution by efficient pruning of unneeded data due to the partition and shuffle strategies. Work [24] addressed the skewed issue that affects the OLA performance by proposing a keep-order algorithm to make sure that intermediate results delivered to downstream operators are consumed in a statistically random fashion.

III. Preliminaries

We start by introducing material required in this paper. This includes two parts: problem definition and introduction of data skew of OLA in the cloud.

A. Problem Definition

In this section, we formalize our problem. This paper mainly focuses on online aggregation for single table, so consider a relation *R* and queries of the form as follows:

SELECT *op*(*exp*(t_{ij})), *col* FROM *R* WHERE *predicate* GROUP BY *col*

where *op* is the operation of Sum, and 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, so the data unit is block while its counterpart in RDBMS is tuple. t_{ij} represents the j-th tuple in block i.

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 the unbiased random samples, and 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. We must guarantee above query can still return accurate results when facing the problem of data skew. Figure 1 shows the basic architecture of COLA, COLA is a cloud-based online aggregation system.The system can provide progressive approximate aggregate answers in large-scale, distributed MapReduce environment. Currently, COLA supports OLA for both single table and multiple joined tables. In this paper, we focus on single-table queries involving "GROUP BY" aggregations.

B. Data Skew of OLA in the Cloud

Data skew is not a new problem for RDBMS, but there are some special issues when we implement OLA in the Cloud. These special issues are mainly caused by the data organization in the cloud and the sampling-based processing of OLA.

1) Skewed Data Distribution: Uneven data distribution can lead to data skew, so lots of methods have been proposed to handle the problem. Among these methods, the normally used size-aware partition manner seems to be a good candidate for the implementation of OLA in the cloud, in which the block contains the same amount of data, thus reducing possible skew of tasks when processing in parallel and improving the performance. However, this data partition is not always effective for OLA. Because OLA is a sampling-based method to obtain approximate results from a subset of data rather than the exact one against the whole dataset, the sampling efficiency becomes the major factor that affects the performance especially when the data distribution is skewed. Given a dataset, a majority of blocks may not contain any tuples that satisfy the query predicate (called relevant tuples) or only contain a small amount of such tuples. For the blocks containing fewer (or none) relevant tuples, the probability of drawing sufficient relevant tuples by sampling may be relatively small (or none), and there will be few or no relevant tuples in the sample during the initial stage of OLA, leading to larger error for the accuracy estimation. Therefore the commonly size-aware data partition cannot solve the problem of data skew when it applies to OLA in the cloud.

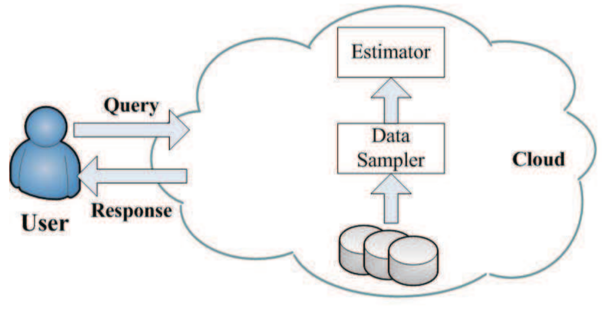


Fig. 1: COLA Architecture

2) Skewed Intermediate Results: The basic idea behind OLA is to estimate the result by sampling data and the approximate answer should be given some accuracy guarantees. The results of OLA highly depend on the sampling and the accuracy of sampling can only be ensured when the dataset is random. One way is to associate a pseudo-random number with each processing unit (which typically refers to a tuple in traditional database). Then, a sorting algorithm is used on the tuples to order them based upon the associated pseudo-random numbers. Thus, input data can be viewed as if it is clustered in a statistically random fashion on disk such that a sequential scan already returns the data in a random order. In this case, there is absolutely no correlation between the ordering of the tuples on disk and the contents of the tuples. Even the slightest correlation can invalidate the statistical properties of OLA algorithm, leading to inaccurate estimates and confidence bounds. In large-scale, distributed environment, the basic unit of data storage and processing is a block[16], which may contain tens or hundreds of megabytes of data. In this type of environment, out of consideration for efficiency, we take data block as the random sampling unit, and conduct the statistical computing based on the aggregate value associated with each block. In this section, we describe our observations of how data skew can yield biases in distributed environment and lead to inaccurate estimates.

