ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 6, June 2015

Using Map-Reduce performing Online Query Aggregation

M Sabir Chauhan[1], Rekha Jadhav[2]

M.E Computer Engineering, G.H.R.I.E.T., Savitribai Phule University, Pune, India[1]

Professor, Department of Computer Engineering, G.H.R.I.E.T., Savitribai Phule University, Pune, India[2]

Abstract: In online aggregation, a database system processes a user's aggregation query in an online fashion. During the query processing, the system gives the user an estimate of the final query result, with the confidence bounds that become tighter over time.Map-Reduce programming approach have close relationship with cloud computing. Today, online aggregation is a very attractive technology. In this I have described how online aggregation can be built into a Map-Reduce system for large-scale data processing. In this I also describes the detail implementation of OLA models in Hayracks . In literature survey section we have briefly discussed various online aggregation methodology such as OATS,COLA , Parallel Online Aggregation with their advantages and limitations. Lastly, I have presented advantages and limitation of OLA. Online Aggregation is an attractive sampling-based technology to response aggregation queries by an estimate to the final result, with the confidence interval becoming tighter over time. It has been built into a Map-Reduce-based cloud system for big data analytics, which allows users to monitor the query progress, and save money by killing the computation early once sufficient accuracy has been obtained. However, there are several limitations that restrict the performance of online aggregation generated from the gap between the current mechanism of Map-Reduce paradigm and the requirements of online aggregation, such as: 1) The low sampling efficiency due to the lack of consideration of skewed data distribution for online aggregation in Map-Reduce.
2) The large redundant I/O cost of online aggregation caused by the independent job execution mechanism of Map-Reduce.

Keywords: Cloud, Hadoop ,Map-Reduce, Hayracks , Online Aggregation.

I. INTRODUCTION

When we are running online aggregation (OLA)[1] during query processing, a database system gives a user a statistically valid estimate for the final answer to an aggregate query, along with confidence bounds. The confidence bound is calculated in the following form: "with probability p, the actual query answer is within the range low to high". As the computation progresses, the bounds goes narrow, until the bounds are zero width, that indicate the complete accuracy. The main benefit of using OLA is that if an acceptably accurate answer can be arrived at very quickly (may be, tiny fraction of the time needed to run the entire query), the query can be aborted, and in this way it is possible to save computer and human time.

In this work, Map-Reduce was originally designed as a batch oriented system.Generally,it is used for interactive data analysis where a user submits a job to extract information from a data set, and then waits to view the results before proceeding with the next step in the data analysis process. This trend has accelerated with the development of high-level query languages such as Hive , Pig and Sawzall that are executed as Map-Reducejobs.Traditional Map-Reduce implementations provide a poor interface for interactive data analysis, because they do not produce any output until the job has been executed to completion. In many cases, user need a "quick and dirty" approximation over a correct answer that takes much longer to compute. In order to get the intermediate result, online aggregation has been used, but the batch-oriented nature of traditional Map-Reduce implementations makes these task difficult to apply .Now day, Online Aggregation has a good scientific impact, but its commercial impact has been limited or even non – existent because of the following two main reasons:

1. During the implemention of OLA within a database engine we require to do the extensive changes to the database kernel. OLA requires some sort of statistically quantifiable randomness within the database engine. Most of the OLA algorithms that has been used, require the blocks (or tuples) in a relation be processed using a "random" ordering. For random ordering we need to do significant changes to most kernels.
2. Some query finish its execution within a fraction and returns the result to the user, even if the user is relatively happy with the results. Ending the query early might save some CPU cycles or disk bandwidth that can then be used by others, but the user who killed the query early may not benefit directly. Furthermore, the database hardware/software/maintenance costs in a self-managed system are not elastic, and do not decrease appreciably if many users decide to stop their queries early.

