A Sampling-based Hybrid Approximate Query Processing System in the Cloud

Yuxiang Wang, Junzhou Luo, Aibo Song, Fang Dong
School of Computer Science & Engineering, Southeast University, Nanjing, P.R.China
Email: {lsswyx,jluo,absong,fdong}@seu.edu.cn

***Abstract*—Sampling-based approximate query processing method provides the way, in which the users can save their time and resources for 'Big Data' analytical applications, if the estimated results can satisfy the accuracy expectation earlier before a long wait for the final accurate results. Online aggregation (OLA) is such an attractive technology to respond aggregation queries by calculating approximate results with the confidence interval getting tighter over time. It has been built into the MapReuduce-based cloud system for big data analytics, which allows users to monitor the query progress and save money by killing the computation earlier once sufficient accuracy has been obtained. Unfortunately, there exists a major obstacle that is the estimation failure of OLA affects the OLA performance, which is resulted from the biased sample set that violates the unbiased assumption of OLA sampling. To handle this problem, we first propose a hybrid approximate query processing model to improve the overall OLA performance, where a dynamic scheme switching mechanism is deliberately designed to switch unpromising OLA queries into the bootstrap scheme for further processing, avoiding the whole dataset scanning resulted from the OLA estimation failure. In addition, we also present a progressive estimation method to reduce the false positive ratio of our dynamic scheme switching mechanism. Moreover, we have implemented our hybrid approximate query processing system in Hadoop, and conducted extensive experiments on the TPC-H benchmark for skewed data distribution. Our results demonstrate that our hybrid system can produce acceptable approximate results within a time period one order of magnitude shorter compared to the original OLA over Hadoop.**

I. INTRODUCTION

In today's fast-paced data-driven business environment, dealing with a tremendous amount of data to derive the latent useful information quickly represents a key desideratum for 'Big Data Analytics'[11]. To perform analytics and deliver exact results on both MapReduce-based systems and parallel databases can be computationally expensive and resource intensive due to the massive volumes of data involved, although they claim 'big data' as their forte. Also the overloaded systems and high delays are incompatible with a good user experience, while the early approximate results that are accurate enough are often of much greater value to users than tardy exact results[12].

One effective technique to handle this performance issue is *online aggregation* (OLA)[10], which aims to give response to large-scale aggregation queries with a statistically valid estimate to the final result earlier (making a tradeoff between time and accuracy). OLA has been studied for RDBMS (relational database management system), streaming data management system and P2P environment[8], [9], [10], [13], [21], [22] and it now emerges as a new research area for cloud environment (especially are the MapReduce-oriented cloud systems) [3], [4], [14], [17], [20], [19]. The basic idea behind OLA is to compute an approximate result against the random samples and refine the result as more samples are received. In this way, users can grasp the overall progress of the running queries and terminate them prematurely if an acceptable result can be arrived at quickly.

In order to obtain the accurate enough approximate results, there is a major assumption needs to be guaranteed first that is the samples collected for OLA processing must be unbiased[10]. If such unbiased assumption cannot be well guaranteed, then the accuracy approximate result cannot be obtained earlier finally. Given a set of unbiased samples, it has two important characteristics: (1) each tuple of the sample must be selected with the same probability, and (2) the sample set must be able to represent the whole population, which implies the good sample quality. However, we note that such two characteristics cannot be always guaranteed during the OLA processing. For the original OLA scheme, a set of random samples are collected without replacement. When the sample size is "small enough", the random sample set can be viewed as the uniform sample set generated from the sampling with replacement[10], then the characteristic (1) can be satisfied easily in the initial stage of OLA, while the characteristic (2) cannot be easily guaranteed for such "small enough" sample size especially when the data distribution is much more skewed[21], [23], **leading to a higher estimated error** which triggers another iteration to refined the result with the expanded sample size. With the continuous OLA processing, another problem emerges along with the enlarged sample set, that is the samples drawn without replacement can no longer be viewed as the good simulation to the one with replacement if the sample size is getting larger to a certain extent[10]. Then the characteristic (1) is violated and cannot be satisfied again for any larger sample set, **leading to the permanent estimate failure**. And all the failure OLA queries will scan the whole dataset to generate the exact results, seriously extending the execution time. Our own preliminary study clearly indicates that the estimate failure issue of OLA exists in practice obviously. Figure1 shows the required iterations for 1000 random aggregation queries, which are processed by the optimized OLA scheme called COSMOS[23] on 10G default TPC-H dataset (uniform data distribution). We note that probably more than 86% of the queries cannot be completed within 20 iterations due to the poor sample quality (the characteristic (2) cannot be easily satisfied during the initial stage of the OLA). Then the more samples are collected for further computation, so that probably 55% of the queries are completed within 50∼80 iterations and 6% of the queries are finished around

0190-3918/14 $31.00 © 2014 IEEE
DOI 10.1109/ICPP.2014.38

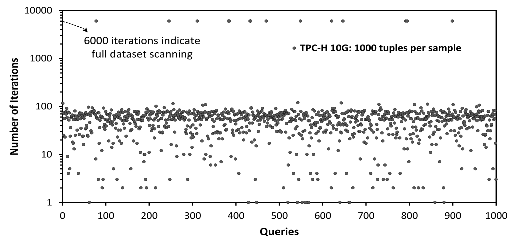

Fig. 1. Preliminary Study of OLA Processing.

100 iterations. Moreover, there are probably 1% of the queries cannot obtain the acceptable results and have to scan the whole dataset due to the estimate failure of OLA (the characteristic (1) is violated for much larger sample set). **Unfortunately, such failed 1% queries occupied probably 53% of the overall OLA execution time, significantly affecting the overall OLA performance**.

Towards to the performance issue resulted from the estimate failure of OLA, the objectives of this paper can be summarized as to (1) find an effective way to deal with the OLA failure for OLA performance improvement, (2) deploy our solution onto the popular cloud environment such as Hadoop[1] to make it more scalable and pervasive.

For the first objective, we propose a hybrid approximate query processing model combined with another resampling-based estimate theory called *bootstrap*[18], which can obtain the estimated results stably by performing resampling with replacement but with more time consumption than the original OLA scheme[12]. There also exist other sampling techniques, such as the jackknife[7], which performs resampling without replacement. In this paper we choose bootstrap technique as the substitution of original OLA method when the estimation fails because of its generality and accuracy[16]. Then we present a dynamic switching strategy, which is the key of our hybrid system, to switch the unpromising OLA query into the bootstrap processing model immediately when the system determines the OLA query has the higher probability of failure, avoiding the unnecessary whole data scanning. Note that the switching operation should be conducted neither too aggressively nor too conservatively. This is because the aggressive policy may lead to the false positive, resulting in more unnecessary bootstrap processing cost. While the conservative policy may lead to more unpromising OLA iterations. In order to make our switching mechanism much more effective, we derive a probabilistic model to depict the OLA estimation failure possibility, and adopt a progressive estimation to reduce the false positive ratio for switching.

While in the case of the second objective, we implemented our hybrid approximate query processing system based on the HOP[1] (Hadoop Online Prototype)[5] to support both single relation query and multi-relations query, and conducted extensive experiments to demonstrate the efficiency and effectiveness of our hybrid system.

