Parallel Online Aggregation in Action

Chengjie Qin
UC Merced
5200 N Lake Road
Merced, CA 95343
cqin3@ucmerced.edu

Florin Rusu
UC Merced
5200 N Lake Road
Merced, CA 95343
frusu@ucmerced.edu

ABSTRACT

Online aggregation provides continuous estimates to the final result of a computation during the actual processing. The user can stop the computation as soon as the estimate is accurate enough, typically early in the execution, or can let the processing terminate and obtain the exact result. In this demonstration, we introduce a general framework for parallel online aggregation in which estimation does not incur overhead on top of the actual processing. We define a generic interface to express any estimation model that abstracts completely the execution details. We design multiple sampling-based estimators suited for parallel online aggregation and implement them inside the framework. Demonstration participants are shown how estimates to general SQL aggregation queries over terabytes of TPC-H data are generated during the entire processing. Due to parallel execution, the estimate converges to the correct result in a matter of seconds even for the most difficult queries. The behavior of the estimators is evaluated under different operating regimes of the distributed cluster used in the demonstration.

1. INTRODUCTION

In the era of Big Data, businesses and governments alike are looking at effective methods to extract insights from the massive amounts of data generated on a daily basis. Due to the sheer volume of the data, this is a daunting task even for simple problems like standard SQL aggregates, not to mention more complicated mathematical models. The solutions proposed by the database community to handle the data deluge fall into two categories. Parallel data processing throws more hardware at the problem and makes use of data partitioning to achieve scalability. Approximate query processing gives up the requirement for exact results and employs simpler algorithms that are faster to execute. We believe that, when considered independently, none of these approaches represents an effective method to analyze massive amounts of data.

In this paper, we investigate how to combine parallel processing and approximation to support efficient analysis of the largest datasets. We focus on alternatives to parallelize sampling-based online aggregation—a method at the intersection of exact and approximate computation. In online aggregation [5], an estimate to the final result of the query with progressively narrower confidence bounds is continuously returned to the user. When the confidence bounds become tight enough, typically early in the processing, the user can decide to stop the execution, thus saving valuable resources. If the estimate does not become stable as more data are processed, the user can allow the execution to complete normally and obtain the correct result in the end.

Although introduced in the late nineties, it is hard to find any commercial system that supports online aggregation even today. In our opinion there are two main reasons that hindered adoption given that multiple academic prototypes [2, 9] proved the feasibility of the approach. The first argument underlines the negative effect online aggregation has on normal query execution, with significant increases in query execution time being common [4]. This is regarded as unacceptable in the commercial world which focuses instead on improving performance through parallel data processing. The second argument addresses the lack of a unified approach to express estimation models. This is reflected by major changes to an existent system architecture whenever a novel estimation model is designed [4, 7, 6, 1].

Contributions. We present a generic framework for online aggregation based on a parallel system for the efficient execution of complex analytical queries over terabytes of data [3]. Estimates and corresponding confidence bounds for the query result are computed during the entire query execution without incurring additional overhead—as far as we know, this is the first system achieving this performance. What differentiates the framework we propose from other online aggregation systems is the unified approach to express estimation models. The main idea is to model any estimation using the standard User-Defined Aggregate (UDA) interface [3]. This provides tremendous flexibility in creating various estimation models without modifying any of the system internals.

In this demonstration, we present the characteristics of the parallel online aggregation framework when processing general SPJA (`SELECT-PROJECT-JOIN-AGGREGATE`) queries over terabytes of data in a distributed cluster. We show how accurate estimates are obtained in a few seconds after the execution starts even for the most complicated queries that contain only a handful of tuples in the result. We compare the execution time of the queries with and without estimation to assess the effect estimation has on query execution. To verify the generality of the framework, we implement multiple sampling estimators for two different online aggregation strategies. During the demo, we evaluate the relative performance of the estimators and compare the two strategies—iterative online aggregation and overlapped execution.