II. LITERATURE SURVEY

1. Online Aggregation in the cloud (OLA Cloud)

In OLA Cloud implementation[3] ,I have used Hadoop Online Prototype (HOP) as a natural candidate for the underlying query processing engine. HOP is a modified version of the original Map-Reduce framework, which is designed to construct a pipeline between Map and

DOI 10.17148/IJARCCE.2015.4632

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940
IJARCCE
International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 6, June 2015

Reduce so that the reduce task could start immediately as long as any Map output is generated. Such pipeline property can help to support OLA by returning the early approximate result of the query, and scaling up such result with the query progress.In this section,I have described the data flow of OLA Cloud, which consists of two steps:
A. Content-aware repartition with fair-allocation strategy.
B. OLA query processing with shared sampling.

A. *Content-Aware Repartition With Fair-Allocation Strategy :* The first step it is nothing but a pre-processing of OLA Cloud, which is implemented by using two functional components:content-aware repartition and fair allocation.This is motivated by the observation that the performance of online aggregation is actually determined by the data distribution rather than data size . Given an input file has already been loaded into the HDFS (Hadoop Distributed File System), the task of such pre-processing is to reorganize the original file in the granularity of blocks according to the attributes. For the content-aware partition, author proposed a block placement strategy called fair-allocation, which replaces the default random strategy, to guarantee the storage and computation load balancing for our content-aware repartition method.

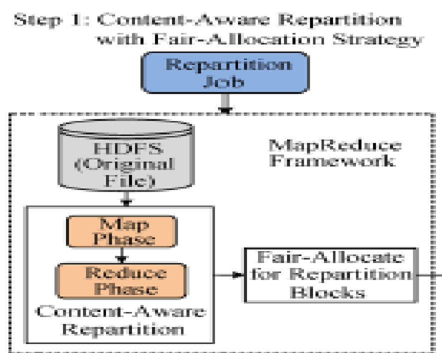

Fig.1.Content-Aware Repartition With Fair-Allocation Strategy.

B. OLA Query Processing With Shared Sampling
This step is implemented by the component called shared sampling.which provide support to the essential procedures of OLA such as sample collection, statistic computation and accuracy estimation.The multiple queries are decomposed into a series of map tasks initially. And we can reuse the samples retrieved by one task to evaluate a number of queries rather than each query retrieves its own samples if there has potential dependency among these map tasks. Above figure shows that OLA Cloud collects a batch of query jobs and analyzes the sharing opportunities among the queries in the granularity of task and groups the shared tasks together to form a new grouped map task, in which the samples collected are reused for accuracy estimation of each involved query.The reduce phase estimates the approximate results for the query jobs once the reducer receives a sufficient map output (a pipeline model). If the accuracy obtained is unsatisfactory, the above reduce process is repeated by taking the latest map output which is aggregated with the previous approximate results to make a new estimate for higher accuracy. The final result is returned when desired accuracy is reached and the users can stop the query early before its completion.

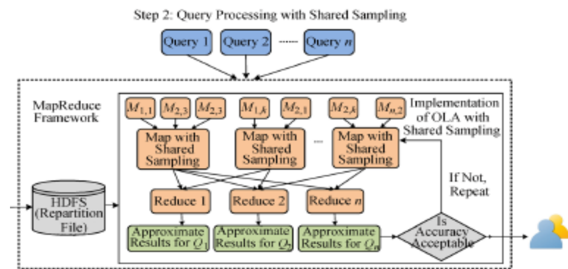

Fig .2. OLA Query Processing With Shared Sampling.

2. Hadoop Implementation To Support OLA Within A Single Job & Between Multiple Jobs:

A. Single-Job Online Aggregation:
In HOP[4], the data records produced by map tasks are sent to reduce tasks shortly after each record is generated. However, to produce the final output of the job, the reduce function cannot be invoked until the entire output of every map task has been produced. Here, it is possible to support online aggregation by simply applying the reduce function to the data that a reduce task has received so far.The output generated of such an intermediate reduce operation is called snapshot. Users would like to know how accurate a snapshot is: that is, how closely a snapshot resembles the final output of the job. Accuracy estimation is a hard problem even for simple SQL queries and particularly hard for jobs where the map and reduce functions are user-defined code.

