Pipelined Implementation of a Parallel Streaming Method for Time Series Correlation Discovery on Sliding Windows

Boyan Kolev[1], Reza Akbarinia[1], Ricardo Jimenez-Peris[2], Oleksandra Levchenko[1], Florent Masseglia[1], Marta Patino[3] and Patrick Valduriez[1]
[1]*Inria and LIRMM, Montpellier, France*
[2]*LeanXcale, Madrid, Spain*
[3]*Universidad Politecnica de Madrid (UPM), Madrid, Spain*

Keywords: Time Series Correlation, Data Stream Processing, Distributed Computing.

Abstract: This paper addresses the problem of continuously finding highly correlated pairs of time series over the most recent time window. The solution builds upon the ParCorr parallel method for online correlation discovery and is designed to run continuously on top of the UPM-CEP data streaming engine through efficient streaming operators. The implementation takes advantage of the flexible API of the streaming engine that provides low level primitives for developing custom operators. Thus, each operator is implemented to process incoming tuples on-the-fly and hence emit resulting tuples as early as possible. This guarantees a real pipelined flow of data that allows for outputting early results, as the experimental evaluation shows.

1 INTRODUCTION

Consider a big number of streams of time series data (e.g. stock trading quotes), where we need to find highly correlated pairs for the latest window of time (say, one hour), and then continuously slide this window to repeat the same search (say, every minute). Doing this efficiently and in parallel could help gather important insights from the data in real time (Figure 1). This has been recently addressed by the ParCorr parallel incremental sketching approach (Yagoubi et al., 2018), which scales to 100s of millions of parallel time series, and achieves 95% recall and 100% precision. An interesting aspect of the method is the discovery of time series that are highly correlated to a certain subset of the time series, which we call targets (Figure 2). This concept has many applications in different domains (finance, retail, etc.), where we would like to use the correlates of a target as predictors to forecast the value of the target for the next time window.

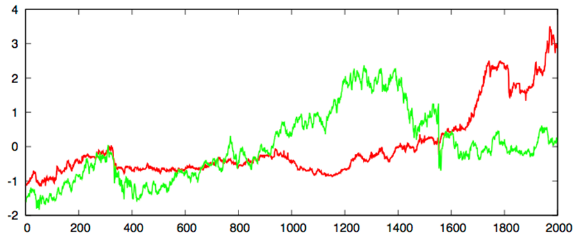

Figure 1: Example of a pair of time series that the method found to be highly correlated over the first several sliding windows of 500 time points, but not thereafter.

Such challenges have been identified by use case scenarios, defined in the scope of the CloudDBAppliance project (CDBA, 2019), which aims to provide a database-as-a-service appliance integrating several data management technologies, designed to scale vertically on many-core architectures. These include an operational database, an analytical database, a data lake, and a data streaming engine. To face these requirements, the ParCorr method was implemented as a continuous query for the highly scalable streaming engine.

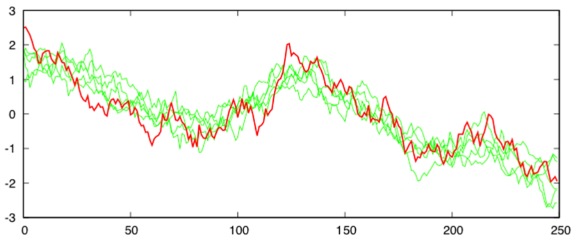

Figure 2: Example of a target time series (red) and its top correlates (green) discovered by the method.

Kolev, B., Akbarinia, R., Jimenez-Peris, R., Levchenko, O., Masseglia, F., Patino, M. and Valduriez, P.
Pipelined Implementation of a Parallel Streaming Method for Time Series Correlation Discovery on Sliding Windows.
DOI: 10.5220/0008191304310436
In *Proceedings of the 8th International Conference on Data Science, Technology and Applications (DATA 2019)*, pages 431-436
ISBN: 978-989-758-377-3

In a previous paper (Kolev et al., 2019), we presented details of the generic implementation of the method. In this paper, we focus on a modification that allows for efficient data pipelining, considering the fact that, at each window, target time series can be hashed as a first step and then all the others can be correlated to the targets in a pipeline. The rest of the paper gives a brief overview of the streaming engine and the ParCorr method, followed by a description of the pipelined implementation in comparison with a naïve one, which are then experimentally evaluated.

