A Survey of Online Aggregation in the Cloud

Yingjie Shi, Yantao Gan, Xiaofeng Meng

Abstract: Online aggregation in the cloud makes it possible to save cost by taking acceptable approximate early answers. There are critical requirements to support online aggregation in the cloud, including the statistical methods to support sampling of data and confidence estimation of approximate results, and the query processing framework to support incremental and continuous computing of aggregations. In this paper, we propose the system architecture of online aggregation in the cloud, and survey the existing works in the corresponding area from three aspects. At last, we give the challenging problems and future work.

Keywords: online aggregation, cloud computing, MapReduce

1. Introduction

As the development of information industry, massive data has been produced in various applications, which include online transaction data, web access logs, sensor data, scientific data, etc. According to the IDC white book, Digital Universe will be 35 Zettabytes by 2020[1], we can say that the age of big data has come. In the special issue on big data, Nature indicates that distilling the meaning from big data has never been in such urgent demand[2]. Because of the volume and variety of big data, the traditional RDBMS is restricted by the scalability and cost when managing big data. Mean while, cloud data management systems provide highly efficient and cost effective solutions for massive data analytics, and applications are gradually migrated to the cloud environment. However, though query processing is normally paralleled in the cloud, due to the data volumes involved, it often takes long time for an aggregation query to return the final results. Since many large-scale aggregation queries are used to get a sketch or “big picture” of the data, one promising approach to address this problem is online aggregation (OLA)[3], which can continuously provide “early returns” with estimated confidence intervals. As more data is processed, the estimate is progressively refined and the confidence interval is narrowed until the precise result is produced.

OLA was initially proposed in the area of relation DBMS in 1997[3], and after that many research findings have been published in the academic area. However, online aggregation has less commercial impact in DBMS market and there are few commercial RDBMSs supporting OLA. Generally speaking, there are two main reasons for the lack of adoption[4]. First, implementing OLA within a database engine would likely require extensive changes to the database kernel. OLA requires some sort of statistically quantifiable randomness within the database engine. Since this would require significant changes to most kernels and would wreak havoc with techniques widely-implemented by database vendors, vendors and kernel developers have justifiably viewed OLA with suspicion. Secondly, the payment model of RDBMS is “pay-before-you-go”, the goal of saving human and computer time has never been as compelling as one might think. In the cloud computing environment, things have changed, and OLA is of more paramount importance due to

the unique characteristics of the cloud. First, the payment model of cloud computing is "pay-as-you-go", users just pay for what they consume. Reducing query processing time will lead to immediate saving of cost, for example, obtaining a result with 95% confidence level in much less time could be very attractive. Secondly, queries in the cloud is always composed of many subtasks executing on different nodes, OLA could help to increase the parallelism degree and resource utilization by processing parallel subtasks in online mode. Thirdly, OLA could reduce or eliminate performance bottleneck due to slowest nodes. Since the cloud is typically a heterogeneous environment consisting of many nodes with diverse hardware setup and performance[5], users' waiting time of a query is significantly influenced by the sub-task in the slowest node. In an OLA system, the data flow between sub-tasks is pipelined and the estimate result is refined continuously, thus the performance "tail" effect could be alleviated or eliminated.

Though online aggregation techniques have been extensively studied for single-site relational database, there are many challenges to adapt them to the cloud. First of all, the distributed environment of the cloud brings up the problems of processing concurrency, data distribution, data skew and other issues that must be accounted for during query processing and statistical computing. In particular, for aggregations over joining results of multiple tables in such environment, how to process the queries in online mode to minimize the time until an estimate is obtained with acceptable precision requires sophisticated considerations. In addition, data in the cloud is typically organized and processed in blocks, which could be thousands times larger than those in traditional file systems[6,7]. During the statistics estimate procedure in OLA, uniform-random sampling is of theoretical significance. However, taking a uniform-random sample can be inefficient for distributed data in such organization. For example, if data is blocked on an attribute associated with the aggregation, in the worst case scenario, every sampling step can be no faster than a full scan of the data. To effectively build statistical estimators with blocked data, a major challenge is to guarantee the accuracy of the estimators while leveraging sampling as efficiently as possible. Last but not least, the typical batch processing mode of the cloud does not match the requirements of online aggregation processing, where "early returns" are generated before all the data is processed. For example, for MapReduce[8], the entire output of each map and reduce task is materialized to a local file before it can be consumed by the next stage. The operators cannot begin until their precursor operators finish.