Related work. This is not the first online aggregation system presented during a demonstration at a major conference. Nonetheless, as far as we know, it is the first parallel system that provides

SSDBM '13, July 29 - 31 2013, Baltimore, MD, USA

continuous estimates and confidence bounds in a distributed cluster environment. It is also the first that supports multiple estimators for different online aggregation strategies. CONTROL [2] is the original prototype that implements online aggregation ideas for a limited subset of queries. DBO [9] extended these ideas to general SPJA queries. Both these systems work only in centralized settings. BlinkDB [1] uses pre-computed samples at multiple resolution levels to provide one-time approximate answers with confidence bounds. EARL [6] uses bootstrapping over a single precomputed sample. The parallel framework we propose provides continuous estimates computed over samples extracted at runtime.

2. ONLINE AGGREGATION

We consider aggregate computation in a parallel cluster environment consisting of multiple processing nodes. Each processing node has a multi-core processor consisting of one or more CPUs, thus introducing an additional level of parallelism. Data are partitioned into fixed size chunks that are stored across the processing nodes. Parallel aggregation is supported by processing multiple chunks at the same time both across nodes as well as across the cores inside a node.

We focus on the computation of general *associative decomposable aggregate functions*, i.e., functions that are associative and commutative. This class is fairly extensive and includes the majority of standard SQL aggregate functions. Associative decomposable aggregates allow for the maximum degree of parallelism in their evaluation since the computation is independent of the order in which data inside a chunk are processed as well as of the order of the chunks, while partial aggregates computed over different chunks can be combined together straightforwardly. The computation of this type of aggregates can be expressed immediately using the UDA interface [3].

The motivation online aggregation is rooted in is the assumption that the time to compute the aggregate is unacceptably long. Reasons for this can be the massive size of the dataset (i.e., a large number of chunks) or the long time to process a chunk. The idea in online aggregation is to compute only an estimate of the aggregate result based on a sample of the data. In order to provide any useful information though, the estimate is required to be accurate and statistically significant. Different from one-time estimation that might produce very inaccurate estimates for arbitrary queries, online aggregation is an iterative process in which a series of estimators with improving accuracy are generated. This is accomplished by including more data in estimation, i.e., increasing the sample size, from one iteration to another. The end-user can decide to run a subsequent iteration based on the accuracy of the estimator. Although the time to execute the entire process is expected to be much shorter than computing the aggregate over the entire dataset, this is not guaranteed, especially when the number of iterations is large. Other issues with *iterative online aggregation* [10, 6] regard the choice of the sample size at each iteration and reusing the work done from one iteration to the following.

An alternative that avoids these problems altogether is to completely *overlap query processing with estimation* [9, 7]. As more data are processed towards computing the final aggregate, the accuracy of the estimator improves accordingly. For this to be true though, data are required to be processed in a statistically meaningful order, i.e., random order, to allow for the definition and analysis of the estimator. This is typically realized by randomizing data during the loading process. The drawback of the overlapped approach is that the same query is essentially executed twice—once towards the final aggregate and once for computing the estimator. As a result, the total execution time in the overlapped case is expected to be higher when compared to the time it takes to execute each task separately. We show that in a parallel environment with multiple threads available for execution this is not the case if data access is carefully shared across the two tasks.

3. PARALLEL ONLINE AGGREGATION

In this paper, we identify and analyze different approaches to provide online aggregation in a parallel cluster environment where data are partitioned across processing nodes. We first devise a mechanism that allows for the computation of partial aggregates. Then we inspect various parallel sampling strategies to extract samples over which partial aggregates are computed. Each of these strategies leads to the definition of an estimator that we further analyze. We apply the entire process to each of the two online aggregation approaches and characterize their properties.

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. We can fix the subset of the data used in partial aggregation and execute a normal query. Or we can 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. They differ in how the partial aggregates computed for each chunk are combined together. In the centralized approach, 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 only then sent to the coordinator. In the parallel 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, with the final result computed at the root 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. While we discussed these alternatives when applied across processing nodes, notice that they are equally applicable inside a processing node, at the level of a multi-core processor.

