FilCorr: Filtered and Lagged Correlation on Streaming Time Series

Sheng Zhong
University of New Mexico, USA
zhongs@unm.edu

Vinicius M.A. Souza
University of New Mexico, USA
vinicius@unm.edu

Abdullah Mueen
University of New Mexico, USA
mueen@unm.edu

***Abstract*—An essential task on streaming time series data is to compute pairwise correlation across disparate signal sources to identify significant events. In many monitoring applications, such as geospatial monitoring, motion monitoring and critical infrastructure monitoring, correlation is observed at various frequency bands and temporal lags. In this paper, we consider computing filtered and lagged correlation on streaming time series data, which is challenging because the computation must be "in-sync" with the incoming stream for any detected events to be useful. We propose a technique to compute filtered and lagged correlation on streaming data efficiently by merging two individual operations: filtering and cross-correlations. We achieve *an order of magnitude* speed-up by maintaining frequency transforms over sliding windows. Our method is *exact*, *devoid of sensitive parameters*, and *easily parallelizable*. We demonstrate our technique in a seismic signal monitoring application.**

***Index Terms*—time series, correlation, filtering, streaming data**

I. Introduction

Large monitoring systems (e.g., a network of seismic stations) typically have hundreds of distributed sensors gathering and transmitting real-time data. An event (e.g., earthquake) in such a system creates dynamic responses at these sensors. The responses can be arbitrarily lagged because of the spatial separation of the sensors and be limited to a specific band of frequencies depending on the type of event. *We demonstrate an algorithm to correlate streaming data generated from distributed sensors in real-time in order to detect events.* A concrete application where our algorithm can be employed is seismic monitoring. In this application, when an earthquake happens, seismometers (i.e., seismic sensors) across the region of the earthquake observe the wave at varying times for varying duration, while the signals recorded at these sensors are often correlated. For better understanding, in Figure 1, we show a magnitude 7.8 event in Gurkha, Nepal, on April 25, 2015. Three waveforms recorded at three stations (marked red in the map) show a high correlation when the lag due to propagation delay is considered. The seismic wave generated in Nepal reaches Japan in about eight minutes and Australia in about eleven minutes. Besides, filtering is a mandatory operation in seismic signal processing to remove undesired noise from data and extract the right frequencies for desired events. We illustrate the importance of filtering in Figure 2. The raw waveforms rarely demonstrate a correlation between events. In contrast, waveforms filtered between 0.4Hz to 3Hz achieve a higher correlation.

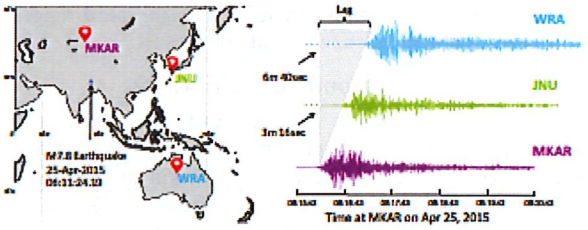

Fig. 1. *(left)* A M7.8 earthquake in Gurkha, Nepal, on Apr-25, 2015. *(right)* The signals recorded at three different stations at different times due to the spatial separation of sensors. Data collected from IRIS [1].

Computing lagged correlation (or asynchronous correlation) can be challenging because of fast data-rate, a large number of stations, and the necessity for accurate correlation values. Although the problem has been studied for decades, none of the existing methods such as BRAID [2], COLR-tree [3], or StatStream [4]), can monitor one hundred seismometers at 10Hz rate on a single workstation, a usual setting for many systems. The main reason for the failure of these methods is the data-dependent pruning, projection, or indexing technique. Seismic traces are mostly white noise (except when events happen) stressing the algorithms to fall behind the stream quickly. In contrast, none of these algorithms are amenable to data pre-processing requirements, such as filtering. Hence, pre-processing must always be done before correlation computation, resulting in a loss of efficiency.