B. Multi-Job Online Aggregation:
Online aggregation is particularly useful when it is applied to a long-running analysis task consist of multiple Map-Reduce jobs.This version of Hadoop allows the output of a reduce task to be sent directly to map tasks. This feature can be used to support online aggregation for a sequence of jobs. Suppose that job_1 and job_2 are two Map-Reduce jobs, and consider job_2 consumes the output of job_1. When job_1's reducers compute a snapshot to perform online aggregation, that snapshot is written to HDFS, and also it is sent directly to the map tasks of job_2 . The map and reduce steps for job_2 are then computed as normal, to produce a snapshot of job_2's output. This process can then be continued to support online aggregation for an arbitrarily long sequence of jobs.

3. COLA: A Cloud-Based System for Online Aggregation

COLA [7] provides an online aggregation executions engine with sampling techniques that support incremental and continuous computing aggregation and minimize the waiting time before an acceptable estimate is available.User friendly SQL queries are also supported in COLA.COLA can convert non OLA jobs into online version so that user do not have to write any special purpose code to make estimate.

C. COLA System Architecture And Implementation
In below Fig 5 shows the System architecture of COLA. In COLA there are four modules: User Interface, Query Engine, Online Aggregation Executor and Data Manager.Users can submit queries through SQL or command-line interface and monitor running estimates via

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940
IJARCCE
International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 6, June 2015

the User Interface. The Query engine serves as a translator that transforms SQL queries into Map-Reduce jobs and converts non-OLA jobs to online mode. The Online Aggregation Executor fetches uniform-random samples from the Data Manager continuously, processes the samples through Map-Reduce jobs in online fashion and reports the estimates back to the client.

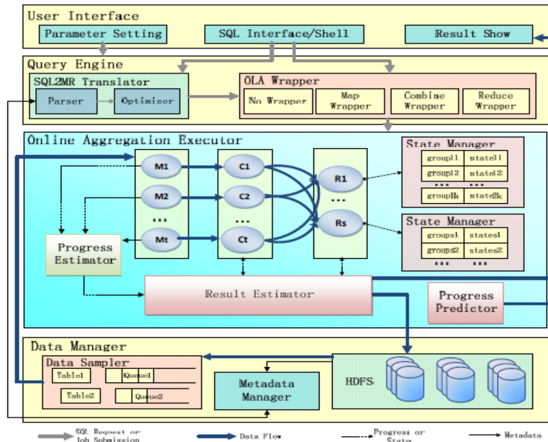

Fig.3. System architecture of COLA.

1. User Interface:

COLA provides interactive and flexible interfaces, users can issue SQL query request through SQL interface or submit Map-Reduce program via shell interface.In addition, the graphical user interface it can also observe the query progress and online estimates with associated confidence intervals during the query processing.

2. Query Engine:

The Query Engine is responsible for compiling the SQL query into directed acyclic graph of Map-Reduce jobs, and translating the non-OLA jobs to online version. Hence users can submit batch-oriented Map-Reduce programs and do not need to have the knowledge of the estimate computation.

3. Online Aggregation Executor:

The Online Aggregation Executor is the key module of COLA to perform online query processing algorithm over Map-Reduce. It is called to process the sample data,and it produces an approximate answers with their associated confidence intervals. It also used to refine the answers. In addition, the module makes predictions about the residual completion time,and also estimates amount saved so far.

4. Data Manager:

The Data Manager makes use of HDFS to store and manage data. It mainly stores the metadata such as mappings between tables and HDFS directories in Metadata Manager, that can be used to do query optimization and compilation in SQL2MR Translator.