Partial aggregation in a parallel setting raises some interesting questions. For iterative online aggregation, the size and location of the data subset used to compute the partial aggregate have to be determined. It is common practice to take the same amount of data from each node in order to achieve load balancing. Or to have each node process a subset proportional to its data as a fraction from the entire dataset. Notice though that it is not necessary to take data from all the nodes. In the extreme case, the subset considered for partial aggregation can be taken from a single node. Once the data subset at each node is determined, parallel aggregation proceeds normally, using either the centralized or parallel strategy. In the case of overlapped execution, a second process that simply aggregates the current results at each node has to be triggered whenever a partial aggregate is computed. The aggregation strategy can be the same or different from the strategy used for computing the final result. Centralized aggregation might be more suitable though due to the reduced interference. The amount of data each node contributes to the result is determined only by the processing speed of

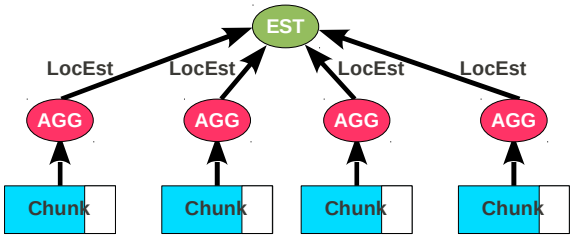

Figure 1: Centralized aggregation with stratified sampling.

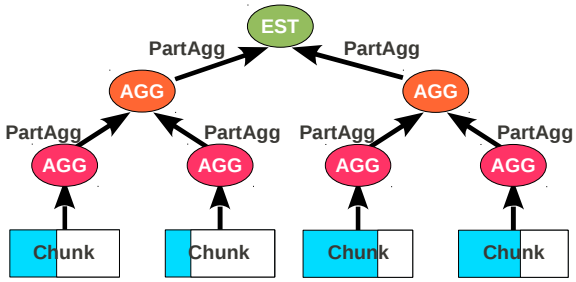

Figure 2: Aggregation tree with global random shuffling.

the node. Since the work done for partial aggregation is also part of computing the final aggregate, it is important to reuse the result so that the overall execution time is not increased unnecessarily.

3.2 Parallel Sampling

In order to provide any information on the final result, partial aggregates have to be statistically significant. It has to be possible to define and analyze estimators for the final result using partial aggregates. Online aggregation imposes an additional requirement. The accuracy of the estimator has to improve when more data are used in the computation of partial aggregates. In the extreme case of using the entire dataset to compute the partial aggregate, the estimator collapses on the final result. The net effect of these requirements is that the data subset on which the partial aggregate is computed cannot be arbitrarily chosen. Since sampling satisfies these requirements, the standard approach in online aggregation is to choose the subset as a random sample from the data.

There are multiple alternatives to obtain a sample from a partitioned dataset—the case in a parallel setting. The straightforward solution is to consider each partition independently and to apply local sampling algorithms inside the partition. This type of sampling is known as *stratified sampling*. While stratified sampling generates a random sample for each partition, it is not guaranteed that when putting all the local samples together the resulting subset is a random sample from the entire data. For this to be the case, it is required that the probability of a tuple to be in the sample is the same across all the partitions. The immediate solution to this problem is to take local samples that are proportional with the partition size. A somehow more complicated solution is to make sure that a tuple can reside at any position in any partition. This can be achieved by randomly shuffling (partition and sort) the data across all the nodes. *Random shuffling* has another important characteristic for online aggregation—it allows for incremental sampling. What this essentially means is that in order to generate a sample of a larger size starting from a given sample is enough to obtain a sample of the remaining size. It is not even required that the two samples are taken from the same partition since random shuffling guarantees that a sample taken from a partition is actually a sample from the entire dataset. Equivalently, to get a sample from a partitioned dataset after random shuffling, it is not necessary to get a sample from each partition.