2 UPM-CEP: STREAMING ENGINE OVERVIEW

Stream Processing (SP) is a novel paradigm for analyzing in real-time data captured from heterogeneous data sources. Instead of storing the data and then processing it, the data is processed on the fly, as soon as it is received, or at most a window of data is stored in memory. SP queries are continuous queries run on a (infinite) stream of events. Continuous queries are modeled as graphs where nodes are SP operators and arrows are streams of events. SP operators are computational boxes that process events received over the incoming stream and produce output events on the outgoing streams. SP operators can be either stateless (such as projection, filter) or stateful, depending on whether they operate on the current event (tuple) or on a set of events (time window or number of events window). Several implementations went out to the consumer market from both academy and industry, such as Borealis (Ahmad et al., 2005), Infosphere (Pu et al., 2001), Storm[1], Flink[2] and StreamCloud (Gulisano et al., 2012). Storm and Flink followed a similar approach to the one of StreamCloud in which a continuous query runs in a distributed and parallel way over several machines, which in turn increases the system throughput in terms of number of tuples processed per second. The streaming engine UPM-CEP (Complex Event Processing) adds efficiency to this parallel-distributed processing being able to reach higher throughput using less resources. It improves the network management, reduces the inefficiency of the garbage collection by implementing techniques such as object reutilization and takes advantage of the novel Non Uniform Memory Access (NUMA) multicore architectures by minimizing the time spent in context switching of SP threads/processes.

The UPM-CEP JCEPC (Java CEP Connectivity) driver hides from the applications the complexity of the underlying cluster. Applications can create and deploy continuous queries using the JCEPC driver as well as register to the source streams and subscribe to output streams of these queries. During the deployment the JCEPC driver takes care of splitting a query into sub-queries and deploys them in the CEP cluster. Some of those sub-queries can be parallelized.

3 ParCorr: METHOD OVERVIEW

The ParCorr time series correlation discovery algorithm (Yagoubi et al., 2018) is based on a work on fast window correlations over time series of numerical data (Cole et al., 2005), and concentrates on adapting the approach for the context of a big number of parallel data streams. The analysis is done on sliding windows of time series data, so that recent correlations are being continuously discovered in nearly real-time. At each move of the sliding window, the latest elements of the time series are taken as multi-dimensional vectors. As a similarity measure between such vectors, we take the Euclidean distance, since it is related to the Pearson correlation coefficient if applied to normalized vectors.

Since the sliding window can result in a very high number of dimensions of time series vectors, which makes them very expensive to be compared to each other, a major challenge the algorithm addresses is the reduction of the dimensionality in a way that nearly preserves the Euclidean distances. For this purpose, random projection approach is adopted, where each high-dimensional vector is transformed into a low-dimensional one (called "sketch" of the vector), by applying a product with a specific transformation matrix, the elements of which are randomly selected from the values of either -1 or 1. This approach guarantees with high probability that the distance between any pair of original vectors correspond to the distance between their sketches.

Furthermore, to simplify the comparing across sketches, each sketch vector is partitioned into subvectors (e.g. two-dimensional), so that for example a 30-dimensional sketch vector is broken into 15 two-dimensional subvectors. Then, discrete grid structures (in the example, 15 two-dimensional grids) are built and subvectors are assigned to grid cells, so that close subvectors are grouped in the same grid cells. This process essentially performs a

[1] http://storm.apache.org/

[2] https://flink.apache.org/

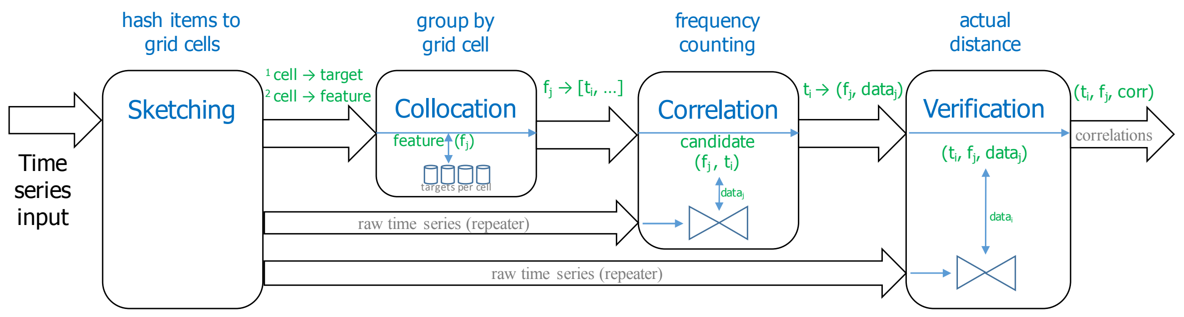

Figure 3: Streaming operators architecture with pipelined data flow within the operators Collocation, Correlation, and Verification.

locality-sensitive hashing of high-dimensional time series vectors, where close vectors are discovered by searching for pairs of vectors, which are represented together in a high number of grid cells. Since this can output false positives, the candidate pairs are explicitly verified by computing the actual distance between them.