In this paper, we propose an algorithm, **FilCorr**, to merge filtering, lagging and correlation computation in order to extract *data-independent* efficiency. Our algorithm efficiently maintains frequency information over the stream and calculates correlation in the frequency domain. The technique is *exact*, *devoid of sensitive parameters*, and *easily parallelizable*. We experimentally show that our technique is suitable for related monitoring applications where the lagged correlation of filtered time series is required, such as seismic monitoring. Our implementation can monitor one hundred seismic stations at a 10Hz rate without any delay. A demonstration of the method running on a seismic network is available on our website [5].

The main contributions of this work are summarized below:

- We demonstrate a working system to cross-correlate hundreds of streams in real-time at 10Hz speed using a conventional workstation;

2020 IEEE International Conference on Data Mining (ICDM) | 978-1-7281-8316-9/20/$31.00 ©2020 IEEE | DOI: 10.1109/ICDM50108.2020.00190

2374-8486/20/$31.00 ©2020 IEEE
DOI 10.1109/ICDM50108.2020.00190

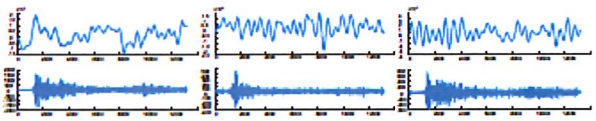

Fig. 2. The top row shows raw waveforms with no clear seismic signal. The bottom row shows filtered waveforms of the corresponding raw waveforms in the top row. Note the decrease in absolute value in the y-axis and the increased visibility of the signal. The filter band is 0.4Hz to 3Hz. The average lagged correlation of all three pairs of raw signals is zero. The average lagged correlation of all three pairs after filtering is 0.53.

- We merge digital filters and correlation computation in one combined step to achieve efficiency;
- We show a case study where such high speed lagged correlation help detecting events of interest.

II. Background and Notation

We define *time series* as a sequence of observations, and they are in the form of real numbers measuring a quantity at a fixed sampling rate. A *streaming time series* is an unbounded sequence of observations generated at a fixed rate. We use s^x to represent a particular time series x. We define the *basic window* as a continuous segment of a time series. A basic window of size m from time series s^x at timestamp t contains observations from $s^x[t-m+1]$ to $s^x[t]$, and it is denoted by w_t^x. We can extract at most $n-m+1$ basic windows of length m from a long time series of length $n >> m$. Adjacent basic windows are not independent of each other, overlapping basic windows are trivially close in the high dimensional space; however, they are not interesting for mining purposes.

Filtering: Filters are most commonly used to remove undesirable components from a time series. The number and types of filters vary enormously. All filters have response functions that are convolved with a signal to apply the filter. Such response functions contain relative weight for each frequency. In an analytical form, the weights are non-zero for all frequencies. However, practically, many weights are significantly smaller than the rest, allowing us to cut off and keep only the essential frequencies. Most applications employ a band that focuses on the need of the application. For example, seismic monitoring uses up to 10Hz [6], and EEG monitoring uses a gamma activity band between 30Hz-50Hz [7]. In this work, we assume that the given filter band is a contiguous set of frequencies, and all frequencies within the band are equally important; frequencies outside the band are zero. This is also known as a box filter. The algorithm is extendable to a non-contiguous set of frequencies with varying weights, i.e., other types of filters. We define frequency window W_t^x that contains all frequency components of the basic window w_t^x, and each $W_t^x[k]$ where $k = 0, 1, 2, \ldots, m-1$ are defined as follow. Note $i = \sqrt{-1}$.

$$W_t^x[k] = \sum_{j=0}^{m-1} w_t^x[j] e^{\frac{-i2\pi}{m}kj} \quad (1)$$

We use f in Hz to represent the sampling rate of the time series s^x and $w_t^{x\prime}$ to denote the filtered version of w_t^x. If we apply box filter with a band from f_s to f_t in Hz, then $w_t^{x\prime}$ can be derived from the following equation:

$$w_t^{x\prime}[j] = \frac{1}{m}\Big(\sum_{k=\frac{f_s}{f}m}^{\frac{f_t}{f}m} W_t^x[k] e^{\frac{i2\pi}{m}kj} + \sum_{k=m-\frac{f_t}{f}m}^{m-\frac{f_s}{f}m} W_t^x[k] e^{\frac{i2\pi}{m}kj}\Big); \quad (2)$$

The Fourier transform of $w_t^{x\prime}$ will only has non-zero components that corresponding to frequency from f_s to f_t in W_t^x. If $f_s = 0$ and $f_t = \frac{f}{2}$ (Nyquist frequency), then the inverse Fourier transform is identical to the basic window w_t^x.