The main contributions of this paper includes:

- We propose a hybrid approximate query processing architecture for online processing of aggregate queries in the cloud, which can eliminate the effect of OLA estimation failure effectively.
- We derive a probabilistic model to depict the OLA estimation failure possibility and then exploit a scheme switching mechanism based on such probabilistic model to switch the unpromising OLA queries into the bootstrap scheme dynamically.
- We propose a progressive estimation method to further reduce the probability of false positive during dynamic scheme switching.
- We deploy our hybrid approximate query processing system based on HOP in SEUCloud[2], and conduct extensive experiments to demonstrate that our hybrid system can deliver reasonable precise online estimates within a time period one order of magnitude shorter compared to the original OLA scheme.

The remainder of this paper is organized as follows. In Section II, we give an overview of our hybrid system. In Section III, we propose the dynamic switching mechanism. Section IV describes the implementation details of our hybrid system over MapReduce. And in Section V, we introduce the experimental setup and report results of the experimental evaluation. Finally, we review related work in Section VI and conclude in Section VII.

II. Overview of The Hybrid System

This section describes the overall architecture of our hybrid approximate query processing system and gives a background on the OLA and bootstrap technology.

Figure2 illustrates the architecture of our hybrid system over Hadoop environment, which comprises three major components: (1) **OLA scheme**. The random sample set S is collected from HDFS by the sampling stage without replacement, and is delivered to the OLA scheme for processing. Since the sample size is "small enough" in the initial stage, the samples can be viewed as a sequence of independent and identically distributed (i.i.d.) random variables. On the other hand, the sample size is also "large enough" (usually more than 1000) so that the samples follows from the standard CLT for i.i.d. random variables. Based on such analysis, the OLA scheme can calculate the estimated result based on the CLT and return to the user if the obtained accuracy is satisfactory. Otherwise, OLA needs to collect a larger sample set $S' = S + \Delta S$ for further estimation, (2) **Bootstrap scheme**. This is another approximate query processing scheme, which can avoid the inefficient full data scanning caused by the OLA estimation failure. When the query processing scheme is changed to bootstrap, the initial sample set S for bootstrap estimation is collected by reusing the already received samples from OLA scheme coupled with the new samples collected from the sampling stage. After the initial sample set S is generated, B resample sets of each has $|S|$ samples are taken from S with replacement. These resample sets are used to generate B results for deriving a result distribution, which

[1]HOP is a modified version of the original MapReduce framework, which can bridge the gap between the batch-oriented original MapReduce paradigm and the requirement of OLA for interactively approximate query processing by a pipeline between Map and Reduce.

[2]Southeast University Cloud Platform, which supports data processing applications of the whole university.

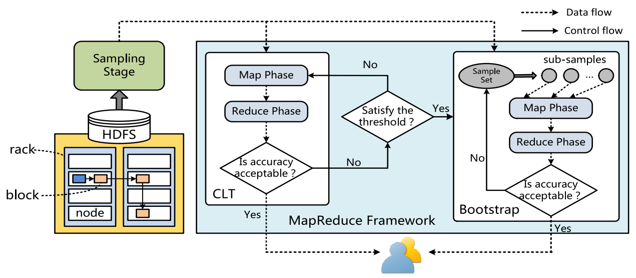

Fig. 2. Architecture of Hybrid Approximate Query Processing System.

are then used for calculating the final approximate result. If the accuracy obtained is unsatisfactory, the above resampling phase is repeated from the new sample $S' = S + \Delta S$, and (3) **Dynamic scheme switcher**. The dynamic scheme switcher monitors the progress of OLA queries and determines when to switch the unpromising OLA queries into the bootstrap scheme for further processing based on the current probability of OLA estimation failure. Moreover, we adopt a progressive estimation method to reduce the probability of false positive for switching.

III. Dynamic Switching Mechanism

In this section, we discuss the details of the dynamic scheme switcher of our hybrid system. The task of dynamic scheme switcher is to switch the unpromising OLA queries into the bootstrap scheme according to the probability of OLA estimation failure. Note that the relatively larger failure probability means the switching operation will be trigged more possibly. And the key of the scheme switcher is to determine when to conduct the switching operation. A straight way for failure evaluation is the static strategy that we define a switching threshold as a fixed number of samples n, a fraction p of the total data size N. In this way, the switcher is trigged if the number of collected samples is larger than n. However, there are two major problems of such static strategy: (1) the fixed threshold cannot adapt to different dataset with varying characteristic such as data distribution. For example, the switching threshold for much more skewed data set should be selected much smaller, since the probability of OLA failures is relatively higher than the uniform distribution due to the poor sample quality, and (2) the fixed threshold may not be the appropriate one for all the queries, leading to the false positive if the threshold is too small (switch earlier) or resulting in more unpromising OLA iterations if the threshold is too large (switch later). To handle this problem, we derive a probabilistic model to evaluate the possibility of OLA estimation failure, and then propose a dynamic switching mechanism based on the OLA failure probability rather than a predefined threshold.

A. Possibility of OLA Estimation Failure

Note that the OLA query can be successfully processed and return the acceptable approximate result only if the drawn sample set without replacement is unbiased. And an important characteristic of unbiased sample is all the samples need to be selected with the same probability, which means the result of sampling without replacement can be viewed as the result of sampling with replacement.

Given the arbitrary random sample $S = \{t_{L_1}, t_{L_2}, ..., t_{L_n}\}$ is drawn without replacement from the whole dataset R, where L_i denotes the random index of the ith tuple t_{L_i} retrieved from R. And there is an obviously characteristic of S that is $L_i \neq L_j$ for all $i \neq j$, which means each tuple in S is drawn only once. If the sample set S can be viewed as the sample set collected with replacement, then the probability of selecting S with replacement should be relatively higher. Otherwise, S cannot be viewed as the normal result of sampling with replacement, and it is unreasonable to use the regular result of sampling without replacement instead of the unconventional result generated from the sampling with replacement. Note that, the probability of selecting n unique tuples as the sample set S with replacement can be calculated as:

$$P_w = \frac{\prod_{i=1}^{n}(m - i + 1)}{m^n} \tag{1}$$

Where m denotes the total size of the whole dataset. And the intrinsic relation between P_w and the possibility of OLA estimation failure denoted as P_f, can be briefly summarised as: (1) P_f is getting higher with the decreasing of P_w, since the smaller P_w indicates the sample set generated from sampling without replacement cannot be seen as the one with replacement of higher probability, and (2) P_f approximates to 100% when $P_w \to 0$. In order to determine the appropriate switching moment for each query, we should derive a mapping function $P_f = f(P_w)$ to depict the P_f. Given a running OLA query Q, our switching mechanism is working as follows: Firstly, Q calculates the P_w by Equation(1) and obtains the corresponding P_f through the mapping function f for each estimation iteration. Then, the scheme switcher determines to switch OLA into bootstrap according to P_f. For example, if $P_f = 85\%$ for a certain query then this OLA query will be switched into bootstrap with the probability of 85%. We note that the mapping function f is the key factor to ensure the effectiveness of our switching mechanism. If the mapping function f can nicely reflect the actual trend of OLA running, then the overhead caused by the "early switching" and "late switching" can be reduced significantly.