Motivated by the requirements and challenges, we propose a system architecture for online aggregation in the cloud. Based on the system architecture, we survey the research work from the following aspects: the online processing of aggregate queries, the estimate of query progress, and the statistical information collection. The rest of this paper is organized as follows. In Section2, we give the framework of online processing of aggregations in the cloud. In Section3, we summarize and compare the key techniques of online aggregation from three aspects. Section4 discusses the open challenging problems of online aggregation in the cloud, followed by the conclusions.

2. Framework of OLA in the Cloud

Two major challenges of online aggregation in the cloud are: how to produce incremental and continuous aggregation computing to provide early returns in cloud computing architecture, and how to provide effective sampling and estimation of confidence of early returned results. Implementing online aggregation in the cloud requires extensions to the traditional MapReduce

framework due to the mismatch of the requirements of online aggregation and the distributed computing mode of MapReduce. First, online aggregation needs to provide “early returns”, so it requires that the intermediate results of all operators should be pipelined during the query processing. However, the traditional MapReduce framework is batch-oriented, which means that a consuming operator cannot begin its function until its producing operators finish. HOP[9] is a modified version of the Hadoop MapReduce framework to allow data to be pipelined within a MapReduce job and between jobs, and it provides a foundation platform for OLA in the cloud. Secondly, in order to make efficient estimates, the online queries should process sample data instead of sequential data stream from data nodes. So efficient sampling module should be designed in the framework of OLA in the cloud.

Based on the requirements of implementing online aggregation in the cloud and the characteristics of cloud computing, we propose a system architecture for OLA in the cloud, which is depicted in Figure1. There are five modules in the architecture: Front-End Interface, Query Processing Module, OLA Processing Module, Data Storage Module and Service Management Module. Every module is responsible for its special functions.