This outlines four main steps of the algorithm:

- Sketching: computation and partitioning of sketches;
- Collocation: grouping together all time series assigned to the same grid cell;
- Correlation: finding frequently collocated pairs as candidates for correlation;
- Verification: computing the actual correlation of each candidate pair to filter out false positives.

To provide prediction capabilities, the method takes into account the correlates of time series that are considered of interest for prediction and called “targets”. Only correlated pairs that involve at least one target time series are considered for discovery.

4 PIPELINED IMPLEMENTATION

The generic implementation involves four stateful streaming operators (Kolev et al., 2019), each processing incoming tuples in the context of the current window, taking into account the current state of the window. Thanks to the flexible API of the UPM-CEP streaming engine that provides low level primitives for implementing custom operators, each operator can process incoming tuples on-the-fly and hence emit resulting tuples as early as possible. This guarantees a real pipelined flow of data that allows for outputting early results. This section includes descriptions of the operators with focus on the improvements enabled through the custom design.

The current modification presented hereby concentrates on extensions of the method, guided by the requirements of the CloudDBAppliance project’s use cases, which include a filtering that enables the discovery of time series that are highly correlated only with a certain subset of all the time series, called “targets”. The rest of the series are called “features” and the objective is to output the most relevant features for each target.

4.1 Architecture

Figure 3 shows the architecture of the streaming operators and the data flow within the pipelined implementation. Although these operators are stateful, large part of their inputs are processed in a pipelined way, allowing to emit resulting tuples as early as possible. Therefore, compared to a naïve implementation based on high-level streaming operators with custom transformation functions, our implementation gives much lower latency of the first output tuples at each window slide.

The parallelization of the algorithm is quite straightforward – sketches of time series vectors on parallel data streams are computed in parallel, which is followed by an additional shuffle step that groups together the identifiers of time series that fit in the same grid cell; then groups are explored for discovering frequent pairs. Since the streaming engine operates in a distributed environment, operators have multiple instances, handling different partitions of data in parallel. This requires shuffles of intermediate data across operator instances and the partitioning is based on a key from the schema of the intermediate dataset. In Figure 3, we use the *Key=>Value* notation to show which fields are used as partition keys.

4.2 Streaming Operators

The Sketching operator computes a hashing of all time series within the current window by assigning each time series item (only the identifier, without the data) to a grid cell in a number of grids. As a first step, it emits the cell assignment of only the target time series and then emits the rest of the items (features).

Then, the Collocation operator first memorizes for each cell a set of all assigned targets, and at the next step processes on-the-fly each of the features by emitting directly the set of targets collocated in the same cell with the feature. Conceptually, this operator efficiently combines the functionality of a group-by followed by a hash join, by constructing the hashed side of the join during the group-by step.

The Correlation operator counts the number of occurrences of each (feature, target) pair and as soon as it exceeds a certain threshold, the pair is considered a candidate for correlation. To perform this counting, each instance of the operator maintains incrementally, for each of the features, a counter map that keeps the current count for each target. When a new tuple arrives (recall that an incoming tuple is a feature id mapped to a set of target ids), the counter map for the feature gets updated by incrementing the counts of the targets present in the tuple. If some of these counts is already equal to the threshold, the corresponding (feature, target) pair is immediately promoted as candidate. Conceptually, this step of the operator is equivalent to a counting aggregate on a keyed stream followed by a filter on the counted value. Hence, the candidate selection with standard streaming operators would first fully process the aggregate in order to compute the counts for filtering.

As soon as a candidate is selected, the Correlation operator immediately retrieves the time series data of the feature from a local copy of the current window, partitioned by time series identifier. This lookup is done using a non-blocking symmetric hash join between the candidate pairs stream and the raw time series repeater stream. Thus, the custom implementation of this operator combines two stateful operations (grouping with counting and then join) into a single one in a way that allows resulting tuples to be emitted as soon as possible.

The same symmetric hash join strategy is applied at the Verification operator to retrieve the time series data of the target, compute the actual correlation of the candidate (target, feature) pair, and, if it exceeds the desired correlation threshold, output it. Note that these joins also benefit from a long-term state within the operator that keeps in a buffer the data for previous windows, so that the repeater streams need to incrementally stream only the newest basic window for the consuming operator to update the current window data. This long-term state is yet another improvement that benefits from the custom design, compared to standard implementations of symmetric hash joins.