In order to obtain a well mapping function f to depict the intrinsic relation between P_w and P_f, we first conclude the characteristics of OLA processing based on the observation from the preliminary study mentioned in Section 1:

Characteristic 1: The increment of P_f in the initial stage of OLA is relatively slower, since the fact that most OLA queries can be completed within 80 iterations.

Characteristic 2: With the decrement of P_w caused by the enlarged required sample size n, P_f is increased gradually and finally infinitely close to 100%. This is because the fact that the number of successful OLA queries begins to reduce gradually after 80 iterations and none of the queries can be completed more than 150 iterations.

Besides the characteristic associated with P_w, the skewness of data distribution is another factor affects P_f:

Characteristic 3: Given the same number of samples have been received, the P_f of skewed dataset is larger than the one of uniform dataset, since the representative sample set cannot be easily guaranteed for the skewed dataset, leading to relatively larger estimate error and requiring more iterations.

Based on the characteristics concluded above, we derive the following decline function of P_f according to P_w:

$$P_f = 1 - e^{-\frac{(1-P_w)^{\mu \cdot s}}{\lambda}} \tag{2}$$

And the shape of this curve can be determined by the following parameters:

(1) μ is the parameter that control the smoothness of P_f for the relatively higher P_w. The larger μ represents the smoother P_f in the beginning, which means that the OLA failure will not happen with higher probability in the initial stage of OLA.

(2) λ is the parameter that guarantee the value of P_f infinitely close to 100% when $P_w \to 0$, which indicates the fact OLA failure will occur with higher probability if the collected sample set is biased.

(3) s is the degree of skewness, which is considered as a non-stationary parameter to reflect the effect of skewed data distribution since the data distribution could be changed due to the data update during the processing. The value range of s is $(0, 1]$, $s = 1$ represents the uniform distribution and the smaller s indicates the more skewed data distribution.

B. Parameters Configuration

Given the different datasets with varying data size, we set λ and μ be the static value since the data size is not the major factor for the estimation failure. While s is dynamically changed for varying data distribution and we have $s = \frac{1}{1+z}$, where z is the value of the exponent characterizing the skewness in the classical Zipfian distribution[6]. To perform our hybrid system efficiently we need to configure the parameters λ and μ with appropriate values. For the configuration of λ, we should guarantee the value of P_f infinitely close to 100% when $P_w \to 0$. To estimate the required λ, we set $s = 1$ and let μ be a larger enough value to enlarge the affect of μ to P_f. In practice we found that $\mu = 10$ gives robust results. Given a fixed μ, the mapping function f is then computed for a small P_w (e.g. 1%) with different candidate values of λ in descending order, and the test interval of λ is 0.01. The execution terminates when the calculated $P_f \geq \varepsilon$, where ε is a predefined value that very closes to 100%. And we note that if the value of λ so determined is satisfying the requirement of P_f for the larger μ, then it can also be held for any smaller μ that is $P_f(\forall \mu_i < \mu, \lambda) \geq \varepsilon$ (this is can be explained by Equation 2). Therefore, a stable λ is determined to guarantee the requirement of P_f for varying μ. While in the case of μ, we need to find an appropriate value of μ to guarantee the switching operation is neither too aggressive nor too conservative. Given μ_i, μ_j and a random P_f, we can calculate the corresponding $P_w(i)$ and $P_w(j)$ according to $f^{-1}(P_f, \mu, \lambda)$. If $\mu_i > \mu_j$, then we have $P_w(i) < P_w(j)$ which indicates the switching operation of μ_i is more conservative than the one of μ_j. Note that, the conservative switching may lead to the unpromising OLA processing, while the aggressive switching may result in the false positive and generate the extra computation cost of bootstrap processing. And in general, the overhead generated from aggressive switching is usually larger than the one caused by the conservative switching, since OLA scheme is more efficient than bootstrap scheme. Therefore, we set λ to be the estimated appropriate value obtained above and test the performance of hybrid system against a uniform test dataset for different μ in descending order, until the terminate condition $T_{\mu_i} \leq T_{\mu_j}$ is satisfied, where T_{μ_i} indicates the time consumption of the case with $\mu = \mu_i$. The μ_i so determined is the estimated value of appropriate μ, since μ_i is the performance inflection point.

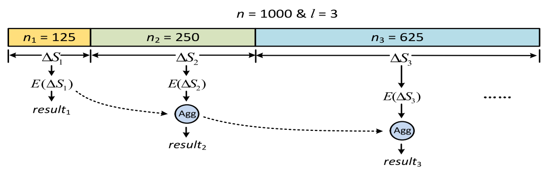

Fig. 3. Example of Progressive Estimation.

The estimation to λ and μ is conducted only once as a preprocessing phase. The appropriate value is obtained as $\lambda = 0.21$ and $\mu = 6$, and we will show the effect of μ in Section 6 for more details.

C. Progressive Estimation

In our hybrid system, the false positive is an important factor that affects the overall performance, since the time cost of bootstrap is usually larger than OLA scheme. To reduce the false positive, there are two efficient methods. The first one is to adjust parameter μ to an appropriate value, and the other one is to complete the OLA queries successfully as far as possible before the relatively larger P_f is reached.

Given an OLA query, the unbiased sample set is the key factor that determines the execution efficiency. But the second characteristic of unbiased sample set cannot be guaranteed for a certain condition due to the uncertainty of random sampling, which means the representative sample may appear any time during the processing period. Then a straight way is to set a static smaller ΔS to estimate OLA queries in a finer granularity. Given the required sample size n, the number of estimation iterations is $\frac{n}{\Delta S}$. For smaller ΔS, the OLA queries may be completed earlier since more opportunities are obtained to detect the representative sample set. However, the smaller ΔS indicates the more estimation iterations, resulting in more estimation overhead. In order to complete OLA queries earlier with lower overhead, we present a progressive estimation method.

We first set a required smaple size n as the estimation period. And we split n into l smaller subintervals each of size n_i, which is used to represent the size of ΔS_i for the ith iteration. The size of ΔS_i can be calculated as follows:

$$\Delta S_i = \begin{cases} \frac{n}{2^{l-i+1}} & \text{if } 1 \leq i \leq l-1 \\ n - \sum_{x=1}^{l-1} \frac{n}{2^{l-x+1}} & \text{if } i = l \end{cases} \tag{3}$$

For the ith iteration, we calculate the statistics $E(\Delta S_i)$ and estimate the query result based on it. If the accuracy expectation is not satisfied, the we enlarge the sample set with ΔS_{i+1} to compute the statistics and calculate the query result combined with the previous statistics $E(\Delta S_i)$. If the total sample size reaches the estimation period n, then the ΔS is reset and we repeat the above operations. In our experiments we found it to be sufficient to set $l = 3$ for $n = 1000$, and Figure3 shows the example instance of such configuration.