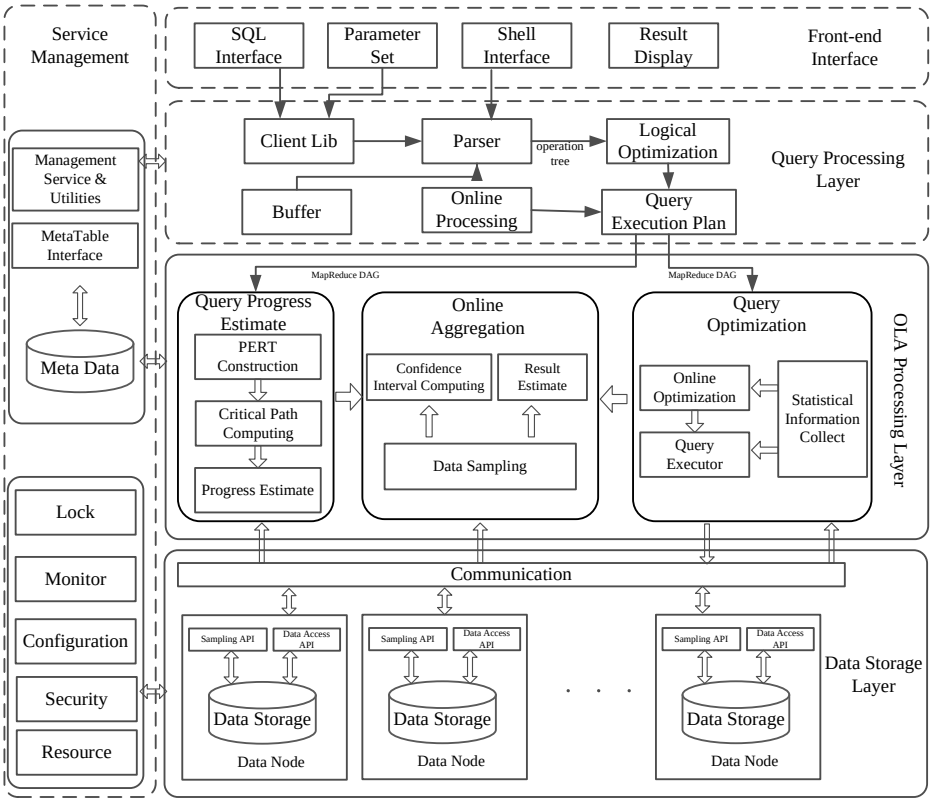

Figure1. The Architecture of OLA in the cloud

- **Front-End Interface:** This module provides both SQL interface and shell interface for the users, so users can input SQL commands and MapReduce shells. They can also set the parameters of OLA through the interface, such as confidence interval, update frequency, etc. The estimate results are displayed to the users.
- **Query Processing Module:** This module receives the SQL commands from the upper layer, it conducts the SQL parsing and logical optimization on the query commands, and generates the query execution plan. Then the online processing is executed on the execution plan to make it support online aggregation.

- **OLA Processing Module:** The online aggregation is processed in this module, which contains three sub-modules: Online Aggregation, Query Progress Estimate and Query Optimization. The online aggregation module conducts result estimate and confidence interval computing on the sampled data. And the other two sub-modules provide basic support: the query optimization conducts online optimization on the execution plan composed of MapReduce DAG, and the query progress estimate module provides the remaining time of query.
- **Data Storage Module:** This module is responsible for distributed data storage, it provides the sampling API and query execution API for the upper layer. It also supports the following features of data storage: replication, parallelism, fault tolerance, key partitioning and synchronization.
- **Service Management Module:** The service management module is responsible for metadata management, operation management and system monitor. The metadata includes the table schema information and the data distribution information used for logical optimization. It also gathers monitoring information such as resource provision, system health, data deployment, etc.

The OLA processing module contains the core techniques for online aggregation in the cloud. This key module contains three sub-modules, and in the following of this paper, we survey the related work from these three aspects.

3. Key Techniques

Based on the system architecture of online aggregation in the cloud, we can see that there are three main problems in the area: online processing of aggregate queries in the cloud, estimating the progress of queries in the cloud, and collecting statistical information of data distribution. These problems are studied to different degrees, in this section we summarize and analyze the existing works.

3.1 Processing of Online Aggregation

Online aggregation was first proposed in the in 1997[3], which focused on single-table queries involving “group by” aggregations. After that extensive studies are conducted in the field of online aggregation in relational DBMS, nowadays, with the development of cloud computing, OLA emerges as a new research area for the cloud and is of more paramount importance due to the unique characteristics of the cloud. In this section, we surveyed the related work from relational DBMS and cloud-based DBMS respectively.

3.1.1 OLA in RDBMS

According to the system architecture in Figure1, there are three parts in the module of online aggregation processing: data sampling, result estimate and confidence interval computing. The data sampling mechanism is very important for online aggregation, and it also determines the algorithms of computing confidence interval. We summarize the existing works of OLA in RDBMS from two aspects: the sampling mechanism and aggregation type, which is shown in Figure2. The uniform sampling adopts tuple as the sampling unit and it guarantees that the possibility of every tuple to be selected is equal, which is called simple random sample in the statistical terms. Tuple-level sampling provides true uniform-randomness, which is the basis of many approximate algorithms. The initial works of OLA focus on the online query processing and

the statistical computing, they adopt uniform random sampling[3,10]. However, the uniform random sampling can be very expensive because data is always clustered by blocks or pages. Many researchers study the complex sampling mechanism. [11] analyzed the impact of block-level sampling on statistic estimation for histogram construct and distinct-value estimate, and the authors also developed the corresponding statistical estimators with block-level samplings. [12] provided a bi-level sampling scheme that combines the rowlevel and page-level sampling methods. In the field of distributed DBMS, there is also some existing work. [13] compared the accuracy and efficiency of different sampling methods for query size estimation in the parallel DBMS: stratified random sampling and simple random sampling with the unit of row-level and page-level.