Advantages of COLA:

COLA provide progressive approximate aggregate answers for both single table and multiple joined tables.COLA can produce acceptable approximate answers within two orders magnitude shorter time compared to getting the accurate results, which makes it possible to save huge amount of computing cost from the pay-as-you-go cost model in the context of cloud computing.

Advantages of OLA :

1. OLA makes the original platform much more flexible by providing a fast and effective way to obtain approximate results within the prescribed level of accuracy rather than the accurate results. This can significantly improve the analytic performance against the large volumes of data.
2. OLA reduces the economic cost of users on the typically pay-as-you-go cloud systems, that is an user can save money by monitoring the estimated result and killing the computation early once the user gets sufficient accuracy .
3. OLA also increases the overall throughput of the cloud system since the released resources of early terminated OLA queries can be delivered to the other running OLA queries immediately, which helps to increase the parallelism degree and resource utilization.

Limitation of OLA:

1)Sampling efficiency is low due to the lack of consideration of skewed data distribution for online aggregation in Map-Reduce.

2) It increases the I/O cost of online aggregation due to the independent job execution mechanism of Map-Reduce.

III. IMPLEMENTATION

In this I have describe our implementation of the OLA model in Hyracks . Hyracks is a new open source project that supports map and reduce operations, along with higher level relational operations such as filter (selection), projection, and join.The Hyracks architecture is similar to Hadoop—it has a single master node for submitting jobs (queries) and housing the task scheduler, which executes tasks on worker nodes running in the cluster.Hyracks tasks support read and write operations in HDFS , which we leverage to store the input to the map tasks and the output of the reduce tasks. Like Hadoop, when a client submits a Map-Reduce job, Hyracks assigns a single map task to a given block in the input data, and creates a configurable number of reduce tasks that are assigned specific groups using some partitioning function.

In this modified the Hyracks implementation in two ways. First, created a single queue containing the blocks in the input data.The order of the blocks in the queue is uniformly shuffled using the java.util.Collections.shuffle routine from the Java Standard Library.When Hyracks schedules a map task, it assigns the current block at the head of the queue. The map task's execution time includes the time to obtain its assigned block from HDFS, the execution of the map function on each input record, and the execution of the combiner on the complete map function output. In this work we ignore performance issues involving locality; although we do account for block locality in our model. In future work, we plan on investigating locality scheduling techniques reminiscent to Delay Scheduling. Our second modification involves running the estimator in the reduce task during the shuffle phase. In the shuffle phase, the reduce task is continuously receiving the output of completed map tasks. The output of a map task includes a *data* file containing the groups assigned to the reduce task and a *meta-data* file containing

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

IJARCCE

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 6, June 2015

timing and locality information.If the map output contains no groups for a given reduce task then an empty data file is given along with a complete meta-data file.The meta-data file contains the block identifier, the time it took to schedule the block and the block locality relative to the map task execution: machine-local, rack-local, or distant. Also included is the map task IP address, start time and end time.

3.1 System Architecture:

Hadoop is composed of Hadoop Map-Reduce, an implementation of Map-Reduce designed for large clusters,and the Hadoop Distributed File System (HDFS), a file system optimized for batch-oriented workloads such as Map-Reduce. In most Hadoop jobs, HDFS is used to store both the input to the map step and the output of the reduce step. Note that HDFS is not used to store intermediate results (e.g. the output of the map step): these are kept on each node's local file system.A Hadoop installation consists of a single master node and many worker nodes. The master, called the Job-Tracker, is responsible for accepting jobs from clients,dividing those jobs into tasks, and assigning those tasks to be executed by worker nodes.

Data will be collected from online sources, data will be in the form of numeric and alpha numeric form based on the type of input dataset selected by us.Once data is collected we would create a Hadoop Mapping class to map data into our respective format as needed by us for processing.After data mapping we would create a Hadoop Reduce class to reduce the given data into Aggregated form.Algorithms as mentioned in the paper would be used to Reduce the data into Aggregated form (check the following examples) .Once aggregated data is found, we use it for result evaluation and comparison purposes.

