# Detection of Highly Correlated Live Data Streams

Rakan Alseghayer, Daniel Petrov, Panos K. Chrysanthis, Mohamed Sharaf*, Alexandros Labrinidis
Department of Computer Science, University of Pittsburgh, USA
*University of Queensland, Brisbane, Australia
{ralseghayer,dpetrov,panos,labrinid}@cs.pitt.edu,*m.sharaf@uq.edu.au

## ABSTRACT

More and more organizations (commercial, health, government and security) currently base their decisions on real-time analysis of fast arriving, large volumes of data streams. For such analysis to lead to actionable information in real-time and at the right time, the most recent data needs to be processed within a specified delay target. Effective solutions for analysis of such data streams rely on two techniques, (1) incremental sliding-window computation of aggregates, to avoid unnecessary recomputations and (2) intelligent scheduling of computational steps and operations. In this paper, we propose a solution that combines both of these techniques to find highly correlated data streams in real-time, using the Pearson Correlation Coefficient as a correlation metric for two windows of data streams. Specifically, we propose to partition a set of data streams into micro-batches that capture the delay target, use sliding windows within a range as the subsequences of values exhibiting a certain level of correlation, utilize the idea of sufficient statistics to incrementally compute the Pearson Correlation Coefficient of pairs of sliding windows, and adopt a deadline-aware priority scheduling to detect the highly correlated pairs of data streams. Our experimental results show that our scheme and in particular our Price-DCS with warm start scheduling algorithm outperform existing ones and enable high degree of interactivity in correlating live data streams micro-batches.

## CCS CONCEPTS

**•Information systems →Information retrieval; Users and interactive retrieval;** *Personalization;*

## KEYWORDS

data streams, data exploration, correlation, search, subsequence

**ACM Reference format:**
Rakan Alseghayer, Daniel Petrov, Panos K. Chrysanthis, Mohamed Sharaf*, Alexandros Labrinidis. 2017. Detection of Highly Correlated Live Data Streams. In *Proceedings of BIRTE '17, Munich, Germany, August 28, 2017,* 8 pages.
DOI: 10.1145/3129292.3129298


*BIRTE '17, Munich, Germany*

DOI: 10.1145/3129292.3129298

## 1 INTRODUCTION

**Motivation** More and more organizations (commercial, health, government and security) currently base their decisions on real-time analysis of business and operational data in order to stay competitive. Towards this, they deploy a variety of monitoring applications to analyze fast arriving, large volumes of data streams. Data analysts explore such large volumes of data streams, typically representing time series of raw measures, looking for valuable insights and interesting events.

A common method to get a better understanding of the observed behavior conveyed in a set of data streams is to find correlations in the data streams [6]. The correlation can be also used as a source to finding similarity measures faster [9], running threshold queries [13], or reducing the size of the data, but preserving some of its characteristics [7]. The following example illustrates the practicality of finding correlated windows of data streams and using them as a source of insights:

***Example 1:*** *Consider a data center, operating 10,000 computers, which hosts an order of magnitude more virtual servers. A monitoring system keeps track of 20 different counters per computer - for CPU core temperature, power supply voltage, memory and network utilization, etc. Each computer reports its counters to the monitoring system every 60 seconds. Each batch of reported data contains 12 consecutive measurements (taken 5 seconds apart). The Operations team can timely detect problematic servers and identify higher order dependencies by finding deviations from the average load per computer and negatively correlated pairs of windows of data streams as the data arrives.*

**Challenges** Finding correlations in data streams is a challenging task. A way to address this challenge is to index the data series [2, 3, 15]. Often the data is extremely large—it does not fit into memory or data from earlier periods is of no interest anymore. Predominantly the users are looking for pairs of highly (negatively) correlated data streams over a short period of time. The high number of data streams implies an even bigger number of pairs—precisely $\frac{n*(n-1)}{2}$ pairs for $n$ data streams. The time to explore completely all pairs on one computer may be prohibitively long.

In our example the total number of counters (i.e., data streams), which the IT specialists should analyze is $10,000 \times 20 = 200,000$. As every computer reports its counters once every 60 seconds, a time frame of 5 minutes will contain 60 measurements per counter. This means that there will be a total of $60 \times 200,000 = 12,000,000$ numbers generated every five minutes of uptime of the data center. The number of different pairs is equal to the number of combinations of 2 counters

$$C_2^{200000} = \frac{200000!}{2!(200000-2)!} = 19,999,900,000 \quad (1)$$

—almost 20 billion pairs. Traversing the data and calculating the correlation is computationally expensive and induces significant

delay in the production of results. This further complicates data analysis during which a user looks for incremental results and demands fast answers. The challenge is exacerbated when the demand is for answers in *real-time* and for a large set of *live* data streams.

**Objective** This real example clearly illustrates the need to develop algorithms that (1) quickly identifies windows of highly correlated data streams and (2) provides results in real-time. In our previous work on data exploration, we addressed the former and proposed two novel algorithms, *iBRAID* and *PriCe* [10]. In this paper, we address the latter by proposing a real-time framework, which identifies correlated data streams and provides incremental results in real-time. Our solution is based on our prior work that uses the Pearson Correlation Coefficient as a metric of correlation of two windows of data streams and on the hypothesis that production of results by a specific deadline can be achieved by integrating caching and real-time scheduling principles.