The online aggregation on joined tables is much more difficult than single tables. [14] presented a family of join algorithms called ripple joins based on the traditional block nested-loops and hash joins, which are designed to minimize the time until an acceptably precise estimate result is available. [15] extended the original ripple joins to speed up the convergence by parallelizing the query processing and sampling, however, it could not provide statistical guarantee when the exact data distribution is unknown or the memory overflows. In order to make query estimate and maintain probabilistic confidence bounds no matter whether the inputs fit in memory or not, [16] combined the traditional sort-merge join and ripple join algorithms, and the authors added a shrink phase to the query processing which runs concurrently with the merge phase to update the estimate result and the confidence interval. Wu et al. extended the online aggregation to the distributed environment which is maintained in a distributed hash table network [17].

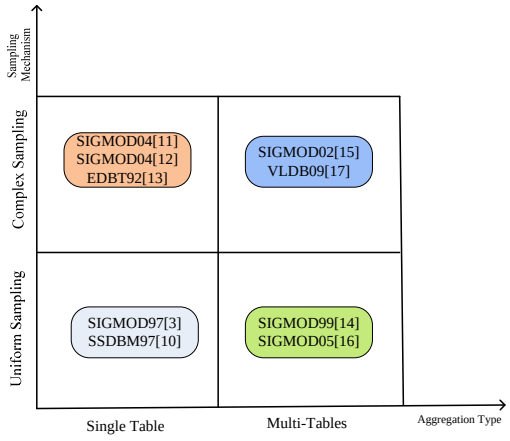

Figure2. OLA in RDBMS

3.1.2 OLA in the Cloud

Nowadays, online aggregation is renewed in the context of cloud computing, and some studies have been conducted based on MapReduce[4,9,18]. [9] pipelined the MapReduce processing of Hadoop and proposed a system called Hadoop Online Prototye(HOP), which allows posterior operators consume the output of precursor operators before the precursor operators complete. HOP can provide the original snapshots of the MapReduce jobs at data dependent intervals, and it supported OLA by scaling up the snapshots with the job progress without any confidence bounds of the query estimate. [18] provides another platform for online processing of MapReduce through memory sharing, it also provides incremental computing during the processing. The framework proposed in [18] focuses on operations with iterative computing, such as the principal component

analysis, k-means, etc. However, the scalability is limited because of the shared-memory architecture. Pansare et al implements OLA over MapReduce based on Bayesian framework[4]. The authors considered the correlation between aggregate value of each block and the processing time, so they took into account the scheduling time and processing time of each block as observed data during the estimate processing. This paper focused on single table's aggregate query containing one MapReduce job, without considering aggregate queries over join results of multiple tables, which contains several MapReduce jobs.

Table1. OLA in the cloud

Existing Work	**Sampling Mechanism**	**Statistical Theory**	**Aggregation Type**	**Pros & Cons**
HOP[9]	Sequential Reading	null	single table multi-tables	**Pros**: Provides the platform for OLA **Cons**: Returns the snapshots directly without result estimate and confidence interval
Incremental Data Mining with MR[18]	Sequential Reading	cross-validation	iterative aggregation	**Pros**: Provides the platform for iterative computing with simple architecture **Cons**: The architecture is non-scalable, the convergence estimation is not accurate as interval
OLA over MR[4]	Uniform Random	Bayesian	single table	**Pros**: Provide result estimate and confidence interval for single table aggregations **Cons**: Does not support aggregations requiring multiple Mapreduce jobs

3.2 Estimating the Progress of Queries in the Cloud

In this section, we also summarize the woks from two areas. First is the area of estimating the progress of SQL queries on single-node DBMS. [19] separates a SQL query plan into pipelined segments, which is defined by blocking operators. The query progress is measured in terms of the percentage of input processed by each of these segments. Its sub-sequent work [20] widens the class of SQL queries the estimator supports, and it increases the estimate accuracy by defining segments at a finer granularity. [21] decomposes a query plan into a number of pipelines, and computes the query progress through the total number of getnext() calls made over all operators. [22] characterize the query progress estimation problem introduced in [19] and [21] so as to understand what the important parameters are and under what situations we can expect to have a robust estimation of such progress. The work of this area focuses on partitioning a complex query plan into segments(pipelines) and collecting the statistics on cardinality information to compute the query progress. These techniques are helpful for estimating the progress of queries running on a single node. However, they do not account for the challenges brought by query parallelization, such as parallelism and data distribution.

