A Survey on Online Aggregation For Large Mapreduce Jobs

M SABIR M SHABBIR, REKHA JADHAV

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. MapReduce programming approach have close relationship with cloud computing. Today, online aggregation is a very attractive technology. In this paper, we have described how online aggregation can be built into a MapReduce system for large-scale data processing. This paper also describes the detail implementation of OLA models in Hyracks . 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, we have presented advantages and limitation of OLA.**

Index Term: **COLA, Hadoop, Hyracks Online Prototype, MapReduce, OLACloud online aggregation.**

I. INTRODUCTION:

When we are running online aggregation (OLA) 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, MapReduce 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 MapReduce jobs.

Traditional MapReduce 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 MapReduce 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.

M SABIR M SHABBIR *ME (Computer Engineering) Savitribai Phule University of Pune, Maharashtra(INDIA) Pune, Maharashtra(INDIA), 09975422328*
REKAHA JADHAV *M.E.(Computer). Asst.Prof. GHRIET pune, Maharashtra(INDIA}09970082182.*

.

.

II. WORKING OF OLA IN HYRACKS

A. System Architecture of Hyracks:

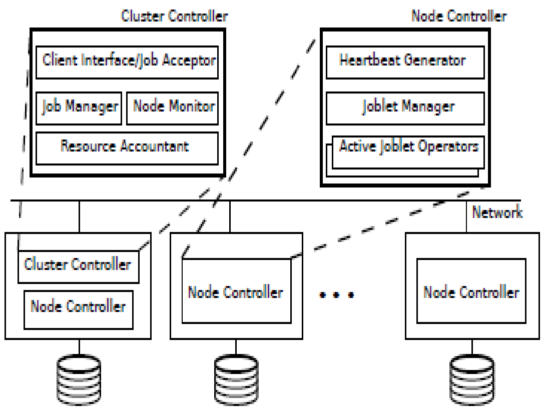

Figure 2.1. System Architecture of a Hyracks

Figure 2.1 describes an overview of the basic architecture of a Hyracks installation. Every Hyracks cluster is managed by a Cluster Controller process. The Cluster Controller accepts job execution requests from clients, plans their evaluation strategies and then schedules the jobs' tasks (stage by stage) to run on selected machines in the cluster. Also, it is responsible for monitoring the state of the cluster to keep track of the resource loads at the various worker machines. The Cluster Controller is also responsible for re-planning and re-executing some or all of the tasks of a job in the event of a failure. Turning to the task execution side, each worker machine that participates in a Hyracks cluster runs a Node Controller process. The Node Controller accepts task execution requests from the Cluster Controller and also reports on its health (e.g., resource usage levels) via a heartbeat mechanism.

B.Implementation of OLA In Hyracks :

In this section we are going to describe implementation of the OLA model in Hyracks [2]. Hyracks is a new open source project that supports map and reduce operations, along with higher level relational operations such as 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 , it is used to store the input to the map operation and the output generated from the reduce operation. Here, when a client submits a MapReduce 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 proposed approach author modified the Hyracks implementation in two ways.

1. ***Creat 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

2. ***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 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. Finally, we include the time when the estimator is called on the reduce task. The reduce task executes the estimator when it receives a new map output. The location of all data and meta-data files received thus far is given to the estimator
when it is called. After the estimator completes, its output can be written to HDFS or forwarded to a downstream operator using the user-defined call-back class.

III. LITERATURE SURVEY:

A. Online Aggregation in the cloud (OLACloud)

In OLACloud implementation[3] , author have used Hadoop Online Prototype (HOP) as a natural candidate for the underlying query processing engine. HOP is a modified version of the original MapReduce framework, which is designed to construct a pipeline between Map and 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, author have described the data flow of OLACloud, which consists of two steps:
1) content-aware repartition with fair-allocation strategy, and
2) OLA query processing with shared sampling.

