# Integrating Massive Data Streams

George Siachamis
supervised by Geert-Jan Houben, Arie van Deursen and Asterios Katsifodimos
Delft University of Technology
g.siachamis@tudelft.nl

## ABSTRACT

Data Integration has been a long-standing and challenging problem for enterprises and researchers. Data residing in multiple heterogeneous sources must be integrated and prepared such that the valuable information that it carries, can be extracted and analysed. However, the volume and the velocity of the produced data in addition to the modern business needs for real-time results have pushed data analytics, and therefore data integration, towards data streams. While data integration is a hard problem in and of itself, integrating data streams becomes even more challenging. Streams are characterized by their high velocity, infinite nature and predisposition to concept drift.

The goal of this doctoral work is to design and provide scalable methods to support data integration tasks on massive data streams, i.e., support *streaming data integration*. The aim of this work is threefold. First, we aim at developing and proposing streaming methods to compute temporal stream data-profiles and summaries that can describe the dynamic state of a stream in the course of time. Second, we aim at developing methods and metrics of stream similarity. Those methods and metrics can serve as means to detect similar or complementary streams in a streaming data lake. Finally, we aim at optimizing distributed streaming similarity joins - a very important operation that precedes entity linking and resolution. This paper discusses exciting challenges and open problems in the field, and a research plan on tackling them.

## 1 INTRODUCTION

Modern enterprises are gathering huge volumes of data either to perform business analytics or manage efficiently their assets. This data resides in disparate sources and although it can convey the same or related information, it can differ considerably in structure and representation based on the different conventions of the managing teams. This untamed heterogeneity leads to the so called data integration problem. Traditionally, data integration has been a manual process upon targeted static sources. The last decades, plenty of research time has been invested to automate and improve the accuracy of data integration tasks [11, 24, 25]. At the same time, a fair amount of work [14, 15, 19] has been done towards improving the efficiency of data integration tasks in static and dynamic databases. However, due to the ever-growing volume and velocity of produced data as well as the demand for data-driven real-time applications, streaming data analytics have emerged. To ensure the quality of these analytics, streaming data must be prepared and integrated in a real-time fashion.

Proceedings of the VLDB 2021 PhD Workshop, August 16th, 2021. Copenhagen, Denmark. 

Recognizing the growing need for efficient streaming data integration, in this project we aim at providing scalable streaming methods to facilitate integrating massive data streams. Essentially, a streaming data integration pipeline consists of multiple basic data integration tasks adapted and optimized to handle data streams. For this doctoral work, our efforts focus on three main tasks: *stream profiling*, *stream discovery* and *similarity joins*. We recognize that these tasks play a major role in data integration pipelines and they are integral for the efficiency of those pipelines. On the one hand, with *stream profiling* and *stream discovery*, temporal profiles can be computed and used for discovering possibly related streams to optimize downstream data integration tasks. On the other hand, *similarity joins* are an integral data integration task with applications to data cleaning and entity resolution but not limited to those. Summarizing, we aim at adapting and optimizing these basic tasks for data streams in order to improve the efficiency of a streaming data integration pipeline. Our ultimate goal is to enable the real-time results needed to ensure the quality of real-time data analytics. We plan on evaluating our work in real-word use cases provided by our industrial partner, ING.

## 2 MOTIVATION & CHALLENGES

In many modern enterprises, different teams publish their streaming datasets to an internal streaming data lake where other teams can access them. However, it is very rare for the published streams to come with valuable time-related metadata. This results to hundreds or even thousands of data streams published in an internal repository but never harnessed because of lacking documentation and valuable metadata. Unfortunately at the moment of writing, the only way of organising and harnessing all these streams is through long valuable labour hours of manual exploration and integration. The existing tools cannot deal with the massive rate of thousands of incoming data per second that usually characterizes modern data streams. Another major challenge is also the fact that streams are possibly unbounded datasets meaning that their volume is growing infinitely. This also means that are not at our disposal at their full prior to their processing, i.e., the whole dataset is not available when it starts to be processed. All of these in combination with the fact that data streams might suffer from concept drift every other day renders the task of handling these data streams too complex for existing tools.