In the area of cloud computing, there also exists initial work for progress estimate of MapReduce jobs. We can classify the related work into two categories: estimating the progress of

tasks during one MapReduce job, and estimating the progress of MapReduce pipelines. Estimating the query progress in the cloud has to face several challenging problems because of the characteristics of cloud. We list the challenging problems in Table2, and summarize the existing work in this area. [5] estimates the time left for a task based on the progress score provided by Hadoop. This paper focuses on task scheduling in MapReduce based on the *longest approximate time to end* of every task, so it orders the task by their remaining times. It computes the progress rate through the progress score and the elapsed time of the task execution, and computes the remaining time of a task based on the progress rate. [5] provides a method to estimate the progress of a MapReduce task, however, there are also several challenges to estimate the progress of MapReduce jobs and MapReduce DAGs. Parallax[23] estimates the progress of queries translated into sequences of MapReduce jobs. It breaks a MapReduce job into pipelines, which are groups of interconnected operators that execute simultaneously. Parallax estimates the remaining time by summing the expected remaining time across all pipelines. It addresses the challenges of parallelism and variable execution speeds, without considering concurrent workloads. ParaTimer[24] extends Parallax to estimate queries translated into MapReduce DAGs. It estimates the progress of concurrent workloads through a critical path based on task rounds, which works well in a *homogeneous* environment. ParaTimer handles task failure through comprehensive estimation, which provides an upper bound on the query time in case of failure. However, ParaTimer assumes only one worst-case failure before the end of the execution, and it has to repeat all the steps in the estimate algorithm to adjust to failures. It may become inefficient when failures are very common, which is one characteristic of cloud. None of the above work handles heterogeneity in the estimation, which is also an important characteristic of cloud.

Table2. Progress Estimate in the Cloud

	Task Parallelism	Variable Speed	Concurrent Workloads	Data Skew	Cluster Heterogeneity	Task Failure	Node Failure	Cluster Scale
LATE[5]	N	N	N	N	N	N	N	Y
Parallex[23]	Y	Y	N	N	N	N	N	Y
ParaTimer[24]	Y	Y	Y	Y	N	N	N	N

3.3 Collecting Statistical Information

The optimization of online query processing is of great importance for OLA in the cloud, which focuses on minimizing the time until an acceptably precise estimate of the query result is available. Summary of data distribution and statistical information provide fundamental reference to support query optimization, and histograms are of particular interest. First of all, histograms provide reference information for selecting the most efficient query execution plan. A large fraction of queries in the cloud are implemented in MapReduce[8], which integrates parallelism, scalability, fault tolerance and load balance into a simple framework. For a given query, there are always different execution plans in MapReduce. For example, in order to conduct log processing which joins the reference table and log table, [25] proposed four different MapReduce execution plans applicable to different data distributions. However, how to select the most efficient execution plan adaptively is not addressed, and histogram can offer great guidance to this problem. Secondly,

histograms contain basic statistics useful for load balancing. Load balance is crucial to the performance of query in the cloud, which is always processed in parallel. In the processing framework of MapReduce, output results of the mappers are partitioned to different reducers by hashing their keys. If data skew on the key is obvious, then load imbalance is brought into the reducers and consequently results in degraded query performance. Histograms constructed on the partition key help to design the hash function to guarantee load balance. Thirdly, in the processing of joins, summarizing the data distribution in histogram is useful for reducing the data transmission cost, which is one of the scarce resources in the cloud. Utilizing the histogram constructed on join key can help prevent sending the tuples that do not satisfy the join predicate to the same node[26]. In this section, we summarize the related work of histogram estimate in the cloud.