Algorithm 1 Query Processing Scheme

```
1: input: Query $Q$
2: SampleInfo *sInfo* = new SampleInfo($Q$)
3: EstimateInfo *eInfo* = new EstimateInfo($c$, $e$, $Q$)
4: JobConf *HybridJob* = new JobConf(*HybridJob.class*)
5: // initialize the HybridJob
6: DefaultStringifier.store(*HybridJob*, *sInfo*, "*sInfo*")
7: DefaultStringifier.store(*HybridJob*, *eInfo*, "*eInfo*")
8: JobClient.runJob(*HybridJob*)
9: FileSystem *fs* = FileSystem.get(*outputPath*, *conf*);
10: while true do
11:    boolean *terminate* = getTerCondition(*fs*)
12:    if *terminate* then
13:       break
14:    end if
15: end while
```

IV. Implementation Over MapReduce

In this section, we implement our hybrid system over the MapReduce paradigm. The query processing scheme can be split into two phases. In the first phase, a certain query Q is submitted as a regular MapReduce job for parallel processing. And in the second phase, a circle detection is conducted to check whether the termination condition is reached.

Algorithm1 illustrates the general procedure. Given a query Q, we need to generate two input parameters firstly (lines 2-3). The first one is *sInfo*, which is used to record the required sample size of each $B_i \in \{Q.p \cap \mathcal{B}\}$ ($Q.p$ indicates the predicate of Q, and $\mathcal{B}$ is the block set in HDFS), denoted as $\{B_i : \langle \Delta S_1^i, ..., \Delta S_l^i \rangle\}$. Note that ΔS_j^i represents the required sample set collected from B_i for the jth progressive estimation. While the second parameter is *eInfo*, which is used to record the user defined confidence, error rate and the aggregate column and aggregate type of Q to support the accuracy estimation. And both *sInfo* and *eInfo* are stored as the parameters of JobConf (lines 6-7). After the *HybridJob* is submitted (line 8), a termination condition detection is conducted (lines 9-15) to read the estimated result from the *outputPath* of HDFS and terminate the job earlier if the result satisfies the accuracy requirement.

A. Single Relation Operation

In this paper, we take SUM, COUNT, AVG as example to show how estimates and confidence intervals can be obtained in the MapReduce framework. Note that, the detailed MapReduce implementation for the case of single relation and multi-relations is different, so that we will consider these two cases separately in the following subsections.

The Map logic of the single relation is shown in Algorithm2. Firstly, the map task needs to load *sInfo* and *eInfo* from *HybridJob*, which can be implemented by overwriting the configure function in MapReduceBase (lines 5-6). For each $\langle k_i, v_i \rangle$, we record it in the ΔS and increase the total number of received sample *sNum* (line 8), which is used to calculate the probability of OLA failure P_f. If the map task collects

Algorithm 2 Map Logic for Single Relation

```
1: input: LongWritable key, Text value
2: output: OutputCollector⟨Text,Text⟩ outputKV
3: $\Delta s \leftarrow \emptyset$ // the increased progress sample
4: *sNum*← 0 // the number of samples have been received
5: SampleInfo *sInfo* = DefaultStringifier.load
   (*HybridJob*, "*sInfo*", *SampleInfo.class*)
6: EstimateInfo *eInfo* = DefaultStringifier.load
   (*HybridJob*,"*eInfo*", *EstimateInfo.class*)
7: for $\forall\langle k_i, v_i\rangle$ in *input* do
8:    $\Delta s$.add($v_i$); *sNum*++
9:    if isEnough($\Delta s$.size, *sInfo*, *this.BlockID*) then
10:      $P_f$ = getFailProbability(*sNum*)
11:      if Random(0,1)< $P_f$ then
12:         // OLA scheme
13:         *stats* = getStats($\Delta s$, *eInfo*)
14:         *outputKV*.collect($Q$, ⟨"*OLA*", *stats*, *taskID*⟩)
15:         $\Delta s \leftarrow \emptyset$
16:      else
17:         // bootstrap scheme
18:         $RS_{\Delta s}$.add(reSample($\Delta s$)) // resampling from $\Delta s$
19:         *statsSet*.getstatsSet($R_{\Delta S}$, *eInfo*)
20:         *outputKV*.collect($Q$, ⟨"*bstrap*", *statsSet*, *taskID*⟩)
21:         $\Delta s \leftarrow \emptyset$
22:      end if
23:    end if
24: end for
```

enough samples, then the estimation is conducted (line 9). And a current P_f is calculated to determine the switching operation (line 10). If $Random(0, 1) < P_f$, then the OLA scheme is activated, otherwise the bootstrap scheme is processing. In the case of OLA scheme, each map task needs to calculate the intermediate statistics *stats* and collect the output for reduce tasks (lines 13-14). There are three kinds of statistics in the *stats*, which are calculated according to the aggregate column and type in *eInfo*: (1) the sum and count of $\Delta S'$ that is $sum(\Delta S') = \sum_{\forall s_i \in \Delta S'} s_i$ and $count(\Delta S') = |\Delta S'|$, where $\Delta S' \subseteq \Delta S$ indicates the sample set that satisfy the query predicate[3]. And this statistic is used to compute the final approximate result, and (2) the quadratic sum of samples that is $X^2 = \sum_{\forall s_i \in \Delta S'} s_i^2$ (calculated for both SUM and AVG), which is used to compute the variance to support the accuracy estimation, and (3) the number of samples have been received, denoted as *sNum*. While in the case of bootstrap scheme, the map task first conducts a resampling to generate n resamples and stores them in $RS_{\Delta S}$ (line 18). Then it computes the intermediate statistics for $\forall \Delta S_i \in RS_{\Delta S}$ and records them in the set of statistics *statsSet* (line 19). The statistics of each resample ΔS_i include $sum(\Delta S_i') = \sum_{\forall s_i \in \Delta S_i'} s_i$, $count(\Delta S_i') = |\Delta S_i'|$, where $\Delta S_i' \subseteq \Delta S_i$ indicates the sample set that satisfy the query predicate, and the total number of received samples, that is *sNum*.

The reduce phase is responsible for collecting statistics from map tasks and calculating the estimate to the final query result. In our implementation, each reduce task is only

[3]If the aggregate type is SUM, then calculate $sum(\Delta S')$, and the $count(\Delta S')$ is calculated for COUNT. While in the case of AVG, both them are calculated.

Algorithm 3 Reduce Logic for Single Relation

```
1: input: IntWritable key, Iterator⟨Text⟩ values
2: output: OutputCollector⟨IntWritable,Text⟩ outputKV
3: EstimateInfo eInfo = DefaultStringifier.load
   (HybridJob,"eInfo", EstimateInfo.class)
4: container ← ∅
5: qType = NULL // query scheme: "OLA" or "bstrap"
6: while values.hasNext() do
7:     update(container, values.next())
8:     if container.isAvailable()&&qType=="OLA" then
9:         // OLA scheme
10:        unistats = getuniStats(container, unistats)
11:        result = estimate(unistats, eInfo)
12:        if isAccuracy(result, eInfo, unistats) then
13:            outputKV.collect(key, ⟨result, "accept"⟩)
14:        else
15:            outputKV.collect(key, result)
16:        end if
17:    end if
18:    if container.isAvailable()&&qType=="bstrap" then
19:        // bootstrap scheme
20:        uniStatsSet = merge(container, uniStatsSet)
21:        result = estimate(uniStatsSet, eInfo)
22:        if isAccuracy(result, eInfo, uniStatsSet) then
23:            outputKV.collect(key, ⟨result, "accept"⟩)
24:        else
25:            outputKV.collect(key, result)
26:        end if
27:    end if
28: end while
```

responsible for a certain Q (collect the map output with the key is Q), and the Algorithm3 illustrates the logic of reduce phase. Given a certain Q, the reduce task loads *eInfo* from the *HybrdiJob* and initializes a variable *container* to classify the input *values* from all map tasks of Q (lines 3-4). For each map task M_i, there is a corresponding item in the *container* to record the statistics of Q. In the case of OLA scheme, the structure of *container* is $\{M_i : \langle stats_1, ..., stats_n \rangle\}$, where $stats_j$ indicates the statistics calculated in the jth iteration of M_i. While in the case of bootstrap scheme, the structure is the same except that the $stats_j$ is replaced by $statsSet_j$. Afterwards, the reduce task needs to check the *container* to make suer it available for accuracy estimation (line 8&18), which means the *container* has contained the statistics for all M_i of Q.