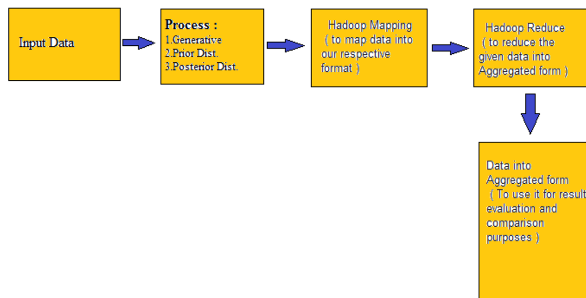

Fig.4. Architecture Online Aggregation of Map-Reduce

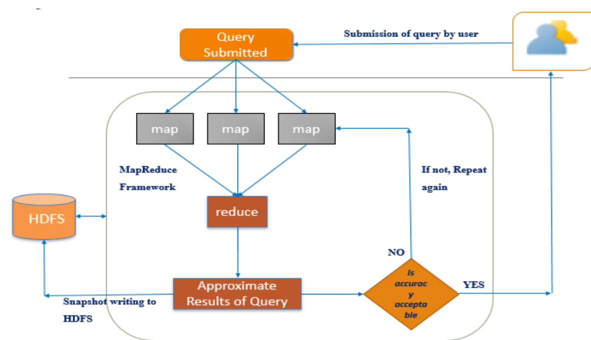

Fig. 5. Architecture of the Proposed System.

3.2 Proposed System Algorithm:

In this,I have[1] consider how estimates and confidence bounds for those estimates can be obtained. As intimated previously, this is a challenging problem, as we must take into account processing times as well as observed aggregate values in order to circumvent the inspection paradox.

1. Overview:

I will apply a Bayesian approach for estimation [13]; for brevity,this section will assume that the reader has some very basic familiarity with Bayesian statistics. The Bayesian approach has several obvious benefits for this particular problem. Most significant is the fact that the inspection paradox "goes away" under the Bayesian approach if one takes into account the time spent waiting for each block to be processed as observed data.

In standard Bayesian fashion, I will first describe a stochastic, parametric process that we imagine was used to produce the"observed" as well as the "hidden" data. The "observed data" will collectively be referred using variable X. This set includes all of the known aggregate values and processing times. Our generative process will also produce a set of unobserved variables collectively referred to as Θ. Θ includes any data that is unobserved (for example, the processing time for a block that has not yet finished)—this data is collectively referred to as Y—as well as any unknown parameters required by the generative process (for example, the mean aggregate value per block). In Bayesian fashion, we will then attempt to infer the distribution P(Θ |X), which is referred to as a *posterior distribution* for Θ. Then, given X as well as P(Θ |X), it is possible to obtain a posterior distribution over the actual query result, which can be used to obtain confidence bounds that are reported to the user.

Note that the discussion in this is directly applicable only to SUM and COUNT queries, which are both evaluated by simply summing xi values (in the SUM case, xi will contain the total aggregate value for the block, and in the COUNT case, xi will contain the tuple count for the block). Extensions to other aggregates such as AVG, VARIANCE and STD DEV are straightforward; in general they require that we maintain zero, first and second moments for each block.

2. Generative process:

To obtain the data that I must analyze to produce estimates and confidence bounds, we imagine that the following stepsare repeated, once for each of the n blocks in the system:

1. $Z_i \sim \text{Normal}(\mu, \Sigma)$

2. $(X_i, Y_i) \leftarrow \text{PostProcess}(Z_i, \sim)$

" ~ " should be read as "is sampled from". After this process has been repeated n times (once for each block)—our goal is then to infer the posterior distribution for Θ using X. This process requires some additional explanation. We begin by describing the vector Z_i. If there are m machines being used to execute a query, we imagine that associated with the ith block is a vector Z_i with 3m + 2