An inspiring example, that we also recognized through our industrial collaboration with ING, is the monitoring of crucial infrastructure. Modern enterprises consist of multiple teams which monitor their own assets and provide alerts for the incident teams. Although these assets might differ considerably in their representation in the data provided by each team, they are often closely related. For any enterprise, it is crucial that an occurring incident

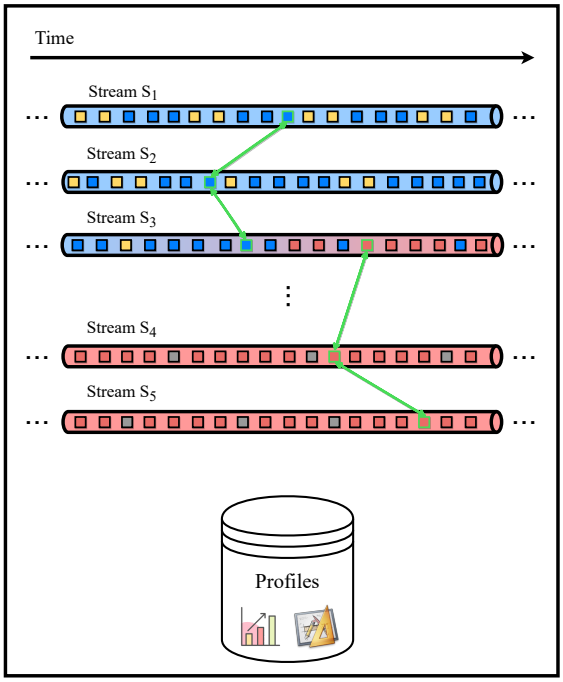


**Figure 1: A streaming data lake applying our streaming methods. For each stream a temporal profile is computed. These profiles are used to find similar streams (streams with the same coloring). The green arrows indicate the matching records that the similarity join between the related streams outputs.**

will be resolved as fast as possible. It is also crucial that in the process of resolving any occurring issue, the involved engineers will not miss any relevant information. In other words, the monitoring streams must be integrated in real-time and in an exact manner.

In this project, our goal is to face these challenges and provide methods that will bring us one step closer to a tool that help a company tame the streams and take advantage of their rich information. To ensure the usefulness of our work, we also plan on evaluating it on a relevant to the given example monitoring case in the industrial environment of our partner, ING.

## 3 SCALABLE METHODS FOR STREAMING DATA INTEGRATION

In this section, we discuss the main three work packages of this doctoral work, our three scalable streaming methods. First we give an overview of the envisioned methods. Then we present each task in details and we discuss the related work.

### 3.1 Overview of Streaming Methods

Our envisioned methods can be used either individually or in various combinations depending on the use case at hand. An overview of these methods is the following.

**Profiling Streams.** Our proposed stream profiling method is designed to work on top of a streaming data lake and provide approximate temporal profiles that will describe statistical and semantic properties of the current state of the stream. These profiles are constantly updated in an incremental manner to reflect the changes in the content of the streams.

**Finding Related Streams.** Based on previously created profiles, our stream discovery method computes the similarity between streams in our streaming data lake indicating streams that have a high chance of being related. These indications of relatedness are adaptively updated as the streams change over time. The resulting related streams are either presented to the end user or used to guide the downstream tasks of a workflow.

**Joining Similar Records.** Our streaming similarity join method takes as input two or more streams and outputs all possible record matches. In order to identify and join the similar records, our method computes a similarity score through a similarity function. In this project, the primary focus is to optimize this similarity comparison to significantly improve the efficiency of the streaming integration tasks. The results of our similarity join task can be used as part of one of the entity resolution pipelines described in [17].

In the rest of the section, we go into details about the individual tasks that the methods perform and we discuss the related work.

### 3.2 Profiling Streams

An important step before integrating streams is to compute data profiles for all the streams in a streaming data lake. Depending on the nature of the downstream tasks, different types of data profiles can be of interest. In this project we mainly focus on two types: basic statistics-based profiles and summaries/sketches.

Basic statistics-based profiles are easier to compute but also less informative for tasks like a similarity join. These profiles typically contain information like the cardinalities, the value distributions or the data types of columns. This information can be used to reduce the combinations of items to be checked from downstream tasks, e.g. combinations of attributes or streams to be checked for similarity. This can be done either in a stream level by identifying streams with common statistical properties or in column level by narrowing down the column combinations to be examined. For example, the value distributions of columns can help identify candidate pairs of columns on which a downstream similarity join will be performed.

On the other hand, sketches and summaries [7, 13] are harder to create but they can give a good estimate of the contents of a stream. Sketches or summaries can be very useful in various problems like approximate and streaming query processing or dataset discovery. In [16] summaries are computed for approximate query answering based on a probabilistic technique that makes use of Maximum Entropy. In [13], sketches are created for dataset discovery by leveraging bloom filters and a skip list. However, none of the above techniques incorporates the time factor in their sketches. In addition, [16] is not targeting data streams and an adaptation is far from trivial due to the complexity of the technique.