**Contributions** In this paper we make the following contributions:

- We propose a framework for detecting correlated live data streams in which real-time data analytical processing is performed in micro-batches whose inter-arrival rate defines the processing deadline. (Sec. 2)
- We develop *Detection of Correlated Streams* (*DCS*) which is a mode of operation that utilizes the information of data stream intervals to reduce the processing time in examining individual pairs of data streams for correlations. Specifically, *DCS* combines the concept of early termination and pruning to identify the data streams in interest. Furthermore, *DCS* distinguishes its execution into *cold start* and *warm start* depending on whether or not the immediately previous micro-batch execution identified correlated pairs of streams. (Sec. 3)
- We present an experimental evaluation of *DCS* applied to *iBRAID* and *PriCe*. Our results using a real dataset show that *DCS* enhanced the performance of detecting correlated pairs by up to 1.8 times. Furthermore, they show the advantage of *Price-DCS* with warm start compared to *Price-DCS* with cold start, and assesses two variations of warm start, which exploit differently prior knowledge of number of correlated windows in pairs. (Sec. 4)

## 2 SYSTEM MODEL

Without loss of generality, we consider a (monitoring) system that receives data from $n$ data streams. Each data point in a data stream is a tuple $t$ consisting of a timestamp $ts$ and a numeric value $val$ ($t = (ts, val)$). The timestamp captures the moment in time, when the tuple was produced.

The data is produced at high velocity. The different streams produce the consecutive tuples at the same rate, and they are all synchronized. However, there are techniques to determine missing values, also to synchronize data, which arrives at different rates, but they are beyond the scope of this paper.

The real-time analytical processing is performed in *micro-batches*.

*Definition 2.1.* A micro-batch is a group of synchronized tuple subsequences over a set of data streams defined by a timestamp *interval* $I$.

In our system, each micro-batch, whether of the same or different data streams, is of the same size, i.e., contains the same number of tuples with consecutive timestamps within the interval. The inter-arrival time of two consecutive micro-batches specify the maximum computational time for processing a micro-batch.

*Definition 2.2.* The inter-arrival time is the *delay target* or *deadline* $d$ by which the last result can be produced while analyzing a micro-batch.

In real-time processing, ideally, the deadline $d$ equals to the interval ($d = I$) so that there is no delay gap in processing between two consecutive micro-batches. However, it is expected to be a bit longer due to various overheads in the system, including any pre-processing of micro-batches.

In this paper, we focus on analytical processing that finds correlated data streams in real-time, using the Pearson Correlation Coefficient (PCC) as a correlation metric for two sliding windows of data streams.

*Definition 2.3.* Given two numeric data streams $x$ and $y$ of equal length $m$, the PCC is calculated with the following formula:

$$corr(x, y) = \sum_{i=1}^{m} \frac{(x_i - \mu_x)(y_i - \mu_y)}{\sigma_x \sigma_y} \quad (2)$$

where $\mu_x$ is the average (or mean) of the values of $x$, $\mu_y$ is the mean of the values of $y$, $\sigma_x$ and $\sigma_y$ are the standard deviations of the values of $x$ and $y$, respectively.

*Definition 2.4.* Two sliding windows of the same range $w$ with a slide of 1 are correlated when the PCC is more than a given threshold $\tau$ (PCC $\geq \tau$).

*Definition 2.5.* A pair of data streams in a micro-batch is correlated when it contains at least $A$ correlated sliding windows with threshold $\tau$.

The windows, which meet the criterion, may be consecutive or stratified over the interval defining a micro-batch.

## 3 DCS FRAMEWORK

In this section, we discuss our contribution *DCS* that enables the fast detection of correlated live streams. *DCS* extends our work on data exploration, where we focused on generating early results, to meet real-time constraints. First, we formally defined our problem. Then, after reviewing the basic algorithms *iBRAID* and *PriCe* from our prior work, we discuss how these are enhanced by *DCS*. Finally, we introduce the concept of *cold start* and *warm start* of analysis to exploit the results of the proceeding analyses.

### 3.1 Problem Statement

We begin by formalizing the algorithmic problem that lays in the epicenter of DCS.

*PROBLEM (Real-time Correlation Detection (RCD)):* Given a micro-batch $B$ of a set of data streams $DS$ with an arrival interval $I$, perfectly synchronized and with no missing tuples, and a deadline $d$, detect the number of correlated pairs of data streams, each of which has $A$ correlated sliding windows with a PCC threshold of $\tau$, by the deadline $d$.

The optimum will be the number of identified correlated pairs in a micro-batch to be equal to the actual, total number of correlated pairs. Hence, our optimization goal in DCS is to maximize the number of identified number of pairs within a deadline. Formally, the ratio of number of determined correlated pairs to the total number of correlated pairs is close to 1 and the metric:

$$DCS\text{-}Precision = \frac{\#\ identified\ correlated\ pairs}{Total\ \#\ correlated\ pairs} \quad (3)$$

## 3.2 Base Algorithms

*3.2.1 iBRAID.* It is a round-robin scanning algorithm that uses incremental computation of PCC. It analyzes the pairs of data streams in a micro-batch sequentially, starting from the first tuple for all data streams. It calculates the sufficient statistics—sum of the elements in each window, the sum of the squares of the elements of each window, and the inner crossproduct of the elements of the two windows for which the correlation is calculated—that are needed to calculate the PCC efficiently [11]. Next, it calculates the PCC for all pairs of windows, starting from the first tuple. Once this is done, the windows are slid further by one tuple, the sufficient statistics are updated incrementally - the first tuple is expired/subtracted from them, and the new tuple is added. The PCC is calculated again for all pairs. Then, it keeps analyzing all data streams by a single tuple, augment the sufficient statistics incrementally, and recalculate the PCC. This is done until the whole micro-batch is analyzed.

*iBRAID* has four key advantages: (1) it is accurate, (2) easy to implement, (3) does not cause "starvation" among the pairs, and (4) it reduces the computations by half due to the usage of the sufficient statistics. *iBRAID* is experimentally shown to perform well for data streams whose data is uniformly distributed and for low correlation thresholds ($\tau < 0.5$).