[27] surveyed the history of histogram and its comprehensive applications in the data management systems. There are different kinds of histograms based on the constructing way, [28] conducted a systematic study of various histograms and their properties. Constructing the approximate histogram based on sampled data is an efficient way to reflect the data distribution and summarize the contents of large tables, it's proposed in [29] and studied extensively in the field of single-node data management systems. The key problem has to be solved when constructing approximate histogram is to determine the required sample size based on the desired estimation error. We can classify the works into two categories by the sampling mechanisms. Approaches in the first category adopted uniform random sampling[30-32], which samples the data with tuple level. [30] focused on sampling-based approach for incremental maintenance of approximate histograms, and it also computed the bound of required sample size based on a uniform random sample of the tuples in a relation. Surajit et al. further discussed the relationship of sample size and desired error in equi-depth histogram, they proposed a stronger bound which lends to ease of use[31]. [32] discussed how to adapt the analysis of [31] to other kinds of histograms. The above approaches assumed uniform random sampling. Approaches of the second category constructed histograms through block-level sampling[11,31]. [31] adopted an iterative cross-validation based approach to estimate the histogram with specified error, the sample size was doubled once the estimate result didn't arrive at the desired accuracy. [11] proposed a two-phase sampling method based on cross-validation, in which the sample size required was determined based on the initial statistical information of the first phase. This approach reduced the number of iterations to compute the final sample size, and consequently processing overhead. However, the tuples it sampled may be bigger than the required sample size because cross-validation requires additional data to compute the error. All the above techniques focus on histogram estimating in the single-node DBMS, adapting them to the cloud environment requires sophisticated considerations.

There is less work on constructing the histogram in the cloud. Authors of [33] focused on processing theta-joins in the MapReduce framework, they adopted the histogram built on join key to find the "empty" regions in the matrix of the cartesian product. The histogram on the join key was built by scanning the whole table, which is expensive for big data. Authors of [34] proposed a method for constructing approximate wavelet histograms over the MapReduce framework. Approach of this paper retrieved tuples from every block randomly and sent the outputs of mappers at certain probability, which aimed to provide unbiased estimate for wavelet histograms and reduce the communication cost. The number of map tasks is not reduced because all the

blocks have to be processed, and the sum of start time of all the map tasks cannot be ignored. Though block locality is considered during the task scheduling of MapReduce framework, there exist blocks that have to be transferred from remote node to the mapper processing node, we believe there is room to reduce this data transmission cost. [35] proposed the histogram estimator called HEDC, which focused on estimating the equi-width histograms of data in the cloud through an extended MapReduce framework. HEDC adopts a two-phase sampling mechanism to leverage the sampling efficiency and estimate accuracy, and constructs the approximate histogram with desired accuracy.

Table3. Histogram Estimators for Data in the Cloud

Histogram Construct in the Cloud	**Sampling Mechanism**	**Histogram Type**	**Pros & Cons**
Histogram estimator for theta-joins[33]	Uniform Random Sampling	Equi-Depth Histogram	**Pros:** Easy to implement the estimator. **Cons:** Sampling mechanism is not efficient; Do not provide the error bound.
Wavelet histogram estimator[34]	Uniform Random Sampling	Wavelet Histogram	**Pros:** Provide unbiased estimate of the histogram and reduce the communication cost. **Cons:** Number of map tasks is not reduced.
HEDC[35]	Two-phase Sampling	Equi-Width Histogram	**Pros:** Provide sampling mechanism to leverage the sampling efficiency and bound the error. **Cons:** The equi-width histogram is used less than equi-depth histogram.

4. Future Work

Online aggregation emerges as a new research area for the cloud and is of paramount importance due to the unique characteristics of the cloud. In this paper, we propose the architecture of OLA in the cloud and survey the existing works. Generally speaking, the research on this area is at early stages, there exist several challenging problems:

(1) Online Query Processing Optimization: The optimization of traditional offline queries focuses on minimizing the time until the time to completion of the query, while the optimization of online queries focuses on minimizing the time until a precise estimate of the query result is available. The existing work focuses on the processing optimization, which includes the scheduling optimization, the incremental computing, data placement, parameter setting, etc[36-39]. However, all the optimizations are based on rules, we believe that optimizing based on cost is more significant and challenging.