Correlation computation in the frequency domain: If we are given two basic windows $w_{t_1}^x$ and $w_{t_2}^y$, the Pearson's correlation coefficient between them is defined as follows:

$$corr_{t_1 t_2}^{xy} = \frac{\sum_{j=0}^{m-1} w_{t_1}^x[j] w_{t_2}^y[j] - m\mu(w_{t_1}^x)\mu(w_{t_2}^y)}{m\sigma(w_{t_1}^x)\sigma(w_{t_2}^y)} \quad (3)$$

To compute the correlation in the frequency domain, we can exploit Parseval's theorem, which is expressed as:

$$\sum_{j=0}^{m-1} |w_t^x[j]|^2 = \frac{1}{m}\sum_{k=0}^{m-1} |W_t^x[k]|^2 \quad (4)$$

Now we demonstrate how each term in Equation 3 can be calculated in the frequency domain. $\mu(w_t^x)$ equals $\frac{W_t^x[0]}{m}$ since $W_t^x[0] = \sum w_t^x[j]$. $\sigma(w_t^x)$ equals $\sqrt{\frac{\sum w_t^x[j]^2}{m} - [\mu(w_t^x)]^2}$ in the time domain and the $\sum w_t^x[j]^2$ term can be computed by applying Equation 4 in the frequency domain. The following equation can be derived based on Parseval's theorem:

$$\sum_{j=0}^{m-1} |w_t^x[j] - w_t^y[j]|^2 = \frac{1}{m}\sum_{k=0}^{m-1} |W_t^x[k] - W_t^y[k]|^2 \quad (5)$$

Then we can extract $\sum_{j=0}^{m-1} w_t^x[j] w_t^y[j]$ and put it on the left-hand side of the equation, the right-hand side equals:

$$\frac{1}{2m}\Big(\sum_{k=0}^{m-1} |W_t^x[k]|^2 + \sum_{k=0}^{m-1} |W_t^y[k]|^2 - \sum_{k=0}^{m-1} |W_t^x[k] - W_t^y[k]|^2\Big) \quad (6)$$

Intuition: *Computing correlation values in the frequency domain is more efficient when we have a band-pass filter.* Based on Equation 2, the computation of correlation in the frequency domain is only $O(B)$, where B is the bandwidth.

Problem Statement: Given N streaming time series, a frequency band (f_s, f_t), and a maximum allowable lag l, compute Pearson's correlation coefficients for all pairs of time series over a sliding window up to the given lag.

We argue in this paper, with the empirical case study, this problem is very practical. In most domains, filtering can get rid of unwanted noise to compute correlation on right signals; and a reasonable maximum lag always exists, beyond which no correlation coefficient is meaningful.

III. Related Work

We categorize related work in four groups according to our proposal properties.

Lagged Correlation Computation: Computing lagged correlation, or cross-correlation, on offline data is a fundamental

operation that is benefited from Fast Fourier Transform. Researchers have proposed an online algorithm for lagged correlation computation named BRAID [2]. However, the method samples frequency coefficients in a logarithmic manner to approximate correlation value. Moreover, the method computes correlations over the entire history of the stream, unlike our method that computes over a sliding window in real-time. One may consider offline indexing methods such as iSAX [8] and COLR-tree [3], for lagged correlation computation. However, such offline indexes suffer from many modifications (insert or delete) operations over correlation computation.