*3.2.2 PriCe.* It is a more informed scanning algorithm; it uses a priority function to analyze the pairs of windows while reusing partial PCC computations as *iBRAID*. It analyzes the most promising pair first, which is the one with the highest priority function value:

$$Pr = PCC * (M/totalExp)/C \quad (4)$$

where *PCC* is the most recent calculated correlation for a pair of sliding windows that belong to the same pair of data streams, $M$ is the number of correlated sliding windows found the corresponding pair of data streams so far, $totalExp$ is the total number of analyzed pair of sliding windows, and $C$ is the cost of analyzing a pair of sliding windows in number of computations. The default values are $PCC = 1$, $M = 0$, $totalExp = 1$, and $C = 1$.

The cost in the priority function is the number of operations needed to calculate the sufficient statistics for a pair of sliding windows. For example, if a pair of data streams shares one data stream with another pair, then, the more advanced one (i.e., the one that has a higher timestamp in the interval) has already calculated the sums and the sum of the squares for the tuples of that data stream. This leaves the lagging behind pair with lower cost to slide its windows, since the more advanced one has already computed some of the sufficient statistics for that shared data stream.

*PriCe* gives the highest priority to the pair of data streams that has a history of high number of correlated sliding windows analyzed and high recent calculated PCC. This captures the idea of space locality and temporal locality: where *space locality* is captured by the recently calculated PCC, and *temporal locality* is captured by the ratio of the number of correlated sliding windows to the total number of analyzed pairs of sliding windows.

## 3.3 DCS

Our contribution *DCS* enhances *iBRAID* and *PriCe* by reducing the amount of processing needed to detect correlated pairs of data streams and by avoiding unnecessary computations. It is based on the concepts of early termination and pruning. The first enhancement is while analyzing a pair of data streams, we can stop analyzing that pair, as soon as we find $A$, the specified number of correlated sliding windows. In this way, we save processing time, which enables us to analyze other pairs of data streams before reaching the deadline $d$ (i.e., the arrival of the next micro-batch).

The second enhancement on *iBRAID* and *PriCe* is in the form of pruning of pairs of data streams and works as follow. Given a micro-batch, we know ahead of time the exact number of sliding windows within the micro-batch interval based on the number of tuples in that interval and the window range (i.e., this equals $I$ - $w$ + 1). We also know the maximum possible number of pairs of sliding windows that can be correlated in a pair of data streams. To give an extreme example, if a pair of data streams is perfectly correlated (i.e., identical data streams), then the number of correlated pairs of windows would be exactly: ($I$ - $w$ + 1). Thus, we keep analyzing the pair until we reach a timestamp where the number of remaining sliding windows to be analyzed is less than what that pair needs to correlate to meet the $A$ criterion. To formalize, we simply terminate the analysis process of a pair if the following condition holds:

$$(A - correlatedWindows) > (I - slidingWindowPosition)$$

where $A$ is the criterion of the number of correlated windows, *correlatedWindows* are the total number of windows that are correlated in a pair of streams according to PCC $\tau$. In other words, $I$ is the interval of the data streams, and *slidingWindowPosition* is the pair's analysis location in the interval.

## 3.4 Cold/Warm Start

When the very first micro-batch arrives at the system, the system has no knowledge about any correlated pairs of streams. However, this is not the case after the analysis of any micro-batch that produces a set of correlated pairs of data streams. This raises the question of how to exploit the results of past micro-batch analyses, for example, in picking the first pair in a new micro-batch to analyze. As opposed to *iBRAID*, this question has a major impact on *PriCe*, since its answer can be used in the initialization of the parameters of its priority function.

We call the case of a micro-batch analysis that starts with no knowledge of correlated pairs of streams, as *cold start* and as *warm start*, otherwise. In *cold start*, *PriCe*'s priority function is initialized to its default values (as discussed above).

In *warm start*, there are multiple ways to approach the priority function initialization. In this paper, the priority function is initialized based on the results of the latest micro-batch analysis. *Warm start* has two variations, *warm (High)* and *warm (Low)*. In *warm (High)*, we utilize the same parameters of the correlated pairs used by the immediately previous micro-batch. The rational behind this

variation is to keep analyzing closely those pairs that already exhibited the highest correlation in the previous micro-batch, potentially indicating critical problematic behavior.

In *warm (Low)*, we focus on the pairs that were not processed at all due to the lack of any correlated windows (i.e., not chosen for analysis due to their low correlation) at the beginning of *PriCe*'s execution. Specifically, we propose to jump start such pairs by altering their previous number of correlated windows (i.e., the parameter that reflects this information) to have the value $A$. This will increase their priority, preventing their starvation and giving them another chance to be analyzed in the new micro-batch. The rational behind this variation is to allow such pairs another chance, potentially identifying problematic behavior, which remained undetected in the previous micro-batch.

## 4 EXPERIMENTS AND ANALYSIS

In this section we present initial results from the evaluation of the impact of combining *DCS* with *iBRAID* and *PriCe* as well as of the cold and warm starts.

### 4.1 Experimental Framework

**Algorithms** In our evaluation, we used three baselines: a *Random* scheduler that picks pairs of data streams to explore randomly and with no preference, and our base algorithms iBRAID and *PriCe*. These algorithms were enhanced with the *DCS* mode: *Random-DCS*, *iBRAID-DCS*, and *PriCe-DCS*.

**Testbed** We implemented all the discussed algorithms and their variations in C++ 11. We ran the experiments on a computer with 2 Intel CPUs, running at 2.66GHz, and 96GB of RAM memory. The operating system used was CentOS 6.5 and the compiler was GCC version 4.8.2.