According to [1] there is still much more ground to cover to perform incremental, online and temporal profiling. For this project, it is important that our profiles are computed online and are constantly updated to capture the temporal properties of our streams. This is not an easy task when data streams are possibly endless, fast and massive and their statistical values can change often. Time

needs to be incorporated on the profiles and any process must be incremental to ensure efficiency.

### 3.3 Finding Related Streams

After computing the desired profile for each stream, we must identify which of our streams are related based on the profiled information. Depending on the type of the collected profiles, different strategies can be employed.

A simple example, in the case of sketches, is the simple procedure described in [13]. Here, the authors suggest to use the computed sketches to acquire a simple estimation of overlapping values between two sources by computing the overlap between the way smaller corresponding sketches. This way an estimation of whether two sources must be integrated or not can be made, as well as an estimation of the cost of this integration. However, the described scenario is pretty simple, it uses only sketches and does not incorporate time-related capabilities like temporal queries.

A better example to showcase the use of profiles though, is the data discovery system Aurum [5]. In Aurum, the authors propose creating a knowledge graph based on previously computed profiles. The knowledge graph is afterwards queried for dataset discovery purposes. The computed profiles consists of multiple statistical properties, discovered dependencies and sketches. These profiles are used from Aurum to prune the search space and reduce the similarity comparisons needed to build the knowledge graph. The authors have opted for a scalable parallel solution that reads the input once, and they also provide a mechanism for keeping up-to-date both the profiles and the knowledge graph. However, Aurum is not designed for streams and does not provide temporal features both in its profiles and its knowledge base. Thus, temporal queries are not supported. In a streaming environment and especially for a crucial task like monitoring infrastructure, such a capability is essential.

### 3.4 Joining Similar Streaming Records

Similarity join [3, 6, 22] is the problem of identifying all pairs of similar records that reside in two or more datasets. A pair of records is considered similar if the similarity score given by a similarity function is above a given threshold. In a stream processing model, the similarity join operation between two given streams is expected to join similar records from the incoming streams based on the values of one or more target attributes. In a streaming environment we can distinguish two types of similarity joins: the full-history and the windowed joins.

Similarity joins are difficult and time-consuming operations. The brute force approach has to compare all the data of the first dataset against the data on the second, leading to a quadratic time complexity, $O(n^2)$. When we take into account that a data stream is a possibly unbounded dataset, it is clear that a brute force solution is infeasible in a streaming environment. Thus, it is essential that the performed comparisons between records are reduced by avoiding unnecessary computations. Additionally, due to the dynamic nature of data streams any occurring concept drift might result on obsolete partitions and load skew. To ensure the high efficiency of the task, we must adaptively partition the data online.

In what follows, we will discuss the main work in the related fields.

**Similarity Joins in Map Reduce.** Similarity joins have been studied a lot for MapReduce environments. In general, MapReduce methods require their inputs at their full before processing, and most of them leverage statistics and properties of the datasets to optimize the task and reduce the transmission and computation costs. They provide a one-off partitioning scheme which cannot be updated adaptively on runtime. Thus, they are not trivially applicable on a streaming environment but they are a great inspiration towards a distributed streaming solution.

There are two main approaches which MapReduce methods usually follow: Filter & Verification and General Metric Space.The Filter & Verification methods [20],[10] rely on prefices and signatures which they leverage to scale out the similarity computations and filter unnecessary comparisons. On the other hand, General Metric space methods [21],[8] divide the metric space in partitions to which similar objects are grouped. However, they require that the similarity function is a metric, or at least a semi-metric. More specifically, [8] and [21] select random centroids, create an inner and an outer partition for each by using centroid proximity and filters, and compute a)the similarity of all pairs of items in an inner partition and b)the similarity of each item of an outer partition with all the items within the corresponding inner partition.

**Similarity Joins for Data Streams.** On the other hand, research on similarity joins in a streaming environment is very limited. [9] introduces the problem of streaming similarity self-join. It proposes a similarity measure which filters out old items and a streaming framework that leverage an optimized for streaming data state-of-the-art inverted index. However, the proposed solution runs on a single machine and thus unable to multiple massive streams for scalability reasons.