Real-time Correlation Computation: Efficient all-pair correlation computation for streaming data is no longer an active research problem. StatStream [4] is a technique that exploits a few Fourier coefficients to prune improbable pairs quickly. ParCorr [9] performs random projections to similarly prune improbable pairs without redundancy due to the sliding window. Hardware-based techniques are often used for computation at MHz to GHz rate [10]. None of these methods consider lagged correlation after filtering.

Frequency Domain Correlation Computation: It is very common in many application areas to calculate correlations in frequency domain mostly enabled by the offline nature of computation [6], [11].

Filtered Correlation Computation: In this category, our method is unique. To the best of our knowledge, no work exploits filtering operation to extract efficiency in correlation computation. It is somewhat surprising considering the widespread usage of filtering in processing real-time data captures. The novelty in our technique is that the speedup is not data-dependent, unlike the aforementioned works.

In Table I, we present a comparison of the main capabilities of FilCorr concerning state-of-the-art algorithms for cross-correlation computation over multiple time series. Some capabilities are not demonstrated but trivially achievable with simple extensions of the algorithms. FilCorr comprehensively covers capabilities across several existing works, making it unique in the suite of correlation computing algorithms.

TABLE I
CAPABILITIES OF FILCORR AND RELATED WORK. ✓ REPRESENTS A CLAIMED CAPABILITY. – REPRESENTS EXTENDABLE CAPABILITY AND × REPRESENTS UNKNOWN.

	FilCorr	ParCorr	BRAID	COLR-Tree	StatStream
Data-independent	✓	×	✓	×	×
Filtering	✓	–	×	×	–
Lagged corr.	✓	×	✓	×	×
Real-time	✓	✓	–	✓	✓
Exact	✓	✓	×	–	✓
Parallel	–	✓	–	–	–

IV. FILCORR: FILTERED LAGGED CORRELATION

For simplicity but without loss of generality, we assume all streams have the same sampling rate f, no discontinuity (no data loss during the transmission) and the observations

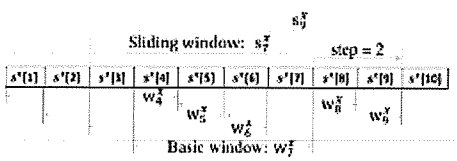

Fig. 3. Various windows on a stream, where $m = 4, l = 3$, and $step = 2$

are all aligned, which means they all have timestamps in set $\{1, ..., t-2, t-1, t, ...\}$

The primary motivation of FilCorr is the need for lagged correlation computation of filtered signals by systems processing streaming data from distributed sensors, such as seismic monitoring. These systems require monitoring hundreds of sensors responsible for generating data at high speed.

A. Lagged Correlation

We use $lcorr_t^{xy}$ to denote the lagged correlation value at timestamp t between two streams s^x and s^y, it is defined as:

$$lcorr_t^{xy} = Max(corr_{tt_t}^{xy}, corr_{t_t t}^{xy}); t_t \in [t-l, t] \quad (7)$$

$lcorr_t^{xy}$ is the largest Pearson's correlation coefficient value among the correlation between the most recent basic window at time t and all the previous basic windows with a timestamp from $t-l$ to t. Such a strategy can cover all possible cases when an event captured by different sensors at different time. **Case 1:** No lag. The event appears in both streams at the same timestamp. **Case 2:** The event appears in s^y first then in s^x at timestamp t, such scenario will be captured by computing correlation between w_t^x and $w_{t_t}^y$. **Case 3:** The event appears in s^x first then in s^y at timestamp t. To combine all cases, we define *sliding windows* s_t^x and s_t^y. Each sliding window will cover all $(l+1)$ basic windows that necessary to compute $lcorr_t^{xy}$. The relations are shown in Figure 3.

B. Lagged Correlation with Filtering

In this section, we demonstrate two algorithms that compute lagged correlation with filtered data. We call one algorithm as naive, which does computation in time domain. Another one is our proposed algorithm FilCorr, which computes the correlation in the frequency domain. When the most recent observations $s^x[t]$, $s^y[t]$ are generated we input two sliding windows s_t^x and s_t^y to both algorithms. Then they will output $lcorr_t^{xy}$. We apply the ideal bandpass filter which has frequency response equals to 1 for frequencies within the filter band, and 0 for frequencies outside the band. For any other known frequency response, we can apply FilCorr trivially.