For the case of OLA scheme, the reduce task calculates the unified statistics *unistats* in an incremental way, that is the *unistats* of the jth iteration can be computed by aggregating all the $stats_j$ in *container* with the previous *unistats* of the $(j-1)$th iteration (line 10). There are two kinds of unified statistic in each *unistats*, which can be computed as follows:

$$op(S') = \begin{cases} \sum_{\forall M_i} sum(\Delta S') + v_{pre} & \text{if } op = sum \\ \sum_{\forall M_i} count(\Delta S') + v_{pre} & \text{if } op = count \end{cases} \tag{4}$$

$$X_{all}^2 = \sum_{\forall M_i} X_i^2 + v_{pre} \tag{5}$$

Where $S' \subseteq S$ in the Equation (4) indicates the sample set satisfies the query predicate and $S = \sum_{\forall M_i} \Delta S$ is the sample set has been received, and v_{pre} represents the previous $op(S')$ or X_{all}^2 respectively. Then the approximate result can be calculated by estimate function according to the aggregate type in the *eInfo* (line 11):

$$v'_{s|c} = \frac{T}{|S|} \cdot op(S') \qquad v'_a = \frac{v'_s}{v'_c} \tag{6}$$

Where the variable $T = \sum_{B_i \in \{Q.p \cap \mathcal{B}\}} |B_i|$ indicates the total number of tuples in the relevant block set of Q.

Afterwards, the function isAccuracy needs to compute the error bound ε' based on the variance of the aggregate values and the accuracy parameters in the *eInfo* (line 12). Note that the variance of the aggregate values can be calculated by applying the formula $\sigma^2(X) = E(X^2) - E(X)^2$ as follows:

$$\sigma_{op}^2 = \begin{cases} \frac{X_{all}^2}{count(S')} - (v'_a)^2 & \text{if } op = avg \\ \frac{X_{all}^2}{|S|} \cdot T^2 - (v'_s)^2 & \text{if } op = sum \\ \frac{count(S')}{|S|} \cdot T^2 - (v'_c)^2 & \text{if } op = count \end{cases} \tag{7}$$

Then the error bound ε' can be computed based on central limit theorem:

$$P\{|v' - v| \le \varepsilon'\} \approx 2\Phi\left(\frac{\varepsilon' \sqrt{|s|}}{\sigma}\right) - 1 \tag{8}$$

Where $P\{|v' - v| \le \varepsilon'\}$ is the predefined confidence c. If $\varepsilon' \le v' \cdot e$, then we can say the *result* is acceptable to the user, and the final 100c% confidence interval of the aggregation result is $[v' - \varepsilon', v' + \varepsilon']$. At last, the output of this reduce task is collected as the pair of $(key, \langle result, \text{"}accept\text{"}\rangle)$ (line 13).

While in the case of bootstrap scheme, the reduce task computes the unified statistics for each resamle in an incremental way based on Equation(4), but replace $\Delta S'$ with $\Delta S'_i$, and then records each *unistats* in *uniStatsSet* (line 20). Then the reduce task can calculate the approximate result for the ith resample based on Equation(6), denoted as $v'_{(s|c|a,i)}$, and the final estimate to the result of Q can be computed as follows (line 21):

$$\bar{v}'_{s|c|a} = \sum_{i=1}^{n} v'_{(s|c|a,i)} \tag{9}$$

Then the reduce task needs to calculate the error bound ε' (line 22), which equals to $z_c \cdot \sigma^2$ in bootstrap scheme and σ^2 is the variance of all the $v'_{(s|c|a,i)}$[4]. Then the final 100c% confidence interval is $[\bar{v}' - \varepsilon', \bar{v}' + \varepsilon']$.

B. Multiple Relations Operation

In our hybrid system, we implement the estimation of multi-relations in the fly through two MapReduce jobs. In *Job*1, the map logic is similar to Algorithm2, but with some difference: (1) such map logic does not need to calculate the statistics in each mapper, since the statistics are related to the samples from both two relations. And the task of this

[4]z_c is the c-quantile in the standard normal distribution and c is the predefined confidence

map logic is only to collect unbiased samples and assign them to the corresponding reducer for repartition join, (2) such map logic needs to redesign the structure of output as $(\langle Q, s\rangle, \langle qType, rTag, taskID\rangle)$ to support repartition join, where s denotes each collected sample, *qType*="*OLA*" or "*bstrap*" and *rTag* indicates the relation that ΔS are drawn from. The task of the reduce logic of *Job*1 is to conduct an ripple join operation firstly, then compute the partial unified statistics *unistats* and estimated results v' for OLA scheme and compute the set of unified statistics *uniStatsSet* for bootstrap scheme. While in *Job*2, the task of map logic is to classify the result of *Job*1 into the appropriate reducer (each reducer only works for a certain Q). And each reducer of *Job*2 collects the input and conduct the final accuracy estimation for both OLA and bootstrap scheme.

Algorithm 4 Reduce Logic of Job1

```
1: input: Texe key, Iterator⟨Text⟩ values
2: output: OutputCollector⟨IntWritable,Text⟩ outputKV
3: EstimateInfo eInfo = DefaultStringifier.load
   (HybridJob,"eInfo", EstimateInfo.class)
4: container← ∅
5: qType = NULL // query scheme: "OLA" or "bstrap"
6: S(R), S(S) ← ∅ // received sample set in each reducer
7: while values.hasNext() do
8:    update(S(R), S(S), values.next())
9:    if container.isAvailable() then
10:      joinSet = rippleJoin(S(R), S(S))
11:      if qType=="OLA" then
12:         stats = getStats(joinSet, eInfo)
13:         unistats = getuniStats(stats, unistats)
14:         result = estimate(unistats, eInfo)
15:         outputKV.collect(key.Q, ⟨"OLA", unistats, result⟩)
16:      end if
17:      if qType=="bstrap" then
18:         $R_{\Delta S}$.add(reSample(joinSet))
19:         statsSet.getstatsSet($R_{\Delta S}$, eInfo)
20:         uniStatsSet = merge(statsSet, uniStatsSet)
21:         outputKV.collect(key.Q, ⟨"bstrap", uniStatsSet⟩)
22:      end if
23:   end if
24: end while
```

Algorithm4 illustrates the implementation details of reduce logic for *Job*1. The computation of *stats* and *unistats* for OLA scheme is same to the method used in the case of single relation (lines 12-13), but we need to replace T with $T_j = \sum_{\forall M_i} |B_j^i|$, S with $S_j = |S_j(R)| \cdot |S_j(S)|$ for Equations (6-8), where B_j^i indicates the block B_i for M_i that sends samples to the jth reducer, and $S_j(R)$/$S_j(S)$ indicates the sample set has been received in jth reducer for *R*/*S* respectively. While the computation of *statsSet* and *uniStatsSet* for bootstrap scheme is also same to the one of single relation (lines 19-20).