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 6, June 2015

entries, which contains both observed and hidden data. Z_i takes the form:

$$\mathbf{Z}_i = \langle x_i, t_i^{sch}, t_{i,1}^{loc}, t_{i,1}^{rack}, t_{i,1}^{dist}, t_{i,2}^{loc}, t_{i,2}^{rack}, t_{i,2}^{dist}, \ldots t_{i,m}^{loc}, t_{i,m}^{rack}, t_{i,m}^{dist} \rangle$$

3. Prior Distributions

To make our model fully Bayesian , I must supply priors on μ and Σ. In our implementation, each μk , ~ InvGamma(1, 1) (where k refers to the kth dimension in Z_i). The inverse Gamma distribution is a standard, uninformative prior for values that must be non-negative—it makes sense to have non-negative means for all of the time values in the Z_i vector. It will also usually make sense to have a non-negative mean for x_i; if not, then another suitable, uninformative prior can be used.Handling the covariance matrix Σ is a bit trickier. The standard prior distribution for a covariance matrix is the inverse Wishart distribution, because it is "conjugate" for the normal. This means that under certain conditions, upon observing the output from a normal distribution with an inverse Wishart prior on the covariance, th posterior on the covariance is still inverse Wishart. Conjugacy is convenient because it can make inference much easier. Unfortunately, these "certain conditions" are not met in our application because I do not always have actual observations from the normal—we may only know, for example, that the processing time has a lower bound (if I am in "case three" from the previous subsection).

Thus,I choose to use an application-specific prior that is easily factorable; that is, where we can easily write the marginal distribution for each entry in the covariance matrix. This makes deriving a Gibbs sampler for inference much easier (see the next subsection).

Specifically, we let $\sigma_k \sim$ InvGamma(1, 1), where $\Sigma_{k,k} = \sigma_k^2$. Then, we assume that the following process is used to generate the rest of Σ:

while *true* **do**
  for k_1 = *1 to (3m+2)* **do**
    for $k_2 = k_1$ + *1 to (3m+2)* **do**
      $\rho_{k_1,k_2} \sim \text{GenBeta}(-1, 1, 1, 1)$;
      $\Sigma_{k_1,k_2} = \Sigma_{k_2,k_1} = \rho_{k_1,k_2} \times \sigma_{k_1} \times \sigma_{k_2}$;
  if Σ *is positive-definite* **then**
    break;

Algorithm 1: Generation of the covariance matrix Σ

Here, GenBeta(−1, 1, 1, 1) refers to a generalized Beta(1, 1) distribution, stretched to cover the range from −1 to 1 (rather than the usual 0 to 1). What this process does is to essentially sample a correlation ρ for each of the pairs of variables in Z_i, and to then check whether a valid covariance matrix has been obtained (one that is positive definite). If it has not, then the whole process is repeated again.The PDF for Σ can then be written as:

$$P(\Sigma) \propto \begin{cases} 0 \text{ if } \Sigma \text{ is not positive-definite} \\ \left(\begin{array}{c} \prod_k \text{InvGamma}(\sigma_k | 1, 1) \times \\ \prod_{k_1,k_2} \text{GenBeta}(\rho_{k_1,k_2} | -1, 1, 1, 1) \end{array} \right) \text{otherwise} \end{cases}$$

4. Posterior Distribution

In this subsection, we tackle the problem of obtaining a formula for the desired posterior distribution, P(Θ|X). Recall that X =U_i{X_i}, and the unobservable data set Θ contains Y =U_i{Y_i},as well as the normal parameters μ and Σ.

From elementary probability, we know that:

$$P(\Theta|\mathbf{X}) = \frac{P(\mathbf{X}|\Theta)P(\Theta)}{P(\mathbf{X})}$$

This means that there are three quantities that we must derive expressions for: P(X| Θ), P(Θ), and P(X).