Haas and Hellerstein's derivation assumes a with-replacement sampling (visualized in Figure 2(a)). In the application domain imagined for OLA, this is reasonable since it was assumed that the computation will be terminated after only a rough estimate of the eventual answer has been obtained.

This means that the computation probably will end before a significant fraction of input data has been processed. In such a situation where sample size is "small enough", the difference between sampling with replacement and sampling without replacement is negligible. As a result, in Haas and Hellerstein's analysis, the samples can be viewed as a sequence of independent and identically distributed (i.i.d.) random variables. Thus, we can bring to bear the tools of statistical estimation theory.

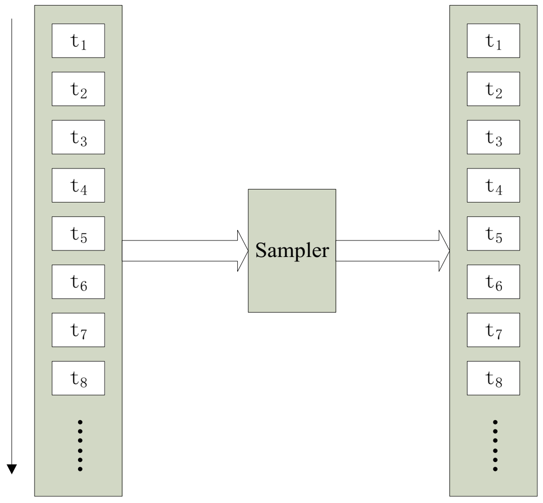
We found that in distributed MapReduce environment, this i.i.d. guarantee no longer holds, because of the impact of data skew on upstream operators' runtime. Take a look into Hadoop, usually, a fixed amount of bytes are assigned to each map task, how unbalance can arise in the runtime of map phase though? [14] confirms that the input data size alone is not a good indicator of task runtime. In real-world Hadoop clusters, there are many causes leading to such unbalanced runtime in map phase, including:

– Instability inherent in system: hardware malfunction, resource contention, cluster conditions, misconfiguration etc.
– Occasionality of runtime environment: locality scheduling (take Hadoop, that is, some blocks may be scheduled to the local machine, some are scheduled to a machine in the same rack, and some are in a distant rack.), etc.

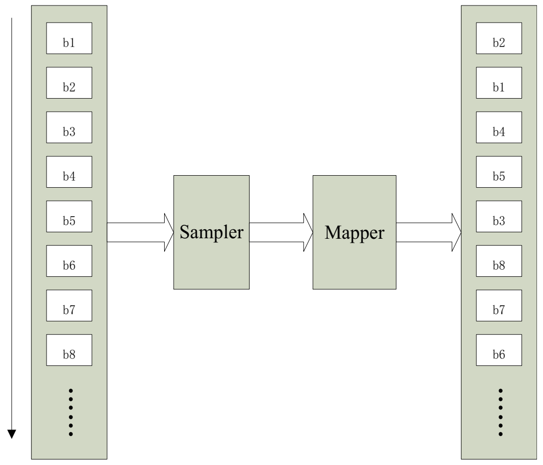
Such variation in the runtime would break the randomness on which statistical tools are based. That is, while mappers access input blocks randomly, the map outputs consumed by reducer are not kept in that random order as shown in Figure 2(b). Since the reducer is the operator that produces the aggregates, what really matters is the processing order in reducer. Unfortunately, in this type of environment, this order is somehow correlated with the aggregate value of each block. The reasoning is intuitive: those blocks with lower selectivity could have a lot of intermediate results, and take longer to process, which we called as the "superchunks". And such a correlation may exist, superchunk with a lot of valid data is more likely to have big aggregates. At any particular point, we look into the set of blocks that have actually completed processing, non-superchunks appear with higher probability. When non-superchunks appear "too quickly", the estimator interprets this as a sign the final aggregate answer is smaller than it truly is, causing the estimator to under-estimate. The sampling unit of database is tuple, so we can process the data records one by one. The order of sampled data is the same as original random sequence, and the sampled data can be viewed as random sequence. The sampling unit of OLA is block, and one block may contain thousands of tuples. The original sequence can be viewed as random while the sampled data can not, because the order of sampled data may be not the same as original sequence. The processing time of a block is decided by how many relevant tuples it contains for the current query. Those blocks which contain less relevant tuples would be received early by downstream operators. This would destroy the randomness of data and cause the estimation unprecise.