Naive approach: The procedures are shown in Algorithm 1. Some intermediate values $w_t^x t$, $w_t^y t$ and μ, σ will only be computed once and will be saved in the memory for sharing with other pair computation and also for future use.

Proposed approach: As shown in Algorithm 2, we propose to combine filtering and correlation computation in one step, eliminating the need to perform Fourier transform and the inverse on every new basic window and decreasing the number of observations that need to be computed. Our approach

Algorithm 1: NAIVE

```
   Function LagFilterCorr(s_t^x, s_t^y, l, m)
1  |  w_t^x ← filter(w_t^x)
2  |  w_t^y ← filter(w_t^y)
3  |  curMax ← oneCorr(w_t^x, w_t^y, m)  //case 1
4  |  for i_t = t − l : t − 1 do
5  |  |  tmp1 ← oneCorr(w_{i_t}^x, w_t^y, m)  //case 2
6  |  |  tmp2 ← oneCorr(w_{i_t}^x, w_t^y, m)  //case 3
7  |  |  curMax = max(curMax, tmp1, tmp2)
   |  end
8  |  return curMax
   end
   Function filter(w_t^x)
9  |  W_t^x ← FFT(w_t^x)
10 |  W_t^x[0 : ⌊m f_s/f⌋ − 1] ← 0
11 |  W_t^x[⌊m f_t/f⌋ + 1 : m − ⌊m f_t/f⌋ − 1] ← 0
12 |  W_t^x[m − ⌊m f_s/f⌋ + 1 : m − 1] ← 0
13 |  return IFFT(W_t^x)
   end
   Function oneCorr(w_{i_1}^x, w_{i_2}^y, m)
14 |  return [Σ_j w_{i_1}^x[j] w_{i_2}^y[j] − mμ(w_{i_1}^x)μ(w_{i_2}^y)] / mσ(w_{i_1}^x)σ(w_{i_2}^y)
   end
```

Algorithm 2: FILCORR

```
   Function LagFilterCorr(s_t^x, s_t^y, l, m)
1  |  LB ← m⌊f_s/f⌋
2  |  UB ← m⌊f_t/f⌋
3  |  W_t^x|_{f_s}^{f_t} ← filter(W_{t−1}^x|_{f_s}^{f_t}, m, LB, UB, w_{t−1}^x[0], w_t^x[m − 1])
4  |  W_t^y|_{f_s}^{f_t} ← filter(W_{t−1}^y|_{f_s}^{f_t}, m, LB, UB, w_{t−1}^y[0], w_t^y[m − 1])
5  |  curMax ← oneCorr(W_t^x|_{f_s}^{f_t}, W_t^y|_{f_s}^{f_t}, m)  //case 1
   |  for i_t = t − l : t − 1 do
6  |  |  tmp1 ← oneCorr(W_t^x|_{f_s}^{f_t}, W_{i_t}^y|_{f_s}^{f_t}, m)  //case 2
7  |  |  tmp2 ← oneCorr(W_{i_t}^x|_{f_s}^{f_t}, W_t^y|_{f_s}^{f_t}, m)  //case 3
8  |  |  curMax = max(curMax, tmp1, tmp2)
   |  end
9  |  return curMax
   end
   Function filter(W_t^x|_{f_s}^{f_t}, m, LB, UB, d, a)
10 |  for q from 0 : UB − LB do
11 |  |  k ← q + LB  //k is index in W_t^x
12 |  |  W_{t+1}^x|_{f_s}^{f_t}[q] = e^{i2πk/m} (W_t^x|_{f_s}^{f_t}[q] − d·e^{−i2πdk/m} + a·e^{−i2πmk/m})
   |  end
13 |  return W_{t+1}^x|_{f_s}^{f_t}
   end
   Function oneCorr(W_{i_1}^x, W_{i_2}^y, m)
   |  /*Assume DC component is filtered */
14 |  return (Σ_q |W_{i_1}^x[q]|^2 + Σ_q |W_{i_2}^y[q]|^2 − Σ_q |W_{i_1}^x[q] − W_{i_2}^y[q]|^2) / (2√(Σ_q |W_{i_1}^x[q]|^2 Σ_q |W_{i_2}^y[q]|^2))
   end
```