(2) Flexible Sampling Mechanism: Sampling is an important part of online aggregation. Because of the characteristics of data organization in the cloud, the sampling mechanism of OLA in the cloud is different from that of relational DBMS. The existing work in the area always conducts simple random sampling, which can be inefficient in the cloud environment. How to design efficient sampling mechanisms for different aggregation types is of great significance for online aggregation in the cloud.

(3) Online Processing of Complex Aggregations: The exiting work of OLA in the cloud provides solutions for simple aggregations, which contain only one MapReduce job.

However, there are many complex aggregations such as aggregations over multiple joined tables, top-k queries, etc, which contain several MapRedcue jobs. The online processing of the complex aggregations is more challenging.

5. Conclusions

Cloud-based data management systems are emerging as scalable, fault-tolerant, and efficient solutions to manage large volumes of data with cost effective infrastructures, and more and more data analysis applications are migrated to the cloud. As an attractive solution to provide a quick sketch of massive data before a long wait of the final accurate query result, online processing of aggregate queries in the cloud is of paramount importance. In this paper, we propose the system architecture of online aggregation in the cloud, and surveyed the existing work based on the architecture from three aspects. Generally speaking, the research of online aggregation in the cloud is at the initial stage, it brings both opportunities and challenges for researchers.

Reference

[1] Gantz J and Reinsel D. Extract ing Value f rom Chaos Technical Report White paper, International Data Corporation (IDC) Sponsored by EMC Corporation, 2011.

[2] Felice Franke and Rosalind Reid. Big data: Distilling meaning from data. Nature, 2008, 455(4):30.

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

[4] 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.

[5] M. Zaharia, A. Konwinski, A. Joseph, R. Katz, and I. Stoica, “Improving mapreduce performance in heterogeneous environments,” in Proc. The OSDI2008 Conference Proceedings, pages29–42, Dec 2008.

[6] S. Ghemawat, H. Gobioff, and S.-T. Leung. The google file system. In SOSP 2003 onference Proceedings, pages 29–43, October 2003.

[7] HDFS. Avaiable at http://hadoop.apache.org/hdfs/.

[8] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. In OSDI 2004 Conference Proceedings, pages 137–150, Dec 2004.

[9] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, J. Gerth, J. Talbot, K. Elmeleegy, and R. Sears. Online aggregation and continuous query support in mapreduce. In SIGMOD 2010 Conference Proceedings, pages 1115–1118, June 2010.

[10] P. J. Haas. Large-sample and deterministic confidence intervals for online aggregation. In SSDBM 1997 Conference Proceedings, pages 51–63, August 1997.

[11] S. Chaudhuri, G. Das, and U. Srivastava. Effective use of block-level sampling in statistics estimation. In SIGMOD 2004 Conference Proceedings, pages 287–298, June 2004.

[12] P. J. Haas and C. Koenig. A bi-level bernoulli scheme for database sampling. In SIGMOD 2004 Conference Proceedings, pages 275–286, June 2004.

[13] S. Seshadri and J. F. Naughton. Sampling issues in parallel database systems. In EDBT 1992 Conference Proceedings, pages 328–343, March 1992.