**Metrics** We evaluated the performance of the algorithms in terms of *execution cost* and *precision*.

*Execution Cost*: We measured the latency in *number of operations* performed to produce a result (i.e., number of correlated pairs of data streams). We used the number of operations as it provides the asymptotic efficiency of the algorithms compared to one another. This does not depend on factors such as the hardware characteristics and the operating system of the computer, which the experiments are run on, nor the efficiency of the compiler / virtual machine, which compiles and/or executes the code.

*Precision*: This is our optimization criterion (Eq. 3). We examined how our algorithms meet real-time deadlines and how many correlated pairs could they detect under such requirement.

**Dataset** *Yahoo Finance Historical Data* [4]: The dataset we have used in our experiments consists of 318 data streams. Those reflect the trading of 53 companies on the NYSE for the last 28 years. This gives us a total of 50403 different pairs to analyze. The data granularity is a day, which includes the price of the stock of the company at opening, the price at the end of the day (closing), the highest price for the day, the lowest price for the day, the amount of shares traded that day, and the adjusted close (calculated according to the standards of the CRSP, Center for Research in Security Prices). The length of each data stream is about 7100 tuples. Those tuples are divided into micro-batches.

**Table 1: Experimental Parameters**

| Parameter | Value(s) | | Parameter | Value(s) |
|---|---|---|---|---|
| PCC $\tau$ | [0.75, 0.90] | | $w$ | 8 |
| $A$ | [112, 225, 450] | | # data streams | 72 |
| $I$ | 900 (180 seconds) | | # *micro-batches* | 4 |

**Experiments** We ran three experiments to measure the execution cost, the precision of cold start and of warm start for two PCC threshold $\tau$'s, 75% and 90%, and for three different values of $A$, 112, 225 and 450. The values of $A$ correspond to the 1/8, 1/4 and 1/50 of the micro-batch interval. The micro-batch interval is set to 900 tuples to simulate an inter-arrival time of 180 seconds, where each tuple is produced each 200 milliseconds. Finally, we experimented with three deadlines corresponding to 25%, 50% and 75% of the total operations needed to determined all the correlated pairs in a micro-batch, i.e., achieve total precision. The experimental parameters are summarized in Table 1.

### 4.2 Experimental Results

In this section, we present the results of three experiments that we conducted to evaluate the performance and the ability of our algorithms to detect the correlated pairs in data streams in real-time.

**Experiment 1 (Figs. 1–3)** In our first experiment, we measured the execution cost or latency in *number of operations* of each algorithm to detect the specified number $A$ of correlated pairs of sliding windows in four consecutive mini-batch.

As expected, *Random*, *iBRAID*, and *PriCe* have the same number of operations due to exhaustive processing of the pairs. These do not use $A$ for either early termination nor pruning. Thus, with fixed number of data streams, intervals, and window range, the number of operations induced by the three algorithms is identical. They will always consume the same amount of operations regardless of the other parameters (i.e., PCC $\tau$ and $A$). The impact of *DCS* mode on all three algorithms is clearly visible in all figures.

In Figure 1, we notice that the higher the PCC $\tau$, the more operations are executed by the algorithms. This is due to the fewer number of windows that are highly correlated according to the PCC $\tau$. This results in more latency in capturing them by the algorithms. On the other hand, in case of lower PCC $\tau$, we see that *DCS* early terminates the analysis process as it reaches the $A$ criterion of number of correlated sliding windows. We also observe that *iBRAID-DCS* consistently underperformed the other *DCS* algorithms in latency. This is a consequence of the *iBRAID* scheduling scheme, where it processes all the pairs in a round-robin fashion to avoid starvation. This leads to having a pair pruned at a late stage of the analysis.

In Figure 2, we notice that *iBRAID-DCS* showed higher latency in the cases where PCC $\tau$ = 0.75. This is counter intuitive. To clearly state the reason, we observe that the lower the $A$ criterion, the later the pruning will occur in case of no high correlated windows were processed. With that in mind, we say that the pairs of data streams in Figure 2 in the case of PCC $\tau$ = 0.75 has produced high amount of correlated windows, enough to delay the pruning towards the end, but not enough to terminate the analysis early. Therefore, the performance of *iBRAID-DCS* was lower with low PCC $\tau$.

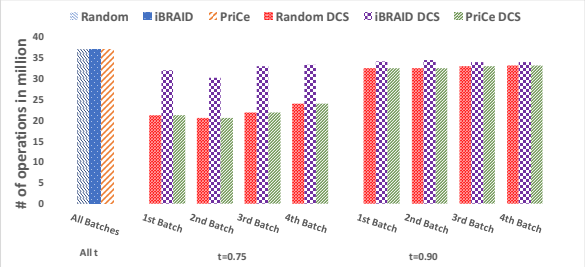


**Figure 1: The cost in number of operations for 4 consecutive micro-batches (the correlation criterion $A$ = 112).**

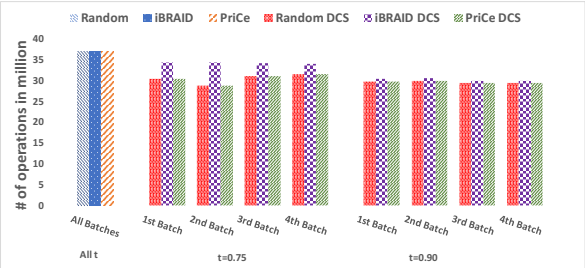


**Figure 2: The cost in number of operations for 4 consecutive micro-batches (the correlation criterion $A$ = 225).**

Our last claim is also supported by our experimental results in Figure 3, which shows clearly that with higher $A$ we were able to reach a better performance than lower $A$. The reason is that with $A$ = 450, if a pair encounters no correlated windows yet, it can be pruned by midway of the analysis process.