IV. Skew Handling in OLA

To address above two problems of data skew of OLA in the cloud, we propose two different approaches respectively in this section.



(a) Database Sampling



(b) OLA Sampling

Fig. 2: Sampling Process

A. Data Filter

For the first type of data skew, the key point is to locate relevant blocks, which contain more relevant tuples for the query. If we can filter out irrelevant blocks at the very beginning, the overall processing time will be reduced. We utilize Hilbert curve to map the data range of each block into one dimension. In order to promote the efficiency, the implementation of Hilbert curve is loaded into the memory of namenode when the system starts. Once the system receives the user's query, the relevant blocks will be selected very soon.

B. Re-randomness of Intermediate Results

As illustrated above, the skewed intermediate results is caused by the destruction of original randomness. In the work [24], authors tried to force the sampled data obey the original order. This is a naive solution. If a block contains lots of relevant tuples, it may be received by the downstream operator very late. Other blocks will wait for the block for a long time and the overall processing time will extend under this situation. We solve this problem by a very tricky approach:

re-random the intermediate results. The processing of re-randomness must be online, just like the processing of stream. This is because intermediate results of OLA are continuous and we cannot wait for all the results ready. Finally, we choose Acceptance/Rejection sampling (A/R sampling) [18] to re-random the intermediate results. The key point of A/R sampling is the choice of ***acceptance probability*** of every block. Suppose there are totally N blocks of the input file *F*, and the ***acceptance probability*** of block B_i is

$$\alpha_i = \frac{b_i}{b_{\max} * \beta}$$

Where b_i is the amount of relevant tuples contained by block B_i, b_{max} is the maximum amount of relevant tuples contained by all the blocks of input file *F*, and β is the adjustment factor. Usually, we do not need β for tuple-level sampling, but for block-level sampling, very few blocks may contain a very large part of the relevant tuples and most blocks may only hold very few relevant tuples. In this situation, if we do not introduce the adjustment factor, the acceptance probability of most blocks will be very low and will not be accepted by downstream operators. The user must wait for a long time to see the initial result which is very bad for OLA. We can prove the ***probability of acceptance***(do not confuse it with the ***acceptance probability***) of every block is equal to *1/N*.

PROOF. We use $\mathrm{p^i_{map}}$ represents the sampling probability of data block $\mathrm{B_i}$ in Map phase, so $\mathrm{p^i_{map}} = 1/\mathrm{N}$. When we want to estimate the result during Reduce phase, the output of $\mathrm{B_i}$ in Map phase is inversely proportional to the amount of valid tuples in $\mathrm{B_i}$. Suppose the amount of valid tuples in $\mathrm{B_i}$ is $\mathrm{b_i}$, and the maximum value of all $\mathrm{b_i}$ is $\mathrm{b_{max}}$. In order to avoid over correction, adjustment factor θ ($0 \leq \theta \leq 1$) is introduced. Now the output of $\mathrm{B_i}$ in Map phase is received by Reduce phase with the probability of $\mathrm{p^i_{red}} = \frac{\mathrm{b_{max}}}{\mathrm{b_i}} * \theta$. Set α_i to be the acceptance probability of $\mathrm{B_i}$, then

$$\alpha_\mathrm{i} = \frac{\mathrm{b_i}}{\mathrm{b_{max}} * \theta} \tag{1}$$

Then the probability of acceptance is

$$\mathrm{p_i} = \mathrm{p^i_{map}} * \mathrm{p^i_{red}} * \alpha_\mathrm{i} = 1/\mathrm{N} \tag{2}$$

