Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Progressive online aggregation in a distributed stream system

Dingyu Yang[a], Jian Cao[a,*], Sai Wu[b], Jie Wang[c]

[a] Department of Computer Science and Engineering, Shanghai Jiao Tong University, 800 Dong Chuan Road, Minhang, Shanghai 200240, China
[b] College of Computer Science, Zhejiang University, Hangzhou 310027, P.R. China
[c] Department of Civil and Environmental Engineering, Stanford University, Stanford, CA 94305, USA

ARTICLE INFO

Article history:
Received 22 March 2014
Revised 23 August 2014
Accepted 9 November 2014
Available online 29 November 2014

Keywords:
Online aggregation
Stream processing
Actor model

ABSTRACT

Interactive query processing aims at generating approximate results with minimum response time. However, it is quite difficult for a batch-oriented processing system to progressively provide cumulatively accurate results in the context of a distributed environment. MapReduce Online extends the MapReduce framework to support online aggregation, but it is hindered by its processing speed in keeping up with ongoing real-time data events. We deploy the online aggregation algorithm over S4, a scalable stream processing system that is inspired by the combined functionalities of MapReduce and Actor model. Our system applies an asynchronous message communication mechanism from actor model to support online aggregation. It can process large scale data stream with high concurrency in a short response time. In this system, we adopt a distributed weighted random sampling algorithm to solve biased distribution between different streams. Furthermore, a multi-level query processing topology is developed to reduce overlapped processing for multiple queries. Our system can provide continuous window aggregation with a confidence interval and error bound. We have implemented our system and conducted plentiful experiments over the TPC-H benchmark. A large number of experiments are carried out to demonstrate that by using our system, high-quality query results can be generated within a short response time and that the approach outperforms MapReduce Online on data streams.

1. Introduction

Online aggregation is a valuable research topic proposed by Hellerstein et al. (1997), aiming at faster response to Online Analysis Processing (OLAP) for business analysis and decision making. Reduced accuracy is the price of shorter response times, for the simple reason that the whole dataset would not be processed in that short time. Instead of providing final results after processing all the data, online aggregation progressively generates approximate results to users and the corresponding statistical analysis (e.g., confidence intervals or error bounds). The results are refined by statistical approaches with more and more data, until the confidence interval and error bounds reach a satisfactory level.

One important issue that has been addressed is how to get early aggregation results in a distributed system. Some researchers have made contributions to this research issue. MapReduce Online (Condie et al., 2010) exploits the powerful data processing capability of the Hadoop MapReduce framework so that it can support online aggregation. Since MapReduce architecture was originally highly optimized for batch processing, Hadoop, an open-source implementation of MapReduce, has achieved a general consensus, which undoubtedly makes it an optimal choice for applications like online aggregation. However, the work of Condie et al. (2010) fails to provide us with a statistical analysis for online aggregation. The work Pansare et al. (2011) solves the "inspection paradox" problem in MapReduce jobs and applies a Bayesian framework for producing estimates and confidence bounds. But their model is very complex and needs $3m+2$ dimensions for one query (where m is the machine number). For such high dimensions, it is difficult to extend to high speed streams.

Another challenge is how to randomly generate samples from distributed streams. As data volume increases, it is no longer practical to collect all the data together and then perform a sampling algorithm since it needs large memories to cache the stream data to generate samples. For example, some sensor applications continuously send data to a processing system with high speed and streams from different sites might have multiple data distributions so that the samples cannot be generated simply. These problems have led to investigations into how to randomly generate samples on a continuous and distributed streaming data.

In this paper, we introduce our approach of deploying a distributed online aggregation algorithm over an open-source system S4

* Corresponding author. Tel.: +86 21 34204426; fax:+86 21 34204728.
E-mail addresses: yangdingyu8686@sjtu.edu.cn (D. Yang), cao-jian@sjtu.edu.cn (J. Cao), wusai@zju.edu.cn (S. Wu), jiewang@stanford.edu (J. Wang).

http://dx.doi.org/10.1016/j.jss.2014.11.027
0164-1212/© 2014 Elsevier Inc. All rights reserved.

(S4, 2010), a scalable stream processing system. In order to support online aggregation, we propose a distributed weighted random sampling algorithm, which does not only sample from distributed streams, but also generates samples with different weights. Our model parses multiple queries into a complex topology to guide the execution, which is called *Multi-level Query Processing*. Tuples are encapsulated as events, and query processing is performed by consecutive processing elements, each responsible for a stage in the complex task. Such a separation-of-duty processing style makes the expensive aggregation queries can be processed in a more reliable and more efficient way. For concurrent queries, we propose a scheme called *Synthetic Processing Topology* to merge streams that may incur repeated computing on overlapped data. We can continuously provide aggregation estimation with a confidence interval and error bound. Our implementation also makes use of the check-pointing technique of Apache S4, so that our aggregation framework can achieve high availability.