maintains frequency coefficients of the most recent basic window by incrementally updating from the previous frequency window. We use $W_t^x|_{f_s}^{f_t}$ to represent the frequency window that only contains Fourier coefficients corresponding to frequencies within the filter band from f_s to f_t.

Line 14 of Algorithm 2 is based on equations 2, 3, 4, and 6. Note that the correlation coefficients from both algorithms are *exactly* the same. The exactness of our algorithm is directly derived from Parseval's theorem described earlier.

FilCorr can output all-pair correlations upon receiving the next set of observations in the streams. However, the output rate does not necessarily have to be the same as input. If the current sliding window is s_t^x, we can slide to s_{t+step}^x, where the *step* is the number of observations in a stream the algorithm gathers before outputting the next set of pairwise correlations. When $step = 1$, the algorithm outputs at the same rate as the the input. The larger the *step*, the slower the output rate.

C. Computational Complexity

The time complexity of computing filtered and lagged correlation at any given time is composed of two parts: i) filtering and ii) correlation computation. For the filtering, FilCorr only needs $O(B)$ based on the *filter* function. B is the number of elements within the filter band, which is $1 + \frac{f_t - f_s}{f} m$. So the worst case is $O(m)$ for FilCorr. However, the naive algorithm will take $O(m \log m)$ to finish based on the *filter* function from Algorithm 1 with FFT applied.

As for the cost of computing lagged correlation coefficients, the for loop in both algorithms will be executed l times. Each iteration of the naive algorithm will take $O(m)$ based on the equation in line 14, while the FilCorr only takes $O(B)$ time. Since the correlation computation has to wait until the filtering finish, the final time complexity for naive is $O(m \log m + lm)$, and $O(B + lB)$ for FilCorr, $O(m + lm)$ in the worst case. In practice, the speedup is more because the number of possible lags is much less than the window size, and the frequency band is much smaller than the number of observations in the signal. Finally, the total cost to run the whole algorithm has another factor of $O(N^2)$ which is the number of pairs that N streaming time series can generate.

The naive algorithm's space complexity is $O(lm)$ to maintain the filtered windows and all the statistics for one pair of streams. The space complexity of FilCorr is $O(lB)$.

V. EXPERIMENTAL EVALUATION

All our experiments are reproducible. The code, data, and additional results are available in our supporting website [5].

A. Setup

All experiments are performed on a single desktop computer. As FilCorr is data-independent, we use synthetic data for various experiments. The performance on real-world data will be discussed in Section VI.

We evaluate the performance under two scenarios: *offline* and *online*. For the offline scenario, all data are available beforehand, and the benchmark is the total execution time including I/O. For the online scenario, the observations are fed into the system as a stream at a specific rate, and we measure the maximum number of streams that the machine can handle without causing *any* delay at *any* correlation computation.

B. Efficiency

In Figure 4, we show the execution time of FilCorr and naive in the offline scenario. The Execution time of FilCorr grows at a much slower rate compared to that of the naive algorithm (see Figure 4). In Figure 5, we show the maximum number of pairs the system can handle. In all experimental settings, FilCorr can process (up to 4×) more sensors than naive. The performance gap increases with higher input or output rate. For other filter bands, the general performance trends hold.

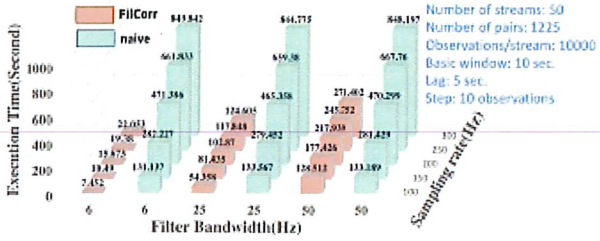

Fig. 4. Offline performance of FilCorr and naive algorithm.