The new sampling processing is shown in Figure 3.

V. Performance evaluation

A. Experiment Overview

The testbed is established on a cluster of 11 nodes connected by a 1Gbit Ethernet switch. One node serves as the namenode of HDFS and jobtracker of MapReduce, and the remaining 10 nodes act as slave nodes. Each node has a 2.33G 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. We implement our solutions on COLA, and all the following experiments are conducted on the improved COLA. In the experiment, we analyze the page traffic statistics of Wikipedia hits log. The dataset we use contains 7 months of hourly pageview statistics for all articles in Wikipedia, with 320GB of compressed data (1TB uncompressed). Table 1 summarizes settings used in the experiments:

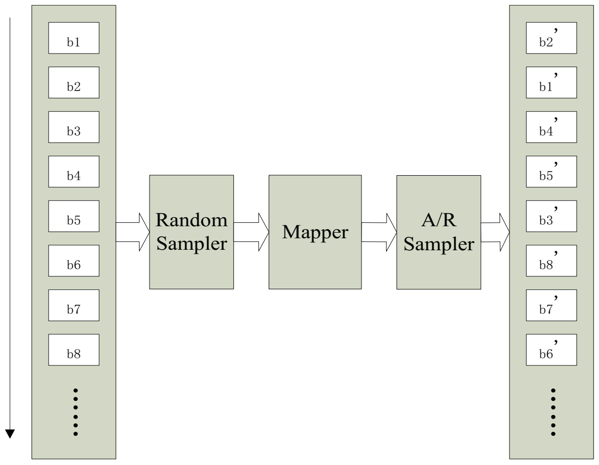


Fig. 3: A/R Sampling

TABLE I: Settings used in the experiments

Parameter	Values
Number of Computer Node	11
Data type	page view statistics
Data size	100G
Platform	COLA
Map task num. per Computer Node	4
Reduce task num. per Computer Node	2

All the data are stored in HDFS. We test online aggregation queries with example queries shown next.

```
Q= SELECT Sum(pageviews), language FROM visit_log
GROUP BY language
```

B. Accuracy of the Estimator

Table II shows the different confidence intervals after a portion of the task has completed: 2%, 4%, 6%, and so on. Here, "Default" stands for the original method while "New" stands for the data filter used in this paper.

TABLE II: Relative Confidence Interval

	2%	4%	6%	8%	10%
Default	0.168	0.124	0.115	0.106	0.072
New	0.147	0.109	0.084	0.062	0.041

From table II, we can find that our method can get more accurate result within fewer time, because our approach involves more relevant blocks. In the first version, random sampler is turned off, so blocks are scheduled in physical order (denoted as non-randomized). The first point to note is that without randomization, estimate results are totally useless. The severe bias in Figure 4 certifies this: the fraction of the time that the final query result is not within the 95% confidence bounds is huge, almost reaching 77.8%. So the estimate result and confidence interval computed without sampling is of no statistical significance. With randomization, the confidence bounds from Figure 5 seem remarkably accurate. most reporting estimates were correct in the sense that the interval did in fact contain the exact answer. The careful reader will immediately notice that the randomized estimator typically under-estimates, which is the expected behavior given that "non-superchunks" appear with higher probability.