Algorithm5 shows the reduce logic for *Job*2. For OLA scheme, each reducer receives the input from each involved mapper and calculates the final result as follows (line 9):

$$\bar{v}'_{s|c} = \sum_{\forall M_i} v'_{(s|c,i)} \qquad \bar{v}'_a = \frac{\bar{v}'_s}{\bar{v}'_c} \tag{10}$$

Then, the error bound ε' can be calculated based on the variance of aggregate values to generate the confidence interval of Q (line 10). Since $v'_{(s|c|a,i)}$ has approximately a normal distribution with mean $v_{(s|c|a,i)}$ and variance $\frac{\sigma^2_{(s|c|a,i)}}{|S_i|}$ based on the analysis in[9], then the linear combination of multiple independent normally distributed random variables still has a normal distribution, so that the variance of $\bar{v}'_{s|c|a}$ can be calculated as:

$$\sigma^2_{s|c} = \sum_{\forall M_i} \frac{\sigma_{(s|c,i)^2)}}{|S_i|} \qquad \sigma^2_a = \frac{\sum_{\forall M_i} (\sigma^2_{(a,i)} \cdot |S_i|)}{(\sum_{\forall M_i} |S_i|)^2} \tag{11}$$

Finally, according to the basic feature of normal distribution, the error bound ε' can be computed by the following formula:

$$P\{|v' - v| \le \varepsilon'\} \approx 2\Phi\left(\frac{\varepsilon'}{\sigma}\right) - 1 \tag{12}$$

While in the case of bootstrap scheme, the computation of the approximate result and the error bound ε' is same to the case of single relation (lines 18-19), but we need to replace $|S|$ and T with $|S(R)| \cdot |S(S)|$ and $T(R) \cdot T(S)$ respectively for Equation (6), where $S(R/S)$ represents the sample set has been collected form R/S and $T(R/S) = \sum_{B_i \in \{Q.p \cap \mathcal{B}(R/S)\}} |B_i|$.

Algorithm 5 Reduce Logic of Job2

```
1: input: IntWritable key, Iterator⟨Text⟩ values
2: output: OutputCollector⟨IntWritable,Text⟩ outputKV
3: EstimateInfo eInfo = DefaultStringifier.load
   (HybridJob,"eInfo", EstimateInfo.class)
4: container← ∅
5: qType = NULL // query scheme: "OLA" or "bstrap"
6: while values.hasNext() do
7:    update(container, values.next())
8:    if container.isAvailable()&&qType=="OLA" then
9:       result = estimate(cotainer, eInfo)
10:      if isAccuracy(result, eInfo, container) then
11:         outputKV.collect(key, ⟨result, "accept"⟩)
12:      else
13:         outputKV.collect(key, result)
14:      end if
15:   end if
16:   if container.isAvailable()&&qType=="bstrap" then
17:      uniStatsSet = merge(container, uniStatsSet)
18:      result = estimate(cotainer, eInfo)
19:      if isAccuracy(result, eInfo, uniStatsSet) then
20:         outputKV.collect(key, ⟨result, "accept"⟩)
21:      else
22:         outputKV.collect(key, result)
23:      end if
24:   end if
25: end while
```

V. Experimental Study

We have used Hadoop-hop-0.2 to implement our hybrid system and run experiments on a virtual cluster with 32 nodes, in which there are 30 data nodes, 1 name node and 1 secondary name node. And each node contains a dual-core CPU, 10GB of main memory and 100GB disks. Both these virtual nodes

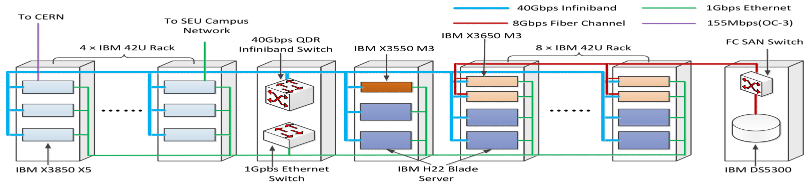

Fig. 4. The Hardware Architecture of SEUCloud

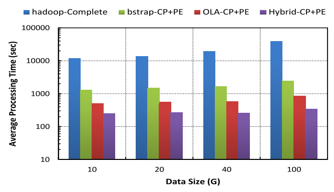

Fig. 5. Effect of Data Size(T_1)

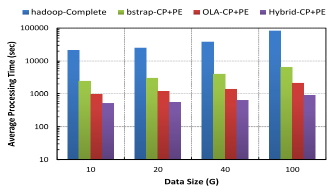

Fig. 6. Effect of Data Size(T_2)

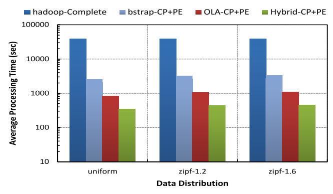

Fig. 7. Effect of Data Distribution(T_1)

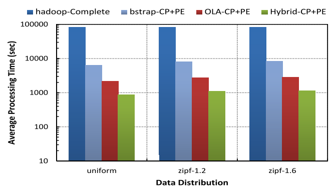

Fig. 8. Effect of Data Distribution(T_2)

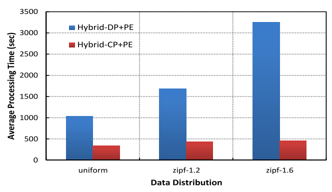

Fig. 9. Effect of Partition Method

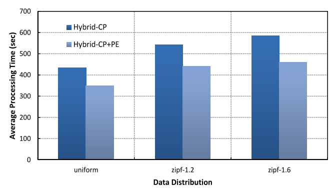

Fig. 10. Effect of Progressive Estimation

are applied from SEUCloud, which supports data processing applications of the whole university, and the architecture of SEUCloud can be depicted by Fig.4.

A. Experiment Settings

In this section, we conduct a set of experiments to evaluate the effectiveness and study the performance characteristics of our hybrid system under skewed input data with different data size. A modified TPC-H toolkit[15] is employed to generate skewed dataset with Zipf distribution, which is determined by the Zipf parameter z (z varies over 0, 1.2, 1.6, where 0 represent the uniform distribution), derived from LINEITEM and ORDER tables as our test data. The scale factor is varied over 10, 20, 40, 100. In our experiments, we generate queries based on the Single relation Template (T_1) and Multi-relations Template (T_2) as follows.

