DEBS Grand Challenge: Deterministic Real-Time Analytics of Geospatial Data Streams through ScaleGate Objects

Vincenzo Gulisano, Yiannis Nikolakopoulos, Ivan Walulya,
Marina Papatriantafilou and Philippas Tsigas
Chalmers University of Technology, Gothenburg, Sweden
{vinmas, ioaniko, ivanw, ptrianta, tsigas}@chalmers.se

ABSTRACT

This paper presents our solution to the DEBS 2015 Grand Challenge. The analysis of the Grand Challenge is partitioned among an arbitrary number of processing units by leveraging *ScaleGate*, a recently proposed abstract data type with its concurrent implementation which articulates data access in parallel data streaming. *ScaleGate* aims not only at supporting high throughput and low latency parallel streaming analysis, but also at guaranteeing deterministic processing, which is one of the biggest challenges in parallelizing computation while maintaining consistency.

Categories and Subject Descriptors

E.1 [**Data Structures**]: Lists, stacks, and queues; H.2.4 [**Database Management**]: Systems—*Concurrency*

1. INTRODUCTION

The motivation behind the DEBS 2015 Grand Challenge [6], shared by distributed and parallel streaming applications, is the need for high processing throughput and low processing latency. The continuously wider adoption of many-core and heterogeneous systems (spanning from high-end servers to embedded system-on-chip devices) is an enabler for such high processing throughput and low processing latency to happen. However, the possibilities enabled by such systems come at a price. More concretely, there are challenges that come along with identifying which parts of the computation can be parallelized efficiently, as well as with ensuring that the analysis' results are produced deterministically, independently of how a given framework is deployed (e.g., which and how many heterogeneous devices run the analysis). The need for deterministic analysis is further exacerbated when data sources are distributed and generate data with different granularities and different speeds [3]. As was detailed in [2, 1, 4], the underlying shared objects which the different processing units utilize (e.g. data streaming operators), constitute a key enabler in achieving high throughput, low latency and deterministic execution in a distributed and parallel analysis framework. Paraphrasing the definitions of deterministic processing provided in [3, 2]:

DEBS'15, June 29 - July 3, 2015, OSLO, Norway.

ACM 978-1-4503-3286-6/15/06...$15.00.
http://dx.doi.org/10.1145/2675743.2776758.

Definition 1. *The property of deterministic data streaming processing guarantees that, by consuming the tuples delivered by an arbitrary set of input streams, the parallel analysis run on such tuples produces exactly the same output that would be produced by its centralized counterpart.*

As discussed in [2], a condition to enforce deterministic parallel analysis is to process the tuples delivered by distinct streams in timestamp order. In [1], we introduced the concept of *ready* tuples:

Definition 2. *Let t_i^j be the i-th tuple in timestamp-sorted stream j. Tuple t_i^j is ready to be processed if $t_i^j.ts \leq merge_{ts}$, where $merge_{ts}$ is the earliest among the latest timestamps from each timestamp-sorted stream j, i.e. $merge_{ts} = min_j\{max_i(t_i^j.ts)\}$.*

In order to allow for parallel analysis of timestamp-sorted streams of *ready* tuples (while preventing the merging of tuples from constituting a bottleneck of the analysis) we introduced in [4] the *ScaleGate* data structure,[1] which (a) guarantees properties essential for concurrently merging streams, (b) integrates the necessary synchronization to allow multiple threads to consume *ready* tuples concurrently and (c) allows for an arbitrary number of source threads to deliver the tuples forwarded by an arbitrary number of timestamp-sorted input streams, also allowing for arbitrary number of reader threads to consume the same sequence of timestamp-sorted *ready* tuples. The interface of *ScaleGate* provides the following methods:

- addTuple(timestamp, tuple, sourceID): which allows a tuple carrying the given timestamp from the source thread sourceID to be merged by *ScaleGate* in the resulting timestamp-sorted stream of *ready* tuples.
- getNextReadyTuple(readerID): which provides to the calling reader thread readerID the next earliest *ready* tuple that has not been yet consumed by the former.