Finally, *DCS* mode of operation was able to enhance the performance of the algorithms up to 1.8 times (Figure 3).

**Experiment 2 (Figs. 4–13)** In this experiment, we studied the detection rate of each algorithm with respect to a given deadline. We set the deadline to be 25%, 50%, and 75% of the processing duration of each interval and measured the percentage of the number of correlated pairs each algorithm was able to detect. The results are shown for the deadlines 25% (Fig. 4–9), 50% (Fig. 10–Fig.11), and 75% (Fig. 12–Fig.13).

In general, we notice that *DCS* mode of operation in all cases for all algorithms outperforms the original algorithms. This is attributed to the pruning and early termination features of *DCS*, which allows the algorithms to analyze other pairs and detect more correlated data streams.

In Figures 4 and 5, we noticed that *iBRAID* and *iBRAID-DCS* showed low percentage of detected correlated pairs of data streams. This is due to the nature of *iBRAID* scheduling, while we see clearly that *Random-DCS* and *PriCe-DCS* have comparable performance. We attribute this to the fact that with low $A$ and low PCC $\tau$, we are expecting higher pairs to be correlated earlier. Thus, *Random-DCS* has shown better performance at two micro-batches (2nd and 3rd). We also note that the average detection percentage for *Random-DCS* with $A$=112 for all PCC $\tau$ at the deadline 25% is 56%.

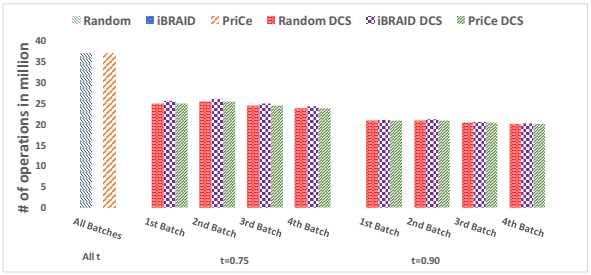


**Figure 3: The cost in number of operations for 4 consecutive micro-batches (the correlation criterion $A$ = 450).**

In Figures 6 and 7, we see that *PriCe* scheduling exhibits the effectiveness of its priority function in capturing more correlated pairs at an early deadline, more specifically when PCC $\tau$ is high. That means, it elect the pairs to explore more intelligently than the other two algorithms. Also, we notice that with high $A$ and early deadline, *iBRAID-DCS* fails to declare and detect any pair of data streams as a correlated one. We also note that the average detection percentage for *Random-DCS* with $A$=225 for all PCC $\tau$ at the deadline 25% is 48%.

In Figures 8 and 9, we see the same observation as in Figures 6 and 7, however, we realize that *PriCe-DCS* has mostly detected all the correlated pairs of data streams, and this is due to the scheduling nature of *PriCe* and the aggressive pruning at high $A$. The *Random-DCS* fails to meet that, since it picks pairs in an unpredictable way, and this delays the analysis duration for each pair, hence, delaying its pruning.

In Figures 10 and 11, we notice that *PriCe-DCS* has outperformed all the other algorithms, also, we see *iBRAID-DCS* detecting some pairs with. This for the obvious reason of having more processing time to advance the sliding windows and capture more correlated windows, which contributes to reaching the criterion $A$ and declare the pair as a correlated one. For $A$ = 112 and $A$ = 450, we noticed the same observation as in Figures 10 and 11.

Finally towards the end of the micro-batch analysis process, we see clearly in Figures 12 and 13 that all the algorithms are detecting pairs with an overall relatively higher percentage than the earlier deadlines, and this is expected with more time to analyze the pairs of data streams. We also observe the effect of pruning clearly on all algorithms under *DCS* mode. This is a result of the $A$ criterion being very high, which leads to early pruning for non promising pairs of data streams. Thus, all the correlated pairs of data streams were detected earlier than the 75% deadline. With $A$ = 112 and $A$ = 225, we noticed the same observations.

**Experiment 3 (Figs. 14–15)** Our previous two experiments show that *PriCe-DCS* exhibits the best performance overall. In our last experiment, we studied how cold, warm (Low), and warm (High) starts (see Section 3.4) affect the detection of correlated pairs of streams with respect to deadlines when using *PriCe-DCS*. As in the experiments above, we experimented with the values of PCC $\tau$, $A$ and deadline $d$, shown in Table 1. All experiments produced similar results; due to space limitation we report here only the first one.

In Figures 14 and 15, we show the percentage of detected correlated pairs at the 50% deadline. This experiment was conducted

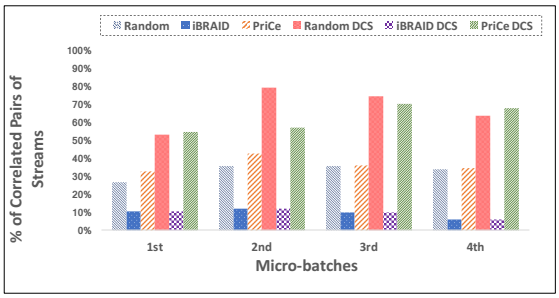


**Figure 4: The % of correlated pairs of streams detected by all algorithms at 25% of the interval $I$ (the correlation criterion $A$ = 112 and $t$ = 0.75).**

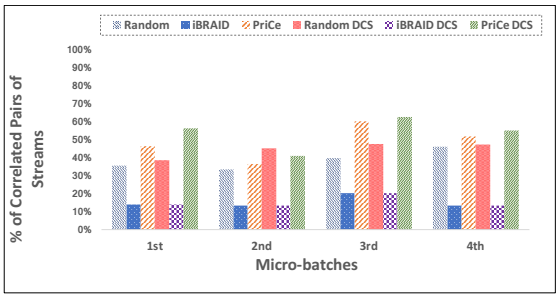


**Figure 5: The % of correlated pairs of streams detected by all algorithms at 25% of the interval $I$ (the correlation criterion $A$ = 112 and $t$ = 0.90).**