1) Content-Aware Repartition With Fair-Allocation Strategy:

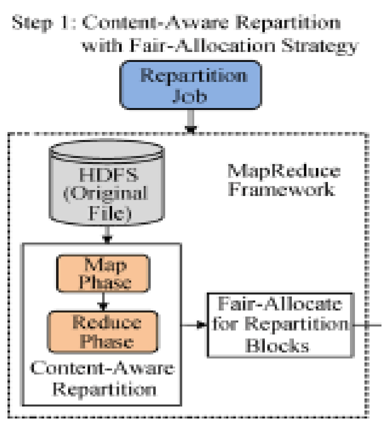
Step 1: Content-Aware Repartition with Fair-Allocation Strategy
Repartition Job
HDFS (Original File)
MapReduce Framework
Map Phase
Reduce Phase
Content-Aware Repartition
Fair-Allocate for Repartition Blocks

Figure 3.1. Content-Aware Repartition With Fair-Allocation Strategy

The first step it is nothing but a pre-processing of OLACloud, 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 .

2) OLA Query Processing With Shared Sampling.

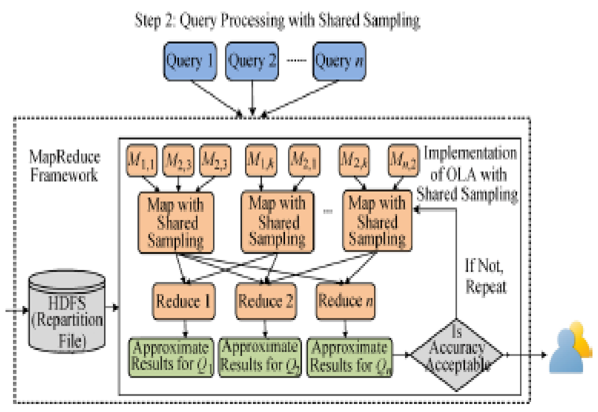

Figure 3.2. OLA Query Processing With Shared Sampling.

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 OLACloud 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 a desired accuracy is reached and the users can stop the query early before its completion.

B. Pipelined Hadoop Implementation To Support Online Aggregation Within A Single Job)
And Between Multiple Jobs

1.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.
.

Progress matrix
Hadoop provides support for monitoring the progress of task executions. As each map task executes, it is assigned a progress score in the range [0,1], based on how much of its input the map task has consumed. Authtor, reused this feature to determine how much progress is represented by the current input to a reduce task, and hence to decide when a new snapshot should be taken.

2. Multi-Job Online Aggregation

Online aggregation is particularly useful when it is applied to a long-running analysis task consist of multiple MapReduce 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 MapReduce 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.
Limitations :

1The output of a reduce function is not "monotonic": the output of a reduce function on the first 50% of the input data may not be obviously related to the output of the reduce function on the first 25%. Thus, as new snapshots are produced by job_1 , job_2 must be recomputed from scratch using the new snapshot. As with inter-job pipelining , this could be optimized for reduce functions that are declared to be distributive or algebraic aggregates.

C. *Online Aggregation With Two-Level Sharing Strategy In Cloud* **(OATS)**

1.System Architecture OATS:

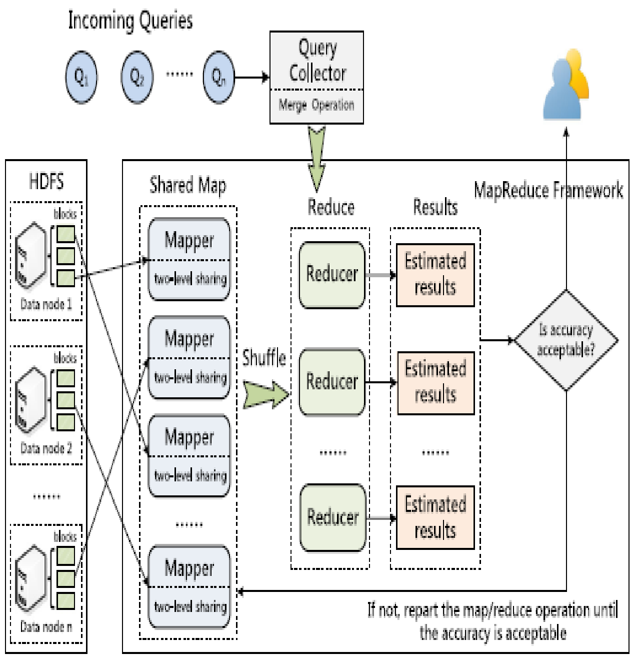

Figure 3.3:System Architecture of OATS

Above figure 3.3 describes the architecture and workflow of OATS[5] in the cloud. Here, the input data is organized in blocks by the content-aware pre-processing technique, which means these blocks have uniform intervals but with unequal block size, and distributed over different data node for storage. Given the incoming OLA queries, OATS does not execute them immediately but first collects them and analyses the potential sharing opportunities by the component of Query Collector . Then the set of incoming queries are combined as a grouped dynamic MapReduce job. Then ,this grouped job is decomposed into a series of map tasks, which are assigned to the data nodes . Afterwards, OATS performs the two-level sharing strategy according to the sharing information of each mapper in the map phase to provide the sharing for sampling and computation, that eliminates the redundant disk I/O cost and reducing the repetitive statistical computation cost. Lastly, the reduce phase estimates the approximate result for each involved shared OLA . If the accuracy obtained is unsatisfactory, the above Map/Reduce phases will be repeated to retrieve samples continuously, calculate the statistics incrementally and update the approximate result progressively. The final result is returned to the user when a desired accuracy is reached and the user can stop the job early before its completion.

Advantages and Limitation of OATS:
1.OATS is the first work on studying the sharing issues of OLA in the cloud. OATS can produce acceptable approximate results within a time period two orders of magnitude shorter compared to the one with exact results, and is on average 80% more efficiency than the OLA-based solution without sharing operation for different data size and data distribution.
2. Besides the sharing issue of online aggregation in the cloud, there exists some other problems that needs to be handled in order to improve the OLA performance such as the estimation failure of OLA due to the unbiased sample set etc.

D. Parallel Online Aggregation

In this work [6], author have described framework for online aggregation based on a parallel system for the efficient execution of complex analytical queries over terabytes of data. In this work , author have also presented different approaches to provide online aggregation in a parallel cluster environment where data are partitioned across processing nodes..

3.1 Partial Aggregation

The first requirement in any online aggregation system is a mechanism to compute partial aggregates over some portion of the data. Partial aggregates are typically a superset of the query result since they have to contain additional data required for estimation. The partial aggregation mechanism can take two forms.
1.Fix the subset of the data used in partial aggregation and execute a normal query.
2. Make the interfere with aggregate computation over the entire dataset to extract partial results before the computation is completed. The first alternative corresponds to **iterative online aggregation**, while the second to **overlapped execution**.

Aggregate evaluation takes two forms in parallel databases. Which are described in following section.
centralized approach: Here, all the partial aggregates are sent to a common node – the coordinator – that is further aggregating them to produce the final result. As an intermediate step, local aggregates can be first combined together and then its result sent to the coordinator.
parallel approach: In this approach, the nodes are first organized into an aggregation tree. Each node is responsible for aggregating its local data and the data of its children. The process is executed level by level starting from the leaves node, and then final result will be computed at the root node of the tree. The benefit of the parallel approach is that it also parallelizes the aggregation of the partial results across all the nodes rather than burdening a single node (with data and computation). The drawback is that in the case of a node failure it is likely that more data are lost .

Iterative Online Aggregation

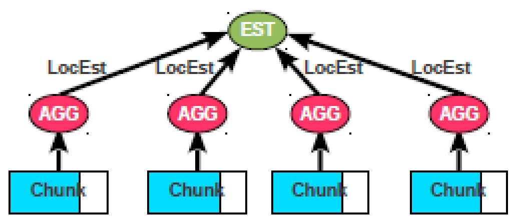

Figure 3.4: Iterative Online Aggregation

In Iterative Online Aggregation, when a sample of the desired size is obtained, partial aggregation is executed across all the partitions using one of the strategies for parallel aggregation. If the estimate computed over the partial aggregate is not accurate enough, a new iteration can be executed. The linear scan continues from the position where it stopped before and the partial aggregate is updated incrementally.