In summary, the contributions of this paper are listed as follows:

- We present a distributed weighted random sampling algorithm, which can quickly generate samples from a distributed stream. We extend Weighted Reservoir Sampling in distributed stream sites and apply a *Coordinator* operator to merge the samples.
- We propose actor model to solve complex logical problems and asynchronously process distributed stream data. An incremental processing mechanism is designed to support one-pass process. As data are read, the stream system fast processes it in memory and updates the results to achieve high performance.
- We develop a dynamic processing topology to direct the stream processing when facing multiple new queries. The topology can reduce the number of overlapping operations. The problem can be decomposed into a set of independent tasks which communicate with each other by sending and receiving messages.
- We implement the online aggregation algorithm and deploy it on the distributed stream system *S*4. Extensive experiments have been conducted to show the efficiency and robustness. It can produce accurate results very quickly through asynchronous message exchange. In general, our system performs 10–30 times faster than MapReduce Online on stream data.

The rest of the paper is organized as follows. In Section 2, we present an example to illustrate the problem and how we deal with it differently from former work. Section 3 provides the detailed description of our system and methodology. Section 4 presents a statistical analysis of the confidence interval and error bound. Extensive experiment results are reported in Section 5. Section 6 offers a brief review of related studies. Finally, we conclude this paper in Section 7.

2. Background

2.1. Actor model

The actor model (Agha, 1985) is a mathematical theory of computation, that treats “Actors” as the universal primitives of concurrent computation. It has been used both as a framework for a theoretical understanding of computation, and as the theoretical basis for several practical implementations of concurrent systems.

An actor is a computational entity that, in response to a message it receives, can concurrently (Agha, 1985):

- send a finite number of messages to other actors;
- create a finite number of new actors;
- designate the behavior to be used for the next message it receives.

The communication between actors can be considered to be either:

- Synchronous, where the sender and the receiver of a communication are both ready to communicate;
- Asynchronous, where the receiver does not have to be ready to accept a communication when the sender sends it.

The processing of communications depends on the behavior of individual actors and buffered communication can improve the efficiency in execution by pipelining multiple actors. A potentially variable topology is also created to define the relationships among actors.

Table 1
SQL example.

```
SELECT Average(Value) FROM (
SELECT Top 50 * FROM (
SELECT Orderkey, Sum(Quantity) as Value
FROM Lineitem GROUP BY Orderkey)
ORDER BY Value)
```

2.2. Online aggregation

In tradition, aggregation is done in a batch mode and users need to keep waiting without any feedback until the aggregation process completes and returns the results. Online aggregation (Hellerstein et al., 1997) is proposed to extend batched aggregation by improving the interactive behavior of database systems in order to process expensive analytical queries. It can progressively refine running estimates of the final aggregation values and statistical analysis is continuously displayed to users. Users can observe the progress of the aggregation and control the execution process on the fly.

We illustrate online aggregation by taking the SQL statement in Table 1 as an example. The final aggregation result is 3250. While 30% of the data have been processed, users can get the early result 3200 with estimation. The confidence interval (3200 ± 10) shows the accuracy of the early result with 95% probability. The aggregation can be stopped during the processing if the result is good enough.

2.3. An example of applying actor model

In this paper, we adopt a pipeline-style approach by using an actor model to execute complex queries over a large volume of data.It decomposes a complex job into independent subtasks, which are processed by actors concurrently. The communication link between two tasks is used to send and receive messages. The job is organized in a directed acyclic topology, which will be automatically deployed to physical resources in practical systems. The data are transferred through the communication links in the topology and processed in memory.

As an example, suppose we have a complex query with nested aggregate and sort operations. The purpose is to obtain the average summary *quantity* of the top 50 ordered by *orderkey* in table *lineitem* from the TPC-H benchmark. The SQL statement is shown in Table 1.

As it is shown in Fig. 1, the input stream is continuously sent to different actors and each actor can support incremental processing. *Partition Actor* splits the data based on the value of *orderkey*. In our example, data events with the same *orderkey* are sent to the same actor through one specific communication link. When data events with a specific *orderkey* arrive in a *Group Actor*, the *Group Actor* incrementally aggregates the results of the *quantity*. After that, the new results will be sent to the next stage *Order Actor*. *Order Actor* concurrently sorts the data results and *Filter Actor* filters out the top 50 orders in local partition. The top 50 orders in each partition are then merged together in *Merge Actor*. The output of *Merge Actor* is the global top 50 orders and is sent to *Average Actor*. Finally, the average quantity of the top 50 orders will be calculated. During the whole process, actors process data in an asynchronous and parallel way.