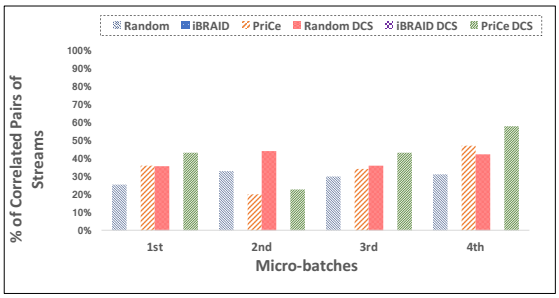


**Figure 6: The % of correlated pairs of streams detected by all algorithms at 25% of the interval $I$ (the correlation criterion $A$ = 225 and $t$ = 0.75).**

with $A$=112. We notice that *PriCe-DCS* in warm (high) start outperformed the rest of the starting techniques. This clearly indicates that using the exact history (i.e., from previous micro-batches intervals) of number of correlated windows for each pair of data streams is the most effective way to initialize the priority function when starting the analysis of each micro-batch. Trying to analyze the pairs that were stifled in previous micro-batches will result in degrading the detection of the correlated pairs in early stages.

**Take Away**: Our experimental evaluation showed that *PriCe-DCS* with warm start is the best policy for determining highly correlated live data streams.

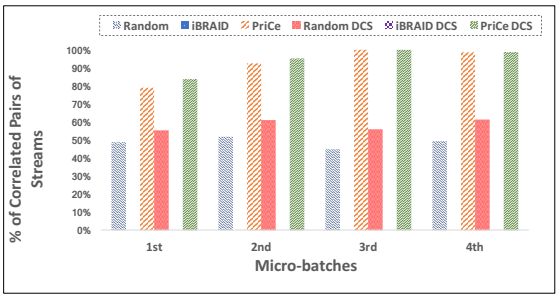


**Figure 7: The % of correlated pairs of streams detected by all algorithms at 25% of the interval $I$ (the correlation criterion $A$ = 225 and $t$ = 0.90).**

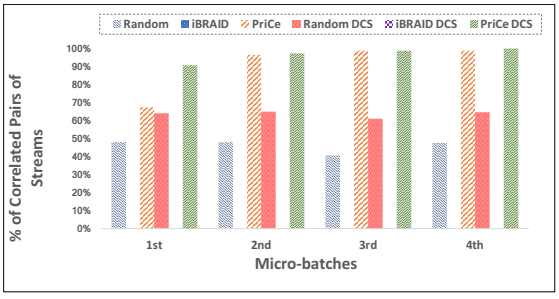


**Figure 8: The % of correlated pairs of streams detected by all algorithms at 25% of the interval $I$ (the correlation criterion $A$ = 450 and $t$ = 0.75).**

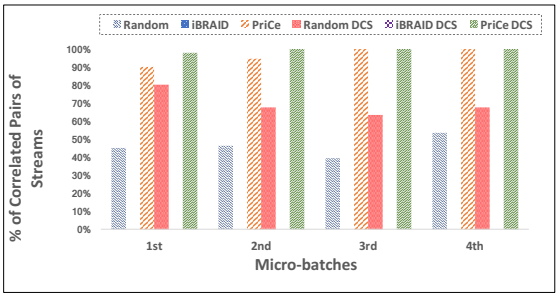


**Figure 9: The % of correlated pairs of streams detected by all algorithms at 25% of the interval $I$ (the correlation criterion $A$ = 450 and $t$ = 0.90).**

## 5 RELATED WORK

The processing of data and fast discovery of correlated subsequences of time series is tackled in two scenarios with respect to the production of data—dynamic, when the data is processed as it is produced [12] and static, when the data is collected upfront and it forms the search space for finding the correlated subsequences [5, 8, 11]. The latter is beyond the scope of our work. In this section we discuss state-of-the-art on computationally cheap identification of correlated data streams.

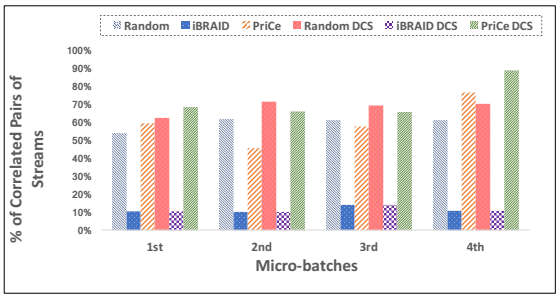


**Figure 10: The % of correlated pairs of streams detected by all algorithms at 50% of the interval $I$ (the correlation criterion $A$ = 225 and $t$ = 0.75).**

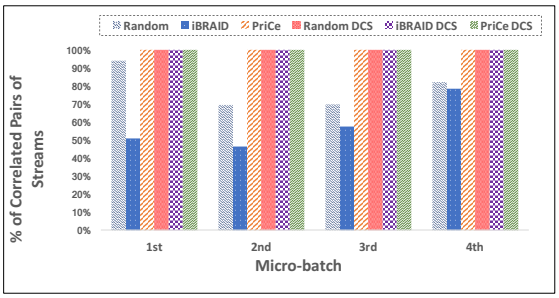


**Figure 13: The % of correlated pairs of streams detected by all algorithms at 75% of the interval $I$ (the correlation criterion $A$ = 450 and $t$ = 0.90).**