So, except for the first operator, which has anyway to wait for the entire window to be collected in order to compute the hashing, all the other operators process the “features” part of the stream in a pipeline, so that a target-feature association be output as early as possible.

4.3 Comparison with a Naïve Implementation

We compare the benefits of the pipelined implementation with a naïve one, where some of the operators must wait for the entire window data to be collected before shuffling resulting tuples to the next operator. This can be easily achieved through the use of standard operators, but results in emitting all candidates for correlations at almost the same time, as opposed to the pipelining approach, where early results can be observed shortly after each window slide.

We have implemented the same algorithm on Apache Spark, using standard operators (such as reduceByKey and join), in order to compare the response times of the first and the last emitted tuples per each sliding window. The major impact on the latency of the first results, compared to the pipelined implementation, has the candidate selection step, where counting is done per candidate through a stateful operation that requires the entire window to be processed.

5 EXPERIMENTS

We consider two implementations of the ParCorr method as follows:

- CDBA: the pipelined implementation on top of the UPM-CEP streaming engine for the CloudDBAppliance platform. It leverages custom streaming operators to minimize the amount of exchanged intermediate data and output resulting tuples as early as possible.
- Naïve: the baseline implementation with standard streaming operators for Apache Spark (version 2.3.3), with custom transformation functions.

The experiments were carried out on a many-core architecture platform, provided for the integration of

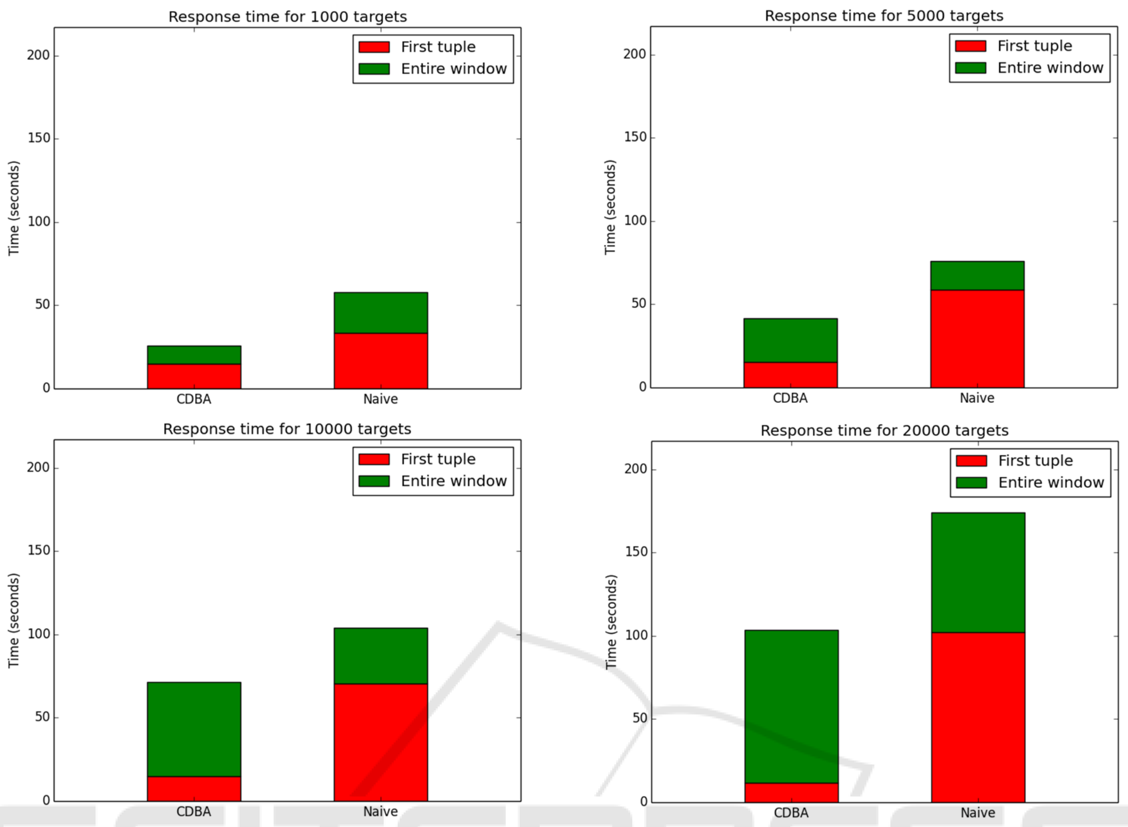

Figure 4: Experimental evaluation of the response times for the pipelined (CDBA) implementation and the naïve one. The dataset contains 1 million random walk time series, each of length 510 points. The number of target time series is varied between 1k, 5k, 10k, and 20k. Response times were measured for each window of size 250, sliding with a step of 20 points (14 windows in total). Average latency to emit the first and the last tuple after the beginning of each window is displayed.

the CloudDBAppliance components. The platform utilizes a configurable number of Intel Xeon Platinum 8153 @2GHz processors, with 3TB of main memory. In our experiments, we have used a level of parallelism of 48 workers, for both implementations. Datasets were loaded in memory, so I/O cost is not considered for either of the solutions.