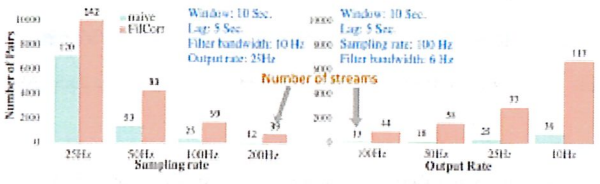

Fig. 5. Online performance of FilCorr and naive algorithm.

C. Comparison to State-of-the-art

Based on Table I, we argue that FilCorr is a comprehensive method for streaming correlation. However, although not an ideal match in capabilities, we identify ParCorr [9] as the most recent baseline with state-of-the-art performance. ParCorr calculates pairwise correlation in parallel based on randomly projected sketches. However, it is important to note that ParCorr does not compute the lagged correlation.

In order to favor ParCorr's implementation, all the experiments are performed in offline mode. Since ParCorr is data-dependent, we use two sets of synthetic data with 2,000 observations in each targeted to emulate the best-case and worst-case scenarios for ParCorr. The cost of ParCorr depends on the number of pairs it can prune without computing the correlation coefficients. Our first synthetic dataset contains sequences of uniformly distributed random numbers, where is expected to contain only uncorrelated pairs. Thus, a random noise dataset is the best data for ParCorr, which can prune all possible pairs. To further boost ParCorr's performance, we use a high correlation threshold, *candThreshold*, for better pruning, and subtract the time to initiate Apache Spark system to offset the overhead cost. On the contrary, the second dataset is a sinusoid that is expected to have all possible pairs of streams to be highly correlated. In this case, ParCorr computes correlation for all possible pairs, failing to prune and demonstrating the worst-case performance.

We show the performance comparison in Figure 6. The light grey shaded area represents the range of performance by ParCorr. The worst-case performance (on sinusoid data) is illustrated by the superior grey line with solid circles, the best-case performance (on random data) by ParCorr is illustrated by the inferior grey line with solid boxes. The actual performance of ParCorr on any other dataset should be in between the worst- and best-case lines. In Figure 6(zoom-out), we see that the time spent by FilCorr for various lags is well inside the

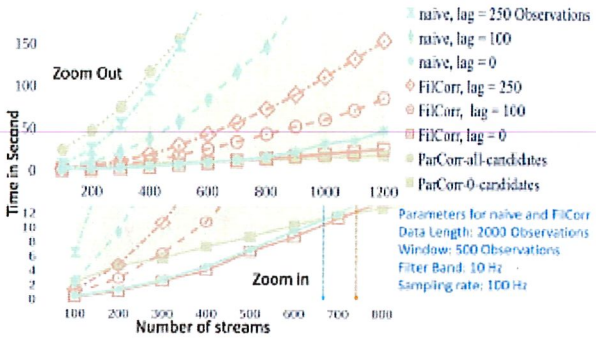

Fig. 6. Comparison with ParCorr fixing *step* = 20. The vertical red line shows the crossing point between ParCorr and FilCorr with lag = 0, and the vertical blue line shows the crossing point between ParCorr and naive algorithm with lag = 0.

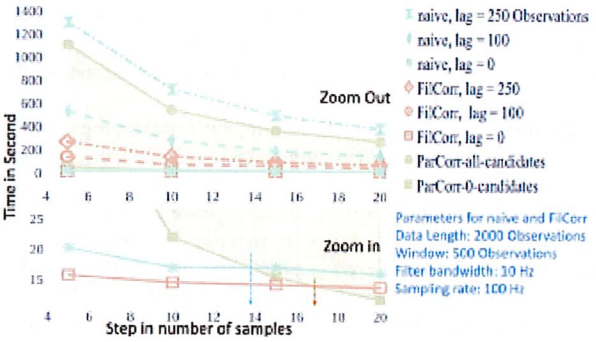

Fig. 7. Comparison with ParCorr varying the *step* from 5 to 20 observations. The vertical red line shows the crossing point between ParCorr and FilCorr with lag = 0; the arrow points the point where the output rate is 6Hz. The vertical blue line shows the crossing point between ParCorr and naive algorithm with lag = 0, and the arrow points to the output rate as 7Hz.

shaded area.