We then ran three versions (randomized, keep-order and accept-reject) on the dataset, and the estimate and confidence interval were tracked throughout execution. Figure 5 shows the online estimates for the English language at various query progress percentages. We plot the upper and lower limit of the confidence interval, as well as the current estimate over time. Our technique successfully improves the estimation, as expected that the estimate becomes more accurate and stable as it processes more data. Note in Figure 5 that the value of the estimate begins approaching the true value at the first beginning, and stay very close to the true value for the remaining time of the experiment. For the accept-reject version, the average relative error of all the estimate results is 0.53%, which is 64% lower than the randomized version with the average relative error being 1.46%. At the time, the accept-reject version is also more accurate than keep-order version.

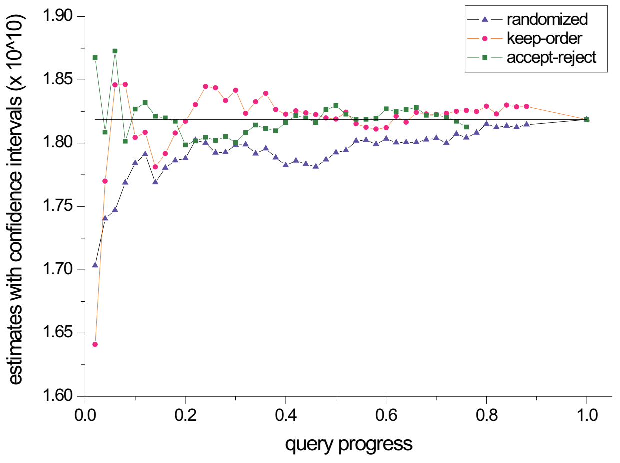


Fig. 5: Estimates with confidence intervals over time

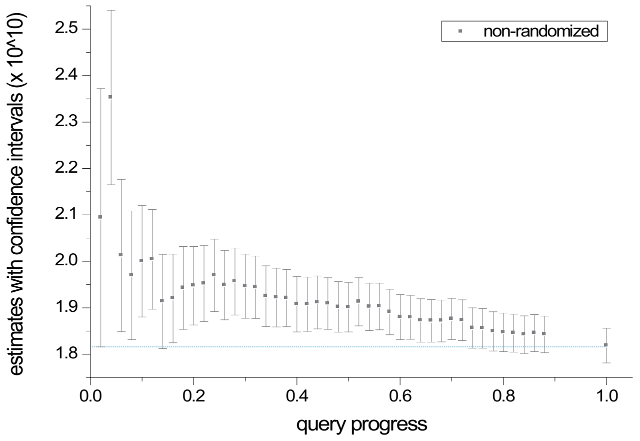


Fig. 4: Non-randomized

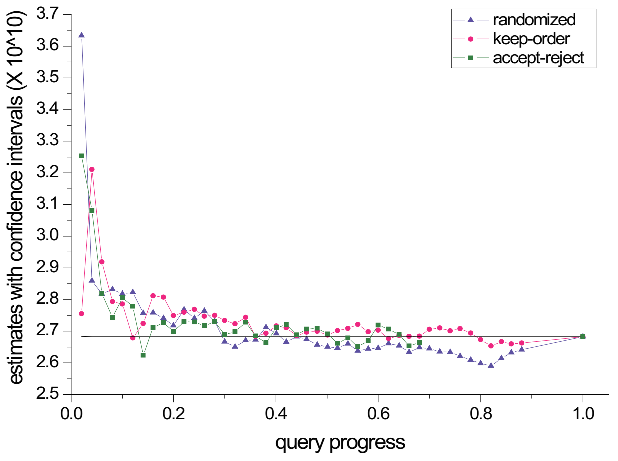


Fig. 6: C=0.0

C. *Impact of Data Skewness*

In this section, we study the effect of data skew. Six sets of datasets were created, using Zipf parameter C in the list (0, 0.2, 0.4, 0.6, 0.8, 1.0). Each tuple has a length of 40 bytes, consisting of language, pagename, pageviews and pagesize just like the Wikipedia dataset. The pageviews is a randomly generated 4-byte integer between 1 and 100. We generate different selectivity in blocks by setting the pagesize attribute to different values. Each dataset set contains 1033 blocks. For each of the six datasets having Zipf parameters, we ran three versions (randomized, keep-order and accept-reject) to completion, and the current estimate and confidence interval were tracked throughout execution. Due to space constraints, we describe only three sets of results having parameters (0, 0.6, 1.0) for English language.