To the best of our knowledge, the only work dealing with distributed streaming similarity joins is [23]. It proposes a distributed streaming similarity join framework that employs a length-based filter to distribute the data across a cluster of nodes. Because the lengths of the incoming tuples might change over time, an adaptive algorithm is also proposed to recalculate the bounds for the length segments based on the online collected statistics. In addition, in order to reduce the computations in a node, an inverted index is built accompanied with a bundle structure to reduce the indexed records. However, the authors consider full history joins without proposing any retention policy which is crucial when dealing with endless streams. In addition, the proposed length-based filter will struggle to scale out efficiently when all the incoming sets are of similar length.

**Load-Balancing on Streams.** Load balancing is a native concern in distributed stream processing environments, since the statistical properties of the data change frequently and the systems need to adapt to achieve full potential. To ensure load balancing, [12] proposes a new dataflow join operator that can adaptively distribute records to nodes and perform state repartitioning through a locality-aware migration strategy. In [18], a streaming variation of the HyperCube algorithm [2] is presented. The authors divide the incoming records to heavy and light hitters and process each heavy hitter in a Hypercube Grid. The characterization of a tuple as heavy

or light hitter, as well as the size of each HyperCube grid is adapted to the statistics that are gathered online for the join values. These works focus on distributing the load and do not present solutions for reducing the needed computations. However, a unified solution with a load balancing scheme tailored to the distribution scheme can achieve both goals and provide better results.

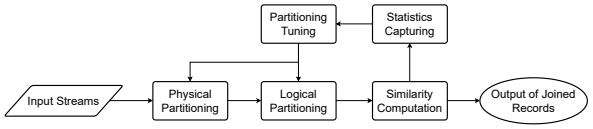


**Figure 2: The Similarity Join process**

**Our approach.**To tackle the streaming challenges we propose a partitioning scheme based on inner and outer partitions inspired by the general metric space approaches from MapReduce. First, we partition the incoming records from our input streams to different physical nodes based on the proximity to a node's representative centroid. Afterwards, we create tighter partitions within a node in a logical way by leveraging the provided similarity threshold and creating new logical partitions based on the incoming records. The last step of our workflow is the actual similarity computation and the output of the wanted joined pairs. The similarity computations are restricted to candidate pairs through our tight logical partitions. In additions, in order to ensure adaptivity, we collect statistics from the similarity computations sub-task to re-calibrate both our logical and physical partitions. It is important to notice that in this project we optimize for high dimensional data. Based on the latest advances in Deep Learning-based Entity Resolution [4, 11, 25], word embeddings can be used to capture more effectively the similarities between records. However, due to their high dimensionality, the similarity computations between two embeddings is very inefficient. We aim to tackle this inefficiency enabling the use of word embeddings for similarity joins on streams.

**Evaluation.** We plan to evaluate our solution on datasets from [23], real-world datasets provided by our industrial partner, and synthetic datasets. We plan to compare it against the discussed state of the art solution [23], and baselines from the general metric space. Since our goal is to provide real-time results, our evaluation will be based on metrics like throughput and latency.

## 4 CONCLUSION

Summarizing, in this paper we present the exciting challenges and open problems of *streaming data integration* and a research plan that aims to provide scalable methods to tackle them. We discuss three envisioned methods to perform profiling, discovery and similarity joins on streams. In addition, we present related work for each task and we discuss their limitations when dealing with streams. Finally, we shortly present our approach for distributed streaming similarity joins by leveraging adaptive data partitions.

## REFERENCES


[1] Z. Abedjan, L. Golab, and F. Naumann. Data profiling: A tutorial. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, page 1747–1751, New York, NY, USA, 2017. Association for Computing Machinery.

[2] F. N. Afrati and J. D. Ullman. Optimizing joins in a map-reduce environment. In *Proceedings of the 13th International Conference on Extending Database Technology*, EDBT '10, page 99–110, New York, NY, USA, 2010. Association for Computing Machinery.

[3] A. Arasu, V. Ganti, and R. Kaushik. Efficient exact set-similarity joins. In *Proceedings of the 32nd international conference on Very large data bases*, pages 918–929, 2006.

[4] R. Cappuzzo, P. Papotti, and S. Thirumuruganathan. Creating embeddings of heterogeneous relational datasets for data integration tasks. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, SIGMOD '20, page 1335–1349, New York, NY, USA, 2020. Association for Computing Machinery.

[5] R. Castro Fernandez, Z. Abedjan, F. Koko, G. Yuan, S. Madden, and M. Stonebraker. Aurum: A data discovery system. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 1001–1012, 2018.

[6] S. Chaudhuri, V. Ganti, and R. Kaushik. A primitive operator for similarity joins in data cleaning. In *22nd International Conference on Data Engineering (ICDE'06)*, pages 5–5, 2006.