We deal with P(X| Θ) first. From the generative process, we know that P(X| Θ) =Π_iP(X_i| Θ). We can easily write an expression for each P(X_i| Θ).

5. Putting It All Together

Since our goal is to produce estimates and confidence bounds for the actual query result, we are not interested in the posterior distribution P(Θ |X) for its own sake. Rather, we will use P(Θ |X) to produce estimates and confidence bounds for the answer.To describe how this is done, note that given a possible value for Θ —combined with the visible data X—we have access to each and every x_i value in the database. Thus, given a particular Θ as well as X it is very easy to compute the query answer as:

$$Q(\Theta, \mathbf{X}) = \sum_i x_i$$

Then by integrating P(Θ |X) over all possible Θ, we obtain various statistics describing the eventual query result. For example, the following gives us the expected value of the query result:

$$\int_{\Theta} P(\Theta|\mathbf{X})Q(\Theta, \mathbf{X})d\Theta$$

And I can obtain the lower end l for a Max % of confidence bound on the query result by computing Λ and l so that:

$$\int_{\Theta \in \Lambda} P(\Theta|\mathbf{X})d\Theta = 0.025 \text{ where}$$
$$\mathbf{max}_{\Theta \in \Lambda}\{Q(\Theta, \mathbf{X})\} \leqslant l \text{ and } \mathbf{min}_{\Theta \in \bar{\Lambda}}\{Q(\Theta, \mathbf{X})\} \geqslant l$$

The upper end could be computed in a similar fashion.Unfortunately, performing this sort of computation exactly is difficult.The difficulty is often circumvented using so-called "Markov Chain Monte Carlo" (MCMC) methods [15] that sample directly from a distribution such as P(X| Θ). In our case, we apply a particular MCMC method called a *Gibbs sampler* to the problem [4].The samples obtained from a Gibbs sampler are easily used to compute expected value and confidence bounds.

IV. EXPECTED RESULTS

In the Online Aggregation I will be used three basic process onto the Input data (Which is as per the base paper) these process are as a.Generative Process, b.Prior Distribution & c.Posterior Distribution. After performing these processes Putting all the process together and generates the expected Ouputs.

And finally I am getting the expected aggregated output. Due to this I will be used in Online fashion of the data (Output) Aggregation.

 DOI 10.17148/IJARCCE.2015.4632

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

IJARCCE

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 6, June 2015

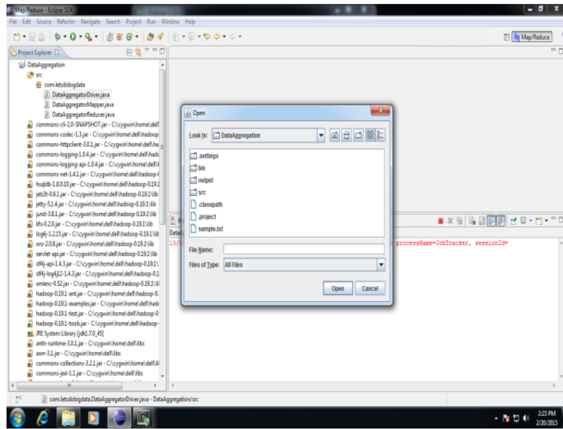

Fig. 6. File Selection

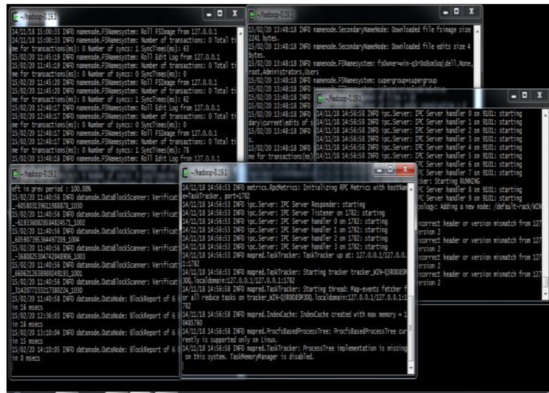

Fig. 7. Expected Aggregated Output.