We note that the confidence bounds were tighter when data is less skewed, as observed in Figure 6 - Figure 8. For the randomized version, the average relative interval is 0.850%, 1.42%, 1.55% respectively with Zipf parameter being 0.0, 0.4, 1.0, the keep-order version is 0.853%, 1.36%, 1.53%. Similarly, the accept-reject version is 0.857%, 1.39%, 1.53%. Actually, this is because, the less skewed dataset contains more tuples satisfying the selection predicate. For example, consider the case of the 1.0 Zipf coefficient:in the "biggest" block, the amount of valid tuples is proportional to 1/1, the second "biggest" one is proportional to 1/2, the third is proportional to 1/3, and so on. In our experiment, the fractions of valid data were set to 100%, 50%, 33% and so on. While the dataset having 0.0 Zipf coefficient contains blocks in which all tuples satisfy the selection predicate, in the dataset having 1.0 Zipf coefficient, some blocks contains 100% valid tuples, some contains 50% valid tuples, and so on. As a result, when data is less skewed, there are more tuples satisfying the selection predicate, leading to much tighter confidence bounds. From figure 6 - 8, although the estimation of our method is not

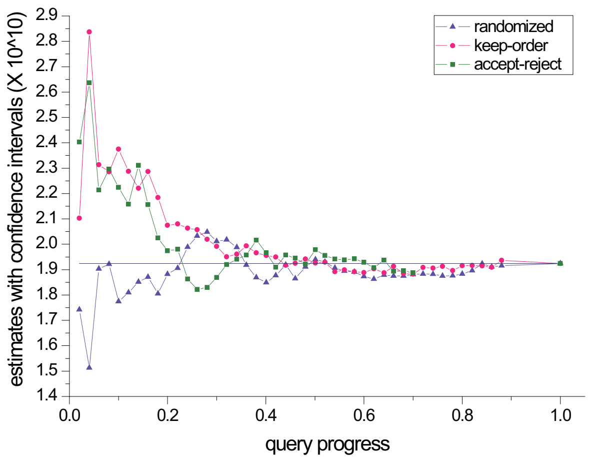


Fig. 7: C=0.6

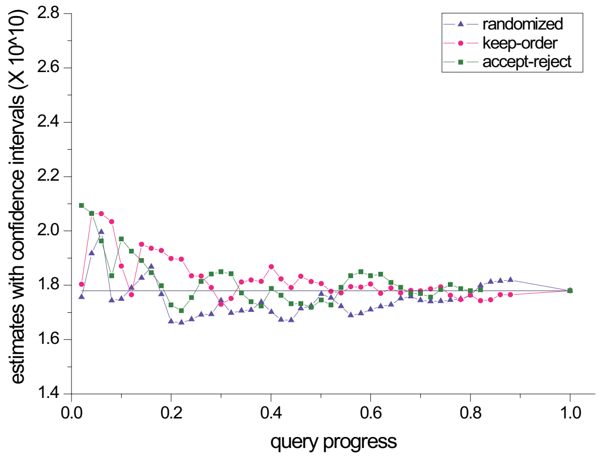


Fig. 8: C=1.0

always better than the other two, the update of confidence interval is more stable than the other two. At the same time, our method can get desired accuracy faster.