[7] G. Cormode, M. Garofalakis, P. J. Haas, and C. Jermaine. Synopses for massive data: Samples, histograms, wavelets, sketches. *Foundations and Trends® in Databases*, 4(1–3):1–294, 2011.

[8] A. Das Sarma, Y. He, and S. Chaudhuri. Clusterjoin: A similarity joins framework using map-reduce. *Proc. VLDB Endow.*, 7(12):1059–1070, Aug. 2014.

[9] G. De Francisci Morales and A. Gionis. Streaming similarity self-join. *Proc. VLDB Endow.*, 9(10):792–803, June 2016.

[10] D. Deng, G. Li, S. Hao, J. Wang, and J. Feng. Massjoin: A mapreduce-based method for scalable string similarity joins. In *2014 IEEE 30th International Conference on Data Engineering*, pages 340–351, 2014.

[11] M. Ebraheem, S. Thirumuruganathan, S. Joty, M. Ouzzani, and N. Tang. Distributed representations of tuples for entity resolution. *Proc. VLDB Endow.*, 11(11):1454–1467, July 2018.

[12] M. ElSeidy, A. Elguindy, A. Vitorovic, and C. Koch. Scalable and adaptive online joins. page 16, 2014.

[13] D. Karapiperis, A. Gkoulalas-Divanis, and V. S. Verykios. Summarization algorithms for record linkage. In M. H. Böhlen, R. Pichler, N. May, E. Rahm, S. Wu, and K. Hose, editors, *Proceedings of the 21st International Conference on Extending Database Technology, EDBT 2018, Vienna, Austria, March 26-29, 2018*, pages 73–84. OpenProceedings.org, 2018.

[14] L. Kolb, A. Thor, and E. Rahm. Dedoop: Efficient deduplication with hadoop. *Proc. VLDB Endow.*, 5(12):1878–1881, Aug. 2012.

[15] L. Kolb, A. Thor, and E. Rahm. Multi-pass sorted neighborhood blocking with mapreduce. *Comput. Sci.*, 27(1):45–63, Feb. 2012.

[16] L. Orr, M. Balazinska, and D. Suciu. Entropydb: a probabilistic approach to approximate query processing. *VLDB Journal International Journal on Very Large Data Bases*, 29(1), 2020.

[17] G. Papadakis, G. Mandilaras, L. Gagliardelli, G. Simonini, E. Thanos, G. Giannakopoulos, S. Bergamaschi, T. Palpanas, and M. Koubarakis. Three-dimensional entity resolution with jedai. *Information Systems*, 93:101565, 2020.

[18] Y. Qiu, S. Papadias, and K. Yi. Streaming hypercube: A massively parallel stream join algorithm. In M. Herschel, H. Galhardas, B. Reinwald, I. Fundulaki, C. Binnig, and Z. Kaoudi, editors, *Advances in Database Technology - 22nd International Conference on Extending Database Technology, EDBT 2019, Lisbon, Portugal, March 26-29, 2019*, pages 642–645. OpenProceedings.org, 2019.

[19] A. Saeedi, M. Nentwig, E. Peukert, and E. Rahm. Scalable matching and clustering of entities with famer. *Complex Systems Informatics and Modeling Quarterly*, (16):61–83, 2018.

[20] R. Vernica, M. J. Carey, and C. Li. Efficient parallel set-similarity joins using mapreduce. SIGMOD '10, page 495–506, New York, NY, USA, 2010. Association for Computing Machinery.

[21] Y. Wang, A. Metwally, and S. Parthasarathy. Scalable all-pairs similarity search in metric spaces. KDD '13, page 829–837, New York, NY, USA, 2013. Association for Computing Machinery.

[22] C. Xiao, W. Wang, X. Lin, J. X. Yu, and G. Wang. Efficient similarity joins for near-duplicate detection. *ACM Trans. Database Syst.*, 36(3), Aug. 2011.

[23] J. Yang, W. Zhang, X. Wang, Y. Zhang, and X. Lin. Distributed streaming set similarity join. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pages 565–576. IEEE, 2020.

[24] M. Zhang, M. Hadjieleftheriou, B. C. Ooi, C. M. Procopiuc, and D. Srivastava. Automatic discovery of attributes in relational databases. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, page 109–120, New York, NY, USA, 2011. Association for Computing Machinery.

[25] W. Zhang, H. Wei, B. Sisman, X. L. Dong, C. Faloutsos, and D. Page. Autoblock: A hands-off blocking framework for entity matching. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, WSDM '20, page 744–752, New York, NY, USA, 2020. Association for Computing Machinery.