3.3 Estimation

The partial aggregates generated from the samples are used to compute the estimate and corresponding confidence bounds. The form of the estimator is determined by the underlying sampling process. In the case of stratified sampling, an independent estimator is defined for each partition. These estimators can be computed entirely from the local data. They need to be all combined together in order to define the estimator for the entire dataset. The accuracy of the global estimate is highly sensitive to the accuracy of each local estimate. It is important to notice that if any of the local estimates is missing it is impossible to compute the global estimate. This is completely avoided if random shuffling is used. In this case, a single estimator is defined over all the partitions. Each partition can contribute with different amounts of data, from no tuple at all to the entire partition. The estimator depends only on the number of tuples in the sample and behaves the same independent of where these tuples come from. Someone might think that computing the single estimator cannot be done in parallel anymore. Quite the opposite. As long as the aggregate is associative decomposable, exactly our case, the contribution of each partition can be computed locally. The global estimate can then be obtained by summing up all the local contributions. This is nothing else than parallel aggregation and can be executed using either the centralized or the aggregation tree strategy. To summarize, the difference between stratified sampling and the single estimator defined over randomly shuffled data is quite subtle. The global estimator is computed in the first case by combining independent estimators (Figure 1) while in the second by summing up partial aggregates (Figure 2). Missing estimators have catastrophic effects while missing partial aggregates have not. A complete characterization of the estimators can be found in [8].

3.4 Putting It All Together

A complete online aggregation solution integrates the three components emphasized above—partial aggregation, sampling, and estimation. Independent of the online aggregation strategy, the driver of the entire process is the sampling procedure. Random shuffling at data loading is standard since it reduces the sampling process to a scan over the data. Online aggregation is then executed as follows. Each partition is scanned linearly. In iterative online aggregation (Figure 1), 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. In the overlapped execution strategy (Figure 2), 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, the aggregation continues until all the data are processed and the final result is computed.

4. IMPLEMENTATION

We implement parallel online aggregation in GLADE [3]—a parallel processing system optimized for the computation of GLAs, i.e., user-defined associative decomposable aggregates. While sam-

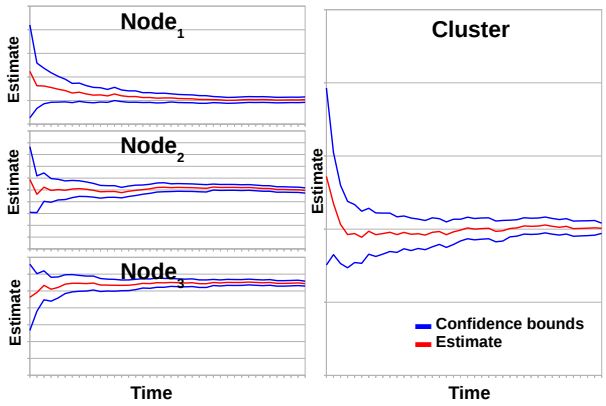

Figure 3: Demonstration GUI. Local estimates and confidence bounds corresponding to each node are displayed in the left panel. The overall estimate is continuously updated in the right panel.

pling can be effectively implemented through linear scans over randomly shuffled data, overlapping GLA execution and partial aggregate computation is a considerably more challenging task. The obvious choice is to process exclusively the partial aggregation in order to generate the estimate and the bounds as soon as possible. This results in dropped chunks and thus incurs delays in the final GLA processing. Nonetheless, it is the only available choice in all the existent online aggregation systems we are aware of. The solution we adopt is to overlap online estimation and GLA processing. Abstractly, this corresponds to executing two simultaneous GLA computations. Given that GLADE is a multi-query processing system optimized for sharing data across queries, optimal performance is expected. To get this performance though, careful consideration of several aspects is required. Rather than treating actual computation and estimation as two separate GLAs, we group everything into a single GLA. This simplifies the computation as follows. Partial aggregation triggers the merging of all existent GLAs, inside the execution engine and under update with chunks. The resulting GLA is the partial aggregate. Unlike the final result which is extracted and passed for further merging across the nodes, a copy of the partial aggregate needs to be kept and used for computing the final result and future partial aggregates. The newly arriving chunks are processed as before. The result is a completely new list of GLAs. The local partial aggregate obtained through merging is added to this new list.