Overlapped Online Aggregation

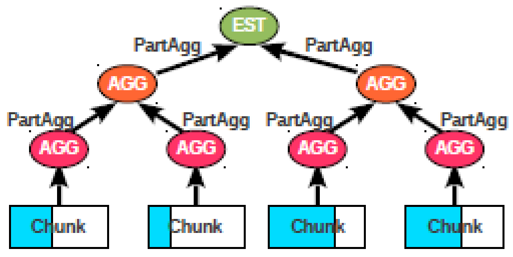

Figure 3.5: Overlapped Online Aggregation

In the overlapped execution strategy , asynchronous linear scans at each partition progress independently. When an estimate request is triggered, partial aggregation across all the partitions is executed. The resulting estimate can be used to stop the final result computation which proceeds normally at all times. If no decision is taken, then the aggregation continues until all the data are processed and the final result is computed.

E.COLA: A Cloud-Based System for Online Aggregation
Yantao Gan, Xiaofeng Meng, Yingjie Shi

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.

1. **COLA System Architecture And Implementation**

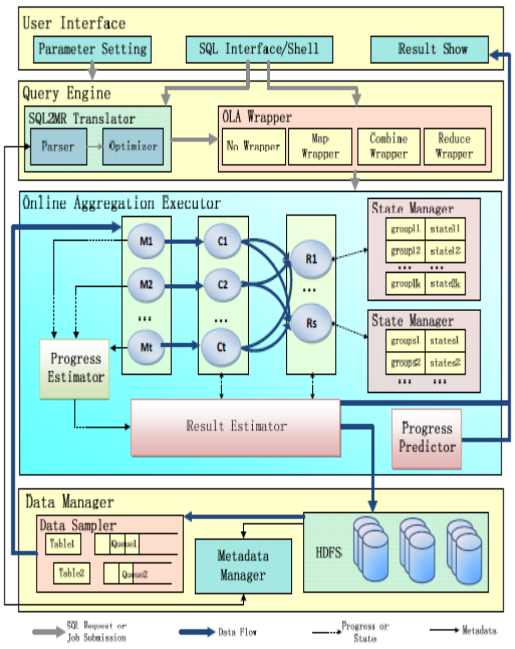

Figure 3.6:System architecture of COLA.

,

Above Figure 3.6 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 the User Interface. The Query engine serves as a translator that transforms SQL queries into MapReduce 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 MapReduce jobs in online fashion and reports the estimates back to the client.

A. User Interface

COLA provides interactive and flexible interfaces, users can issue SQL query request through SQL interface or submit MapReduce 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.

B. Query Engine

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

C. Online Aggregation Executor

The Online Aggregation Executor is the key module of COLA to perform online query processing algorithm over MapReduce. 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.

D. 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.

IV.ADVANTAGES AND LIMITATION OF OLA

A.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.

B.Limitation of OLA:

1) Sampling efficiency is low due to the lack of consideration of skewed data distribution for online aggregation in MapReduce.
2) It increases the I/O cost of online aggregation due to the independent job execution mechanism of MapReduce.

V.CONCLUSION

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 need to handle in future. Scheduling computation near the data is the primary optimization in today's MapReduce
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.

VI.REFERENCES:

[1] N. Pansare, V. R. Borkar, C. Jermaine, and T. Condie. Online aggregation for large mapreduce 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/S11390-013-1393-6

[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.

[11] S. Wu, S. Jiang, B. C. Ooi, and K. L. Tan. Distributed online aggregation. In VLDB 2009 Conference Proceedings, pages 443–454, August 2009.

[12] Bose JH, Andrzejak A, Hogqvist M. Beyond online aggregation: Parallel and incremental data mining with online Map-Reduce. Proceedings of the workshop on massive data analytics on the cloud, Raleigh, 2010.