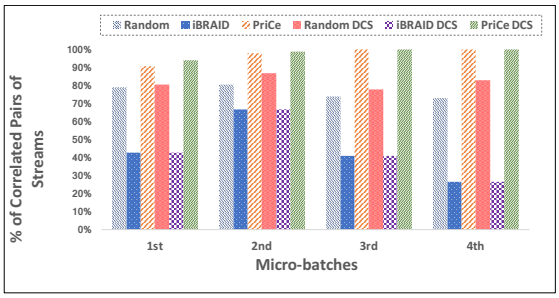


**Figure 11: The % of correlated pairs of streams detected by all algorithms at 50% of the interval $I$ (the correlation criterion $A$ = 225 and $t$ = 0.90).**

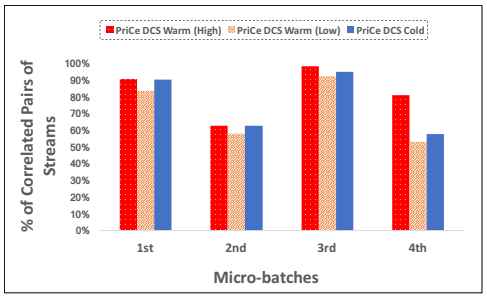


**Figure 14: The percentage of correlated pairs of streams detected by *PriCe-DCS* using the warm (high), warm (Low), and cold start at the deadline 50% start with $A$ = 112 and $t$ = 0.75**

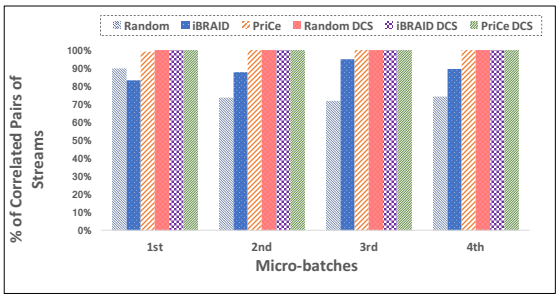


**Figure 12: The % of correlated pairs of streams detected by all algorithms at 75% of the interval $I$ (the correlation criterion $A$ = 450 and $t$ = 0.75).**

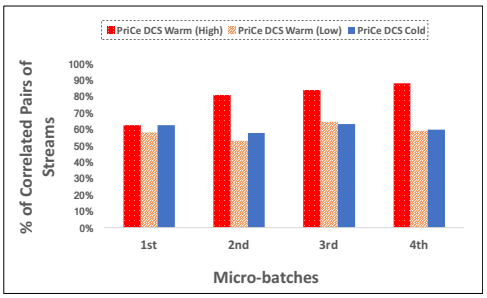


**Figure 15: The percentage of correlated pairs of streams detected by *PriCe-DCS* using the warm (high), warm (Low), and cold start at the deadline 50% start with $A$ = 112 and $t$ = 0.90**

In BRAID [11] the authors propose a technique to find correlated pairs of subsequences. Their work extends to the point that the two subsequences might not start at the same point in time—there might be a lag between them, i.e., the subsequences are of the same length, but one of them starts $l$ timestamps after the other one. The BRAID variants navigate the data space sequentially and preemptively. They calculate the PCC of each pair of subsequences with lag 0 as they navigate the space and use these values to average out (or "smoothen") the PCC for larger lags. The authors use powers of 2 in order to build a multilevel "smoothened" PCC as a geometric progression of the lag. Our work can be extended to support lagged pairs of subsequences at the cost of keeping in memory the sufficient statistics from the timestamps at which each of the subsequences starts.

Our work is in some respect related to interactive data exploration. In fact our *iBRAID* and *PriCe* algorithms were designed to support interactive data exploration. The authors of [5] propose an extension to SQL, which allows the definition of new types of queries. Those cannot be expressed easily with the traditional operators such as *GROUPBY* and *COUNT* – queries, which run arithmetic operators over ranges of data entries. They also propose a sampling-guided, data-driven search space navigation technique

for interactive data exploration. They build a grid over the search space and calculate a number of attributes for each cell of the space. In their example, they use the SDSS dataset, and the attributes they calculate for each cell are, for example, average brightness of the cell and the number of stars in the cell. These attributes are precomputed offline. They use a best-first heuristic and a priority queue to navigate the order of exploration of the cells. The algorithms we propose also prioritize the exploration of pairs of subsequences, which are more likely to produce results. Unlike [5], we use the smallest step possible for advancement of the subsequences - one timestamp. The granularity of the grid might impact significantly the search space navigation. We also do not precompute any data.

RainMon [12] proposes a 3-stage technique to mine bursty data streams. The received ticks are first decomposed, in order to obtain a smoothed representation of the data. In the next stage, called summarization, the received data goes through incremental principal component analysis in order to outline the long-term trends in the streams and to identifies anomalies, if there are any. In the last stage, named Prediction, the system forecasts trends, relying on the output from the summarization stage. In our work, we do not make predictions and our approach uses the data as it is delivered to identify correlated data streams.

A framework for identification of highly correlated pairs of data streams is also presented in StatStream [14]. One of the assumptions of the work is that only an approximation of the PCC is sufficient to identify the pairs of highly correlated DS. Based on this assumptions, the authors proposed a two fold approach to efficiently identify the pairs of interest. They employ a computationally cheap discrete Fourier transformation (DFT) technique to calculate an approximation of the PCC. Furthermore, they proposed an n-dimensional grid structure, which stores the DFT statistics and PCC approximations of each stream, whereby neighboring cells reflect highly correlated streams. This is the springboard for identification of the highly correlated pairs of DS. However, if the DS mimic white noise, DFT is known for its poor performance on DS, which mimic white noise, whereby the required number of DFT coefficients to precisely represent the DS is high and requires a lot of computations. Our work differs in the calculation of PCC. We also calculate the PCC incrementally over sliding windows, but our framework calculates it precisely for each pair, over each sliding window. Our studies showed that once a pair of DS is selected as being highly correlated due to a high value of the approximated PCC, a precise calculation of the PCC is required to prove the hypothesis. This operation requires two passes on the data. Similarly to StatStream, our framework supports sliding windows on data streams. We evaluate the possibilities to extend our framework to support landmark windows and damped windows in the future.