2. PROPOSED SOLUTION

Challenge Overview The DEBS 2015 Grand Challenge focuses on the analysis of a high volume stream of travel reports generated by taxis in New York City. Such reports are analyzed to find the top-10 most *Frequent Routes* and *Profitable Areas* for taxi drivers. Routes and areas are identified based on grid cells of partitions of the New York City metropolitan area. Travel reports (also referred to as tuples in the remainder) are composed by attributes $\langle$ taxi, ts, Q1_pickupCell, Q1_dropoffCell, Q2_pickupCell, Q2_dropoffCell, amount $\rangle$. The pick up and drop off locations of each travel report are expressed by means of cell grids, composed by either 300X300 Q1 cells of 500^2 m^2, or 600X600 Q2 cells of 250^2 m^2.

[1]We refer the reader to [4] for a detailed discussion of *ScaleGate* design and implementation choices.

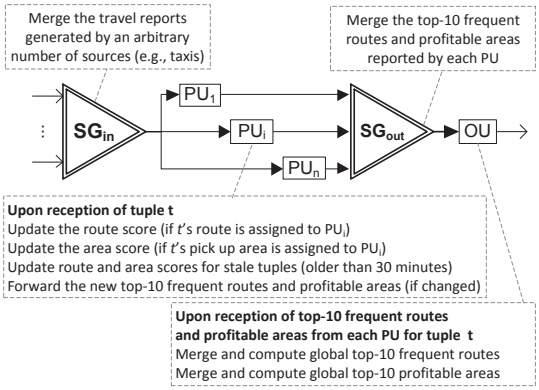

Figure 1: Overview of the system architecture

Query *Frequent Routes* aims at computing the top-10 most frequent routes during the last 30 minutes (each route defined as a pair $\langle$Q1_pickupCell, Q1_dropoffCell$\rangle$). The score of each route is defined as the number of tuples associated with the specific route within the given time window. Query *Profitable Areas* aims at computing the top-10 most profitable areas (each defined as a Q2 cell). The score of each area is defined as its *profit* divided by its number of *empty taxis*. Given a Q2 cell C, the *profit* is the median value of the amount attribute for tuples observed during the last 15 minutes for which Q2_pickupCell = C. *Empty taxis* is the number of tuples observed during the last 30 minutes for which Q2_dropoffCell = C, given that each of the taxis has not appeared in another tuple with a different drop off location later during the same period.

Solution Overview By leveraging the *ScaleGate* data structure, we distribute and parallelize the analysis of queries *Frequent Routes* and *Profitable Areas* among n Processing Units (PUs), each responsible for approximately 1/n of the overall analysis. The idea behind the workload-partitioning scheme is based on the observation that the top-k values for a set of elements can be computed by first computing the top-k values of disjoint partitions of such a set and subsequently choosing the global top-k by merging the results from the disjoint partitions. In this regard, the parallelism of the analysis can be enhanced by increasing the number of top frequent routes and profitable areas being observed (and thus allowing for such analysis to be carried out by an increasing number of PUs) and subsequently computing the global top-10 frequent routes and profitable areas. It should further be noted that, in order to increase the pipeline parallelism, the top-10 most frequent routes and profitable areas observed for disjoint routes and areas partitions should be merged into the global top-10 only when a change is observed in at least one of the partitions.

In the proposed solution, each of the PUs computes the top-10 most frequent routes for its share of routes (1/n of the overall routes), which are merged in sorted order to produce the overall top 10 most frequent routes for the period under consideration. Similarly, each of the PUs computes the top-10 most profitable areas for its share of areas (1/n of the overall areas), while the overall top-10 most profitable areas are computed by merging them as in the previous query case. Figure 1 overviews the architecture of the proposed solution, distributing the analysis among n PUs.