The two implementations of the method were evaluated on a synthetic random walk dataset of 1 million time series, each consisting of 510 values. At each time point, the random walk generator draws a random number from a Gaussian distribution N(0,1), then adds the value of the last number to the new number. Sliding windows of size 250 with a step of 20 points (14 windows in total) have been taken into account to measure the response times of the first and the last emitted tuple after the beginning of each window.

Figure 4 shows the results of the experimental evaluation, comparing the two implementations, with respect to latency of the first results emitted after the beginning of a window and the response time of the entire window result. CDBA shows an advantage in the total window processing time, thanks to its optimized data flow and customized allocation of local (to the operator instances) long-term memory storage. In addition to the overall performance, CDBA also shows a significant advantage in the latency of the first results, thanks to the pipelining approach provided by the custom operator logic.

Raising the number of targets, as expected, raises the total execution time, as the number of candidates is proportional. However, with respect to the first results latency, the number of targets shows no particular impact on the pipelined approach. An interesting observation is that the latency even decreases as the number of targets increases. This can be explained by the higher number of targets associated to each feature (at the Correlation step), which results in a higher probability for early presence of a frequent feature/target pair. This effect, however, cannot be observed with the naïve implementation, due to its inability to detect early candidates.

6 CONCLUSIONS

We presented a parallel streaming implementation of the ParCorr method for window correlation discovery on time series data, in the context of the CloudDBAppliance project, adapted to discover correlations with respect to a particular subset of the input series, called "targets". The implementation leverages the development of custom streaming operators that boosts the performance and minimizes the response time by optimizing intra-operator communication and utilizing pipelining of intermediate data. The experimental study evaluates the performance benefits of this implementation, compared to a naïve streaming setup, with respect to latency of the first results and response time of the entire output at each time window. Our implementation shows a significant advantage in starting to emit results very shortly after a window slides, thanks to the pipelining approach provided by the custom operator logic. In addition, the overall response time per window gets also improved.

ACKNOWLEDGEMENTS

This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732051.

REFERENCES

Ahmad, Y., Berg, B., Cetintemel, U., Humphrey, M., Hwang, J., Jhingran, A., Maskey, A., Papaemmanouil, O., Rasin, A., Tatbul, N., Xing, W., Xing, Y., Zdonik, S., 2005, Distributed Operation in the Borealis Stream Processing Engine. *ACM SIGMOD International Conference on Management of Data* (2005), pp 882–884.

CDBA, 2019. *The CloudDBAppliance Project*. http://clouddb.eu

Cole, R., Shasha, D., Zhao, X., 2005. Fast Window Correlations over Uncooperative Time Series. *In: Proceedings of the International Conference on Knowledge Discovery and Data Mining (SIGKDD)*. ACM, pp 743–749.

Gulisano, V., Jimenez-Peris, R., Patino-Martinez, M**,** C. Soriente, C.**,** Valduriez, P., 2012. Streamcloud: An Elastic and Scalable Data Streaming System. *IEEE Trans. Parallel Distrib. Syst.* 23(12), pp 2351–2365.

Kolev, B., Akbarinia, R., Jimenez-Peris, R., Levchenko, O., Masseglia, F., Patino, M., Valduriez, P., 2019. Parallel Streaming Implementation of Online Time Series Correlation Discovery on Sliding Windows with Regression Capabilities. *Proceedings of the 9th International Conference on Cloud Computing and Services Science (CLOSER)*, pp 681–687.

Pu, C., Schwan, K., Walpole, J., 2001. Infosphere Project: System Support for Information Flow Applications. *SIGMOD Record* 30(1) (2001), pp 25–34.

Yagoubi, D.-E., Akbarinia, R., Kolev, B., Levchenko, O., Masseglia, F., Valduriez, P., Shasha, D., 2018. ParCorr: Efficient Parallel Methods to Identify Similar Time Series Pairs across Sliding Windows. *Data Mining and Knowledge Discovery*, vol. 32(5), pp 1481-1507. Springer.