StatStream was further improved to handle "uncooperative" data streams in [1], but it still calculates an approximation of PCC only. The proposed technique employs structured random vectors. The experimental results show that the proposed technique outperforms linear scan and the discrete Fourier transformations, proposed in StatStream [14]. Our framework, like this work, updates the required statistics in fixed amount of time. Unlike it, *DCS* calculates PCC of the pairs of DS *precisely* for each sliding window and avoids the need to be calculated later and at a higher cost, once a pair is selected as being "promising".

# 6 CONCLUSIONS

In this paper we presented a number of priority-based search algorithms for real-time detection of correlated data streams. Our work aims to assist analysts in finding pairs of data streams, in which windows of tuples exhibit certain levels of correlation.

Our solution, called *DCS* for *Detection of Correlated Streams*, is a mode of operation that uses the Pearson Correlation Coefficient as a metric of correlation of two sliding windows of data streams and employs pruning to avoid examining pairs of data streams that cannot be part of the result. DCS utilizes early termination and distinguishes its execution into warm and cold start depending on whether or not the immediately previous micro-batch execution identified correlated pairs of streams. In warm start, *DCS* uses the results of the preceding micro-batch analysis as part of the initialization of the new micro-batch's analysis. Our real data experimental evaluation shows that our *DCS* mode of operation enhanced both *iBRAID* and *PriCe* algorithms. In particular, when *DCS* mode is used with *PriCe*, *PriCe-DCS* outperforms the other algorithms up to 1.8 times. Also, we studied the cold and warm start when analyzing a micro-batch based on the results of the analysis of the prior micro-batches. We found that *PriCe-DCS* with warm start outperformed the rest of the algorithms.

**Acknowledgments**. We would like to thank the anonymous referees for their helpful comments and suggestions for improving this paper.

# REFERENCES

[1] Richard Cole, Dennis Shasha, and Xiaojian Zhao. 2005. Fast Window Correlations over Uncooperative Time Series. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining (KDD '05)*. ACM, New York, NY, USA, 743–749. https://doi.org/10.1145/1081870.1081966

[2] Kaiyu Feng, Gao Cong, Sourav S. Bhowmick, Wen-Chih Peng, and Chunyan Miao. 2016. Towards Best Region Search for Data Exploration *(ACM SIGMOD'16)*. 1055–1070.

[3] Stratos Idreos, Olga Papaemmanouil, and Surajit Chaudhuri. 2015. Overview of Data Exploration Techniques *(ACM SIGMOD'15)*. 277–281.

[4] Yahoo Inc. 2016. Yahoo Finance Historical Data. (2016). https://finance.yahoo.com/quote/YHOO/history

[5] Alexander Kalinin, Ugur Cetintemel, and Stan Zdonik. 2014. Interactive Data Exploration Using Semantic Windows *(ACM SIGMOD'14)*. 505–516.

[6] Alexander Kalinin, Ugur Cetintemel, and Stan Zdonik. 2015. Searchlight: Enabling Integrated Search and Exploration over Large Multidimensional Data. *Proc. VLDB Endow.* 8, 10 (jun 2015), 1094–1105.

[7] Dongeun Lee, Alex Sim, Jaesik Choi, and Kesheng Wu. 2016. Novel Data Reduction Based on Statistical Similarity *(SSDBM '16)*. 21:1–21:12.

[8] Abdullah Mueen, Suman Nath, and Jie Liu. 2010. Fast Approximate Correlation for Massive Time-series Data *(ACM SIGMOD'10)*. 171–182.

[9] Mahsa Orang and Nematollaah Shiri. 2015. Improving Performance of Similarity Measures for Uncertain Time series Using Preprocessing Techniques *(SSDBM '15)*. 31:1–31:12.

[10] Daniel Petrov, Rakan Alseghayer, Mohamed Sharaf, Panos K. Chrysanthis, and Alexandros Labrinidis. 2017. Interactive Exploration of Correlated Time Series. In *Proceedings of the ExploreDB'17 (ExploreDB'17)*. ACM, New York, NY, USA, Article 2, 6 pages. https://doi.org/10.1145/3077331.3077335

[11] Yasushi Sakurai, Spiros Papadimitriou, and Christos Faloutsos. 2005. BRAID: Stream Mining Through Group Lag Correlations *(ACM SIGMOD'05)*. 599–610.

[12] Ilari Shafer, Kai Ren, Vishnu Naresh Boddeti, Yoshihisa Abe, Gregory R. Ganger, and Christos Faloutsos. 2012. RainMon: An Integrated Approach to Mining Bursty Timeseries Monitoring Data *(ACM KDD '12)*. 1158–1166.

[13] Eleni Tzirita Zacharatou, Farhan Tauheedz, Thomas Heinis, and Anastasia Ailamaki. 2015. RUBIK: Efficient Threshold Queries on Massive Time series *(SSDBM '15)*. 18:1–18:12.

[14] Yunyue Zhu and Dennis Shasha. 2002. StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time. In *Proceedings of the 28th International Conference on Very Large Data Bases (VLDB '02)*. VLDB Endowment, 358–369. http://dl.acm.org/citation.cfm?id=1287369.1287401

[15] Kostas Zoumpatianos, Stratos Idreos, and Themis Palpanas. 2014. Indexing for Interactive Exploration of Big Data series *(ACM SIGMOD'14)*. 1555–1566.