Enforcing Deterministic Processing To ensure the correctness of the results when multiple PUs are in charge of computing the top-10 routes and areas, each PU should process its incoming tuples in a deterministic fashion. To achieve this, we enforce ordered processing of input tuples for an arbitrary number of PUs by feeding them with the timestamp-sorted stream of *ready* tuples generated by *ScaleGate* (referred to as SG_{in} in the figure). Furthermore, in order for results to be correct, we split the analysis ensuring that exactly one PU is expected to maintain the frequency of a given route or the profitability of a given area. Finally, in order to produce the global top-10 most frequent routes and profitable areas deterministically, a second *ScaleGate* instance (referred to as SG_{out} in the figure) is leveraged to gather and merge all the top-10 frequent routes and profitable areas produced by an arbitrary number of PUs into a timestamp-sorted stream of *ready* top-10 values. Such stream is fed to a dedicated Output Unit (OU), which merges the top-10 values into the results of queries *Frequent Routes* and *Profitable Areas*.

Evaluation Results The DEBS 2015 Grand Challenge requests the proposed solutions to be delivered in a virtual machine with 4 cores and 8GB of memory. Based on this request, we opted for a multi-threaded process implementation of our solution. This implementation choice gave us valuable insights not only on its performance, but also on the best practices to run it both in shared-memory and shared-nothing environments. We refer the reader to [5] for a detailed description of our solution, as well as an evaluation discussing the applicability of the proposed parallelization method, its performance in terms of execution time and processing latency, the scalability of the *ScaleGate* data type and the tuples' processing cost for the different threads running within the process.

In our evaluation we show that, even with a machine having access to a reduced number of cores, the leveraging of the *ScaleGate* data type allows the data produced by thousands of taxis over a period of one year can be analyzed in approximately 11 minutes, with a processing throughput of approximately 260,000 t/s and processing latencies of 0.09 seconds. These results provide evidence of the benefits enabled by streaming-aware concurrent data structures such as *ScaleGate* in the context of parallel data streaming analysis.

Acknowledgments. The research leading to these results has been partially supported by Chalmers Transportation Area of Advance, by the collaboration framework of Chalmers Energy Area of Advance project "SN09: EXAMINE", by the European Union Seventh Framework Programme (FP7/-2007-2013) through the EXCESS Project (www.excess-project.eu) under grant agreement 611183 and by the Swedish Research Council (Vetenskapsrådet) project "Fine-grain synchronization in parallel programming", contract nr. 2010-4801.

References

[1] D. Cederman, V. Gulisano, Y. Nikolakopoulos, M. Papatriantafilou, and P. Tsigas. Concurrent data structures for efficient streaming aggregation. *Report, Chalmers University of Technology*, 2013.

[2] D. Cederman, V. Gulisano, Y. Nikolakopoulos, M. Papatriantafilou, and P. Tsigas. Brief announcement: Concurrent data structures for efficient streaming aggregation. In *Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '14, pages 76–78, 2014.

[3] V. Gulisano, R. Jimenez-Peris, M. Patino-Martinez, C. Soriente, and P. Valduriez. Streamcloud: An elastic and scalable data streaming system. *IEEE Transactions on Parallel and Distributed Systems*, 99, 2012.

[4] V. Gulisano, Y. Nikolakopoulos, M. Papatriantafilou, and P. Tsigas. ScaleJoin: a deterministic, disjoint-parallel and skew-resilient stream join enabled by concurrent data structures. *Report, Chalmers University of Technology*, 2014.

[5] V. Gulisano, Y. Nikolakopoulos, I. Walulya, M. Papatriantafilou, and P. Tsigas. Deterministic real-time analytics of geospatial data streams through ScaleGate objects. *Report, Chalmers University of Technology*, 2015.

[6] Z. Jerzak and H. Ziekow. The DEBS 2015 Grand Challenge. In *DEBS 2015: the 9th ACM International Conference on Distributed Event-Based Systems*, June 2015.