VI. Conclusions

In a large-scale, distributed MapReduce environment, online aggregation should appear more and more attractive relative to their o2ine counterparts, given that it can save cost in the cloud by permitting early termination of queries when the approximate answer is sufficiently precise. Incorporating online aggregation into a MapReduce engine, however, raises questions regarding the statistical guarantees in the presence of data skew. In this article, we analyze how biases can arise when estimating aggregates on skewed data in a distributed environment.We propose a re-randomness method, targeting eliminating such biases. The experimental results demonstrate the efficiency of our proposal.

Acknowledgment

This research was partially supported by the grants from the Natural Science Foundation of China (No. 61532016, 61532010, 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(1997)171-182

[2] Haas, Peter 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(1999)287-298

[4] Luo, G., Ellmann, C.J., Haas, P.J., Naughton, J.F.: A scalable hash ripple join algorithm. ;In SIGMOD Conference(2002)252-262

[5] Jermaine, C., Dobra, A., Arumugam, S., Joshi, S., Pol, A.: A Disk-Based Join With Probabilistic Guarantees. ;In SIGMOD Conference(2005)563-574

[6] Wu, S., Ooi, B.C., Tan, K.: Continuous sampling for online aggregation over multiple queries. ;In SIGMOD Conference(2010)651-662

[7] S. Wu, S. Jiang, B.C. Ooi, and K. Tan, ”Distributed Online Aggregation”, ;presented at PVLDB, 2009, pp.443-454.

[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(2010)1115-1118

[9] Shi, Y., Meng, X., Wang, F., Gan, Y.: You can stop early with COLA: online processing of aggregate queries in the cloud. ;In CIKM(2012)1223-1232

[10] Gan, Y., Meng, X., Shi, Y.: COLA: A cloud-based system for online aggregation. ;In ICDE(2013)1368-1371

[11] Pansare, N., Borkar, V.R., Jermaine, C., Condie, T.: Online Aggregation for Large MapReduce Jobs. ;PVLDB(2011)1135-1145

[12] HKalavri, V., Brundza, V., Vlassov, V.: Block Sampling: Efficient Accurate Online Aggregation in MapReduce. ;In CloudCom (1)(2013)250-257

[13] Wang, Y., Luo, J., Song, A., Dong, F.: Partition-Based Online Aggregation with Shared Sampling in the Cloud. ;J. Comput. Sci. Technol.(2013)989-1011

[14] Qin, C., Rusu, F.: Parallel online aggregation in action. ;In SSDBM(2013)46-46

[15] Wang, Y., Luo, J., Song, A., Dong, F.: OATS: online aggregation with two-level sharing strategy in cloud. ;Distributed and Parallel Databases.(2014)1-39

[16] Antoshenkov, G.: Random Sampling from Pseudo-Ranked B+ Trees. ;In VLDB(1992)375-382

[17] Chaudhuri, S., Das, G., Srivastava, U.: Effective Use of Block-Level Sampling in Statistics Estimation. ;In SIGMOD Conference(2004)287-298

[18] Olken, F., Rotem, D.: Random Sampling from Database Files: A Survey. ;In SSDBM(1990)92-111

[19] Haas, P.J., Koenig, C.: A Bi-Level Bernoulli Scheme for Database Sampling. ;In SIGMOD Conference(2004)275-286

[20] Chaudhuri, S., Das, G., Datar, M., Motwani, R., Narasayya, V.R.: Overcoming Limitations of Sampling for Aggregation Queries. ;In ICDE(2001)534-542

[21] Acharya, S., Gibbons, P.B., Poosala, V.: Congressional Samples for Approximate Answering of Group-By Queries. ;In SIGMOD Conference(2000)487-498

[22] Babcock, B., Chaudhuri, S., Das, G.: Dynamic Sample Selection for Approximate Query Processing. ;In SIGMOD Conference(2003)539-550

[23] COLA, http://idke.ruc.edu.cn/COLA/

[24] Gan Y, Meng X, Shi Y. Processing online aggregation on skewed data in mapreduce[C]//Proceedings of the fifth international workshop on Cloud data management. ACM, 2013: 3-10.