In GLADE, the partial aggregate, or GLA, is defined by the user. It contains all the necessary data to allow estimation. The GLA computation is also entirely specified by the user. The system acts like a framework that calls the user-defined functions at the right place in the execution pipeline. As a result, the user gets direct access to the partial aggregate and can execute any computation required for estimation, during the aggregation process, or at the end. This model supports the computation of any type of aggregate and any estimation algorithm. Online aggregation based on sampling is only one such estimation technique.

5. THE DEMONSTRATION

Conference participants attending the demonstration will experience parallel online aggregation in a distributed cluster environment consisting of nine nodes and running GLADE. An instance of the TPC-H dataset scale 1,000 (1TB) resides on each processing node. Several TPC-H queries will be ready-to-go—from simple, single-table aggregates, to complex aggregate functions computed over multi-table joins (see [8] for examples). The attendee can choose or modify one of the queries and execute it on the cluster. Our system spawns a graphical user interface (Figure 3) that shows the estimates and confidence bounds for each processing node as well as the entire cluster.

To provide a complete picture on parallel online aggregation, the demonstration is organized around a series of scenarios:

Estimator convergence. The estimate and corresponding confidence bounds are displayed as a function of time (or, equivalently, the amount of data processed). The participants will witness how the width of the confidence bounds is continuously shrinking as more data are processed, collapsing on the true query result at the end of execution. The processing can be stopped whenever the participant is content with the accuracy of the estimate. We also analyze the convergence rate of the estimators discussed in Section 3.3 for different query configuration parameters.

Estimator reliability. We study how stratified sampling and the single estimator over randomly shuffled data behave when there is significant variability in workload across the processing nodes. We vary the number of nodes with increased load and let the participants observe the behavior of the local estimators as well as the effect on the overall estimator and its corresponding confidence bounds. As an extreme case, we halt a node altogether.

Execution with and without estimation. To study the overhead introduced by the estimation process, we execute two instances of the same query in parallel—with and without estimation. This is possible due to the multi-query processing capabilities provided by GLADE. The participants will confirm that the overhead introduced by estimation is negligible, if at all visible.

Iterative online aggregation. We illustrate the differences between the two strategies for online aggregation—iterative and overlapped execution. Instead of just running the query and specifying the confidence level for the bounds (overlapped execution), the demo attendee is required to provide an initial sample size and a confidence level (iterative). Choosing the optimal sample size is more of an art rather than something that can be done in a principled manner. The result is a single value with a confidence interval around it. The attendee can decide to execute another iteration if unhappy with the accuracy of the result. Alternatively, the desired accuracy can be specified from the start and the system executes the required number of iterations to achieve it.

Acknowledgment. This work was supported in part by a gift from LogicBlox.

6. REFERENCES

[1] S. Agarwal and al. Blink and It's Done: Interactive Queries on Very Large Data. *PVLDB*, 5(12), 2012.
[2] R. Avnur and al. CONTROL: Continuous Output and Navigation Technology with Refinement On-Line. In *SIGMOD 1998*.
[3] Y. Cheng and al. GLADE: Big Data Analytics Made Easy. In *SIGMOD 2012*.
[4] A. Dobra and al. Turbo-Charging Estimate Convergence in DBO. In *VLDB 2009*.
[5] J. Hellerstein and al. Online Aggregation. In *SIGMOD 1997*.
[6] N. Laptev and al. Early Accurate Results for Advanced Analytics on MapReduce. *PVLDB*, 5(10), 2012.
[7] N. Pansare and al. Online Aggregation for Large MapReduce Jobs. In *VLDB 2011*.
[8] C. Qin and F. Rusu. PF-OLA: A High-Performance Framework for Parallel On-Line Aggregation. *CoRR*, abs/1206.0051, 2012.
[9] F. Rusu and al. The DBO Database System. In *SIGMOD 2008*.
[10] S. Wu and al. Distributed Online Aggregation. In *VLDB 2009*.