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

[15] 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.
[16] C. Jermaine, A. Dobra, S. Arumugam, S. Joshi, and A. Pol. A disk-based join with probabilistic guarantees. In SIGMOD 2005 Conference Proceedings, pages 563–574, June 2005.
[17] S. Wu, S. Jiang, B. C. Ooi, and K. L. Tan. Distributed online aggregation. In VLDB 2009 Conference Proceedings, pages 443–454, August 2009.
[18] 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.
[19] Luo, G., Naughton, J.F., Ellmann, C.J., Watzke, M.: Toward a progress indicator for database queries. In: 24th ACM International Conference on Management of Data, pp. 791-802. ACM Press, New York (2004)
[20] Luo, G., Naughton, J.F., Ellmann, C.J., Watzke, M.:Increasing the accuracy and coverage of SQL progress indicators. In: 21th IEEE International Conference on Data Engineering, pp.853-864. IEEE Press, Washington(2005)
[21] Chaudhuri, S., Narassaya, V., Ramamurthy, R: Estimating progress of execution for SQL queries. In: 24th ACM International Conference on Management of Data, pp. 803-814. ACM Press, New York (2004)
[22] Chaudhuri, S., Kaushik, R., Ramamurthy, R.: When can we trust progress estimators for SQL queries. In: 25th ACM International Conference on Management of Data, pp. 575-586. ACM Press, New York (2005)
[23] Morton, K., Friesen, A., Balazinska, M., Grossman, D.: Estimating the progress of MapReduce pipelines. In: 26th IEEE International Conference on Data Engineering, pp.681-684. IEEE Press, Washington(2010)
[24] Morton, K., Balazinska, M., Grossman, D.: ParaTimer: A progress indicator for mapreduce DAGs. In: 30th ACM International Conference on Management of Data, pp. 507-518. ACM Press, New York (2010)
[25] S. Blanas, J. M. Patel, V. Ercegovac, J. Rao, E. J. Shekita, and Y. Tian. A comparison of join algorithms for log processing in mapreduce. In SIGMOD 2010 Conference Proceedings, pages 975–986, June 2010.
[26] A. Okcan and M. Riedewald. Processing theta-joins using mapreduce. In SIGMOD 2011 Conference Proceedings, pages 949–960, June 2011.
[27] Y. E. Ioannidis. The history of histograms (abridged). In VLDB 2003 Conference Proceedings, pages 19–30, August 2003.
[28] V. Poosala, Y. E. Ioannidis, P. J. Haas, and E. J. Shekita. Improved histograms for selectivity estimation of range predicates. In SIGMOD 1996 Conference Proceedings, pages 294–305, June 1996.
[29] G. Piatetsky-Shapiro and C. Connell. Accurate estimation of the number of tuples satisfying a condition. In SIGMOD 1984 Conference Proceedings, pages 256–276, June 1984.
[30] P. B. Gibbons, Y. Matias, and V. Poosala. Fast incremental maintenance of approximate histograms. ACM Transactions on Database Systems, 27(3):261–298, 2002.
[31] S. Chaudhuri, R. Motwani, and V. R. Narasayya. Random sampling for histogram construction: How much is enough? In SIGMOD 1998 Conference Proceedings, pages

436–447, June 1998.
[32] S. Chaudhuri, R. Motwani, and V. R. Narasayya. Using random sampling for histogram construction. Microsoft Research Report, 1997.
[33] A. Okcan and M. Riedewald. Processing theta-joins using mapreduce. In SIGMOD 2011 Conference Proceedings, pages 949–960, June 2011.
[34] J. Jestes, K. Yi, and F. Li. Building wavelet histograms on large data in mapreduce. In VLDB 2011 Conference Proceedings, pages 109–120, August 2011.
[35] Y. Shi, X. Meng, F.Wang, and Y. Gan. HEDC: a histogram estimator for data in the cloud. In CloudDB 2012 Conference Proceedings, pages 51–58, October 2012.
[36] Chi Y, Moon H, Hacigümüs H: iCBS: Incremental Costbased Scheduling under Piecewise Linear SLAs. Proceedings of the VLDB Endowment, 2011, 4(9):563-574
[37] Zaharia M, Borthakur D, S. Sarma J, Elmeleegy K, Shenker S, Stoica I. Job Scheduling for Multi-User MapReduce Clusters. Technical Report University of California at Berkeley, April 2009.
[38] Phan L, Zhang Z, Loo B, Lee I: Real-time MapReduce Scheduling// Technical Report, University of Pennsylvania
[39] Zaharia M, Konwinski A, Joseph A, Katz R, Stoica I: Improving MapReduce Performance in Heterogeneous Environments//OSDI 2008,2008:29-42