T_1: **SELECT sum**(C_i)|**count**(C_i)|**avg**(C_i) **FROM LINEITEM**
WHERE $[l\_discount > x$ **and** $l\_discount < x + y]$ &
$[l\_quantity > x$ **and** $l\_quantity < x + y]$ &
$[l\_extendedprice > x$ **and** $l\_extednedprice < x + y]$&

T_2: **SELECT sum**(C_i)|**count**(C_i)|**avg**(C_i) **FROM LINEITEM L, ORDERS O**
WHERE $L.orderkey = O.orderkey$ &
$[l\_discount > x$ **and** $l\_discount < x + y]$ &
$[l\_quantity > x$ **and** $l\_quantity < x + y]$ &
$[l\_extendedprice > x$ **and** $l\_extednedprice < x + y]$ &
$[o\_totalprice > x$ **and** $o\_totalprice < x + y]$

Where the parameters x and y are some random values that belong to the value range of C_i. The aggregate type (AVG, COUNT, SUM) for each query is randomly selected during the query generation process. The default error rate e and confidence c used for the accuracy estimation are 0.01 and 95%. Moreover, We use the average processing time of the query as the metric and we execute 100 queries for 5 times in each experiment to remove any side effects.

For comparison purposes, we implement several query processing methods, which can be divided into 4 categories: (1) ***hadoop-Complete*** is a straightforward method without online aggregation to return the exact results, (2) ***OLA*-based** method-s, which are the implementation of original online aggregation, (3) ***bstrap*-based** methods, which are the implementation of the bootstrap scheme, and (4) ***Hybrid*-based** methods are deployed with our scheme switcher. In the case of categories (2∼4), an additional content-aware partition strategy is deployed to obtain further performance improvement, which was proposed in our previous work[19], [20], [21], and the default partition size (*ps*) for our content-aware partition is 6. We implement progressive estimation method (PE) and this content-aware partition strategy (CP) as the optional components (*DP* denotes the default HDFS size-aware partition manner), which can be added to the methods of categories (2∼4). For example, *Hybrid-CP+PE* represents our hybrid system that was implemented with both optional components.

B. Effect of Data Size and Distribution

In this experiment, we first vary the data size from 10G to 100G to evaluate the performance of different methods under the uniform data distribution.

Figure5&6 represent the the results of template 1 and template 2 respectively. We note that both results show the similar trend, but the join queries take much more time since there are more tasks need to be initialized in map phase, and the join operation in the reduce phase consumes extra computation time. Figure5 indicates that the performance of

TABLE I. EFFECT OF PARTITION SIZE.

Metric	Partition Size (*ps*)				
	$ps = 2$	$ps = 4$	$ps = 6$	$ps = 8$	$ps = 10$
Time	1196.47	926.53	460.95	430.68	405.12

TABLE II. COMPARISON OF TWO ESTIMATION MANNERS.

Metric	PE	Default Estimation				
	(1000,3)	200	400	600	800	1000
Time	460.95	563.27	528.93	612.11	872.49	923.62

hadoop-Complete decrease with the increment of data size, while the approximate methods are scalable with regard to the data size and outperform the *hadoop-Complete*. This is because the approximate methods only need a small sample set to calculate the estimated results rather than the whole data set. Among these approximate methods, the performance of *bstrap-CP+PE* is worsen than the *OLA-CP+PE*. This is expected the bootstrap-based method needs to collect more resamples for estimation, leading to more sampling overhead and computation cost. On the other hand, our *Hybrid-CP+PE* outperforms *OLA-CP+PE*, for example, the performance improvement is approximately 49.2% for 100G dataset in T_1. This is because the unpromising OLA queries are switched into the bootstrap processing scheme through the dynamic switcher.

Moreover, in order to study the effect of data distribution in details, we compare the performance of 100G dataset for different data distribution in Figure7&8. Take the result of T_1 as example, we note that these approximate methods are also scalable with regard to the data distribution besides *hadoop-complete*. This is because the fact that the content-aware partition method can significantly improve the performance for skewed data distribution by pruning off the unnecessary data and guaranteeing the appropriate sample proportion. On the other hand, our hybrid system always performs better than the other two approximate schemes since the factor of skewness is carefully considered as the parameter s in our scheme switcher.

C. Effect of Partition Strategy

In this test, we study the effect of partition strategy under 100G skewed dataset. As results of template 1 and 2 show similar trend, we only present the results of template 1.

Figure9 illustrates the performance of hybrid system with both partition methods. Note that the method with our partition strategy outperforms the default one from two aspects: (1) much more scalable with regard to the data distribution, and (2) much better computation performance. This is excepted as the sample quality can be guaranteed as far as possible in the initial stage, reducing the number of failed OLA queries. For example, *Hybrid-CP+PE* is on average 75.34% more efficiently than *Hybrid-DP+PE* for different data distribution.

Moreover, the partition size *ps* is an important parameter affecting the performance of our content-aware partition strategy, so that we show the effect of *ps* in this test too. For facilitate to discuss, each partitioning column has the same partition size (varies from 2 to 10). TableI only shows the result for template 1 (as the template 2 has the similar trend) for zipf-1.6 100G dataset. Note that the processing time of *Hybrid-CP+PE* is significantly decreased for relatively smaller partition size and the performance improvement is getting slower with increment of partition size. This is expected as the extra volume of pruned unnecessary data for "large enough" partition size is getting less than the case of smaller partition size.

TABLE III. EFFECT OF PARAMETER l.

Metric	PE: $n = 1000$			
	$l = 2$	$l = 3$	$l = 4$	$l = 5$
Time	484.21	460.95	504.73	525.34

TABLE IV. COMPARISON OF TWO FUNCTIONS OF P_f.

Metric	Linear	Our method				
	$P_f = 1 - P_w$	4	5	6	7	8
Time	836.51	987.23	698.75	460.95	524.33	587.67
P_s	23.7%	28.8%	12.9%	2.6%	1.8%	1.2%

D. Effect of Progressive Estimation

In this experiment, we test the performance of our hybrid system with/without progressive estimation for 100G skewed dataset. In *Hybrid-CP*, the accuracy estimation is refined periodically for every 500 samples, and we set the estimation period n and number of subintervals l be (1000,3) in *Hybrid-CP+PE*. Figure10 shows the result of the template 1. Note that the performance of *Hybrid-CP+PE* is better than *Hybrid-CP*, since the false positive rate is reduced by applying our progressive estimation. On the other hand, we also compare our progressive estimation strategy with the default sampling method for different refine period for 100G zipf-1.6 dataset, as shown in TableII. Note that in the case of default methods, the larger refine period leads to less estimation overhead but poorer efficiency to eliminate the false positive. While in the smaller refine period, the better efficiency for false positive reduction is achieved with more estimation overhead. And the optimal result for default method is obtained when refine period equals to 400. Note that our progressive estimation is better than the default method for different refine period. Moreover, we also tested the performance of our progressive sampling for different l, as shown in TableIII, which indicates $l = 3$ for $n = 1000$ is sufficient for our experiment.