V. CONCLUSION & FUTURE WORK

Like the earlier works on Online Aggregation,I focus on single table query plans involving "Group By" aggregations, which is precisely the workload targeted by Map-Reduce. The focus of our work here is to develop a model that accounts for biases that can arise when estimating aggregates in a cluster environment. This model allows us to export "early returns"of query aggregates that are statistically robust. Cloud-based data management systems are emerging as scalable, fault-tolerant, and efficient solutions that manages large volumes of data with cost effective infrastructures.It is an attractive solution to provide a quick sketch of massive data before a long wait of the final accurate query result. The main benefit of OLA is that if we get an acceptably accurate answer within a fraction of time , then we can abort the query execution thus, saving significant computer and human time.

Locality scheduling in the context of online aggregation is a major issue that needs to handle in future. Scheduling computation near the data is the primary optimization in today's Map-Reduce Systems. Further, we would also like to consider external constraints on the scheduler. For example, we may wish to schedule only those tasks from the highest priority jobs.

ACKNOWLEDGMENT

I would like to thank **Miss. Rekha Jadhav** for her valuable guidance and supporting to Implements the 'Using Map-Reduce performing Online Aggregation'.

REFERENCES

[1] N. Pansare, V. R. Borkar, C. Jermaine, and T. Condie. Online aggregation for large map-reduce jobs. In VLDB 2011 Conference Proceedings, pages 1135–1145, August 2011.

[2] Vinayak Borkar, Michael Carey, Raman Grover, Nicola Onose, Rares Vernica Hyracks: A Flexible and Extensible Foundation for Data-Intensive Computing Proc. IEEE 27th International Conference on Data Engineering (ICDE) Hanover, Germany (2011), pp. 1151–1162.

[3] Wang YX, Luo JZ, Song AB . Partition-based online aggregation with shared sampling in the cloud. Journal of computer science and technology28(6): 989{1011 Nov. 2013. DOI 10.1007/S11390-013-1393

[4] T. Condie, N. Conway, P. Alvaro, J. Hellerstein, K. Elmeleegy, and R. Sears. Mapreduce online. In *NSDI Conference*, pages 21–21, 2010.

[5] Yuxiang Wang , Junzhou Luo ,Aibo Song ,Fang Dong OATS: online aggregation with two-level sharing strategy in cloud Distrib Parallel Databases (2014) 32:467–505 DOI 10.1007/s10619-014-7141-2.

[6] Qin, C., Rusu, F. Parallel online aggregation in action. In: Proceedings of the 25th International Conference on Scientific and Statistical Database Management (SSDBM), pp. 46–49, 2013.

[7] Yantao Gan, Xiaofeng Meng, Yingjie Shi COLA: A Cloud-Based System for Online Aggregation Data Engineering(ICDE),2013 IEEE 29th International conf, Pages 1368-1371,April-2013.

[8] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online aggregation. In SIGMOD 1997 Conference Proceedings, pages 171–182, May 1997.

[9] P. J. Haas and J. M. Hellerstein. Ripple joins for online aggregation. In SIGMOD 1999 Conference Proceedings, pages 287–298, June 1999.

[10] G. Luo, C. J. Ellmann, P. J. Haas, and J. F. Naughton. A scalable hash ripple join algorithm.In SIGMOD 2002 Conference Proceedings, pages 252–262, June 2002.

BIOGRAPHIES

M Sabir Chauhan received his B.E. degree in Computer Engineering from Amravati University, Maharashtra, India, in 2012.Currently obtaining M.E degree in Computer from GHRIET College of engineering, Wagholi,Pune. His research interests include HADOOP.

Rekha Jadhav received her B.E. & M.E. degree in Information Technology from Pune University, Maharashtra, India.