E. Effect of the Function of $P_f = f(P_w)$

In this test, we take the result of the T_1 for zipf-1.6 100G dataset as an example to illustrate how the function $f(P_w)$ affects the performance of our hybrid system. In Section III.B, we have given the default configuration of the parameter $\lambda = 0.21$. Then we show the result of *Hybrid-CP+PE* with such optimal λ for different μ and compare it to the one with leaner function $P_f = 1 - P_w$ in TableIV. The metrics consists of the computation cost (time) and the percentage of the switched OLA queries P_s. As shown in the TableIV, P_s is decreased along with the increment of μ. This is expected as the smaller μ indicates the relatively higher P_f in the initial stage of OLA based on Equation(2), which indicates the more aggressive switching policy leading to extra bootstrap processing overhead. For the larger μ the false positive can be avoided to an extend, however, the false negative emerges that leads to extra OLA processing overhead. And we note that $\lambda = 0.21$ and $\mu = 6$ is sufficient for our experiment, and our carefully designed mapping function of $P_f = f(P_w)$ outperforms the linear function of P_f in most cases.

VI. Related Work

Online aggregation[10] is one commonly-used approximate query processing technique to provide a time-accuracy tradeoff for aggregation queries. And in [8], Haas illustrates how the different theories can be used to derive formulas for both large-sample and deterministic confidence intervals. To support join operation for online aggregation, Hass and Hellerstein introduced a novel join methods called ripple joins in[9]. To handle the memory overflows in ripple join, hash ripple join algorithm is proposed in[13], which combines parallelism with sampling to speed convergence. However, all works in[10], [8], [9], [13] are focused on single query processing. Therefore, Wu et al. proposed COSMOS to support multiple queries optimization for online aggregation[23].

However, these centralized online aggregation methods cannot be extended to distributed manner easily, so the well designed distributed online aggregation systems are proposed along with the development of P2P and cloud computing[22], [4], [3], [14], [17], [20]. Wu et.al. extend the online aggregation to a P2P context where sites are maintained in a DHT network[22]. In addition, [4], [3] demonstrated a modified version of Hadoop that supports online aggregation, which can only return the query progress without any precision estimation. And [14] proposed a Bayesian framework to produce estimates and confidence intervals for online aggregation in Hyracks[2], but lacks of the consideration for join operation. In[17], the authors formulated a statistical foundation that supported block-level sampling in online aggregation to improve the sampling efficiency. While in the work of [20], the authors proposed a fair-allocation strategy to guarantee the storage and computation load balancing for running online aggregation over MapReduce-based cloud system.

However, there is an serious limitation that restricts the OLA performance that is the estimation failure of OLA query processing. And none of the above papers mainly focused on such problem to further improve the OLA performance as we have discussed in this paper and implemented the solution over the MapReduce framework for further scalability.

VII. Conclusions

To support OLA in the cloud makes it possible to save user's time and computation cost by taking acceptable approximate results earlier. However, there exists a major limitation restricting the OLA performance that is the estimation failure of OLA. To overcome this limitation, we propose a hybrid approximate query processing system, which consists of several contributions: (1) a dynamic scheme switching mechanism to switch unpromising OLA queries into the bootstrap scheme, avoiding the whole dataset scan, and (2) a progressive estimation method to eliminate the false positive of switching. To our best knowledge, our hybrid system is the first work on studying the estimation failure of OLA. We have implemented the hybrid approximate query processing system in Hadoop, and conducted an extensive experimental study on the TPC-H benchmark for skewed data distribution. Our results demonstrate that the hybrid system can produce acceptable approximate result within a time period one order of magnitude shorter compared to the original OLA.

VIII. Acknowledgments

This work is supported by National Key Basic Research Program of China under Grants No. 2010CB328104, National Natural Science Foundation of China under Grants No. 61320106007, No. 61202449, No. 61370207, No. 61300024, National High Tech R&D Program of China under Grants No. 2013AA013503, China National Key Technology R&D Program under Grants No. 2010BAI88B03 and No. 2011BAK21B02, China Specialized Research Fund for the Doctoral Program of Higher Education under Grants No. 20110092130002, Jiangsu research prospective joint research project under Grants No. BY2012202, No. BY201307301, Jiangsu Provincial Key Laboratory of Network and Information Security under Grants No. BM2003201, Key Laboratory of Computer Network and Information Integration of Ministry of Education of China under Grants No. 93K-9.

References

[1] The apache software foundation. hadoop. http://hadoop.apache.org.

[2] V. Borkar, M. Carey, R. Grover, et al. Hyracks: A flexible and extensible foundation for data-intensive computing. ICDE '11, pp. 1151-1162.

[3] J.H. Böse, A. Andrzejak, et al. Beyond online aggregation: parallel and incremental data mining with online map-reduce. MDAC '10, pp. 3.

[4] T. Condie, N. Conway, P. Alvaro, and Hellerstein. Online aggregation and continuous query support in mapreduce. SIGMOD '10, pp. 1115-1118.

[5] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears. Mapreduce online. In *NSDI*, volume 10, page 20, 2010.

[6] E. K. Donald. The art of computer programming. *Sorting and searching*, 3:426–458, 1999.

[7] B. Efron. Bootstrap methods: Another look at the jackknife. *Annals of Statistics*, 7(1):1–26, Jan. 1979.

[8] P. J. Haas. Large-sample and deterministic confidence intervals for online aggregation. SSDBM '97, pp. 51-62.

[9] P. J. Haas and J. M. Hellerstein. Ripple joins for online aggregation. *SIGMOD Rec.*, June 1999, pp. 287-298.

[10] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online aggregation. *SIGMOD Rec.*, 26:171–182, June 1997.

[11] H. Herodotou, H. Lim, G. Luo, et al. Starfish: A self-tuning system for big data analytics. In *In CIDR*, pages 261–272, 2011.

[12] N. Laptev, K. Zeng, and C. Zaniolo. Early accurate results for advanced analytics on mapreduce. *PVLDB*, 5(10):1028–1039, 2012.

[13] G. Luo, C. J. Ellmann, P. J. Haas, and J. F. Naughton. A scalable hash ripple join algorithm. SIGMOD '02, pp. 252-262.

[14] N. Pansare, V. Borkar, C. Jermaine, and T. Condie. Online aggregation for large mapreduce jobs. VLDB '11, pp. 1135-1145.

[15] S.Chaudhuri and V. Narasayya. Program for tpc-d data generation with skew. ftp://ftp.research.microsoft.com/pub/user/viveknar/tpcdskew.

[16] S.E.Syrjala. A bootstrap approach to making sample-size calculations for resource surveys. SCC '11, pages 53–60, 2001.

[17] Y. Shi, X. Meng, et al. You can stop early with cola: online processing of aggregate queries in the cloud. CIKM '12, pages 1223–1232.

[18] J. S. D. Tu. The jackknife and bootstrap. *Springer Series in Statistics, New York*, 85:486492, 1995.

[19] Y. Wang, J. Luo, et al. Oats: online aggregation with two-level sharing strategy in cloud. *DPD*, pages 1–39, 2014.

[20] Y. Wang, J. Luo, et al. Partition-based online aggregation with shared sampling in cloud. *JCST*, 2013, 28(6): 989-1011.

[21] Y. Wang, J. Luo, et al. Improving online aggregation performance for skewed data distribution. DASFAA'12, pp. 18-32.

[22] S. Wu, S. Jiang, B. C. Ooi, and K.-L. Tan. Distributed online aggregations. *Proc. VLDB Endow.*, August 2009, pp. 443-454.

[23] S. Wu, B. C. Ooi, and K.-L. Tan. Continuous sampling for online aggregation over multiple queries. SIGMOD '10, pp. 651-662.