To be fair to ParCorr, when we consider the synchronous correlation (lag = 0), FilCorr is more efficient than the best-case of ParCorr up to around 700 streams as shown in Figure 6(zoom-in). Therefore, we recommend using FilCorr on a single desktop when the number of streams is less than 700, instead of using a parallel system.

In the second experiment, we fix the total number of streams at 800 and vary *step* from 5 to 20 observations, which correspond to the output rate of 20Hz to 5Hz. The results are shown in Figure 7. The execution time for all methods increases when the *step* is getting smaller to compute more correlation coefficients for a higher output rate. However, ParCorr's time increases at a higher rate compared to FilCorr. The zoom-in figure shows that FilCorr with $lag = 0$ has better performance than the best-case of ParCorr when the output rate is higher than the 6Hz.

VI. Seismic Event Monitoring

We have deployed a system for monitoring a seismic network with 29 stations in Yellowstone, Wyoming, through

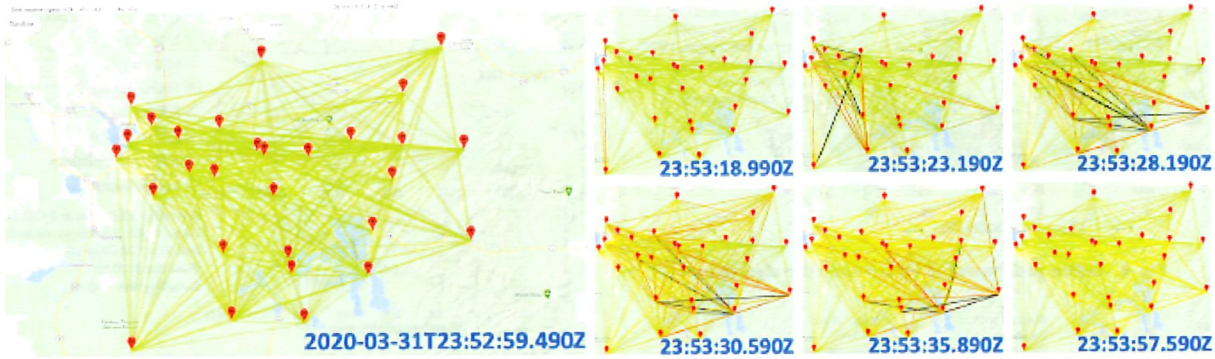

Fig. 8. Pairwise correlation among 29 stations at Yellowstone, WY, over different times given an M6.5 earthquake occurred in Challis, ID, at 23:52:30 2020-03-31 (UTC).

the use of FilCorr. The system ingests real-time data from IRIS (Incorporated Research Institutions for Seismology) [1] and calculates correlation coefficients between all stations (628 pairs) at a 100Hz rate.

Correlation coefficients can capture earthquake propagation through a region in real-time, which can easily be converted to a detector with the rule: *"If more than Q% of pairs of stations are highly correlated (> 0.9), an earthquake is propagating."* The utility of such a detector is massive for early warning systems because seismic wave propagates at 8 km/s, which is much slower than electronic signals carrying the warning.

As we previously showed in Figure 2, filtering is essential for seismic data to remove noise and obtain a frequency domain representation that better describes the signals. We consider a 20 seconds window size, a box bandpass filter with a cutoff frequency of 3Hz and 7Hz, and a lag of 10 seconds.

In Figure 8, we illustrate the propagation of a seismic event (id: us70008jr5), which our system observed about 300 miles away in the stations at Yellowstone. In this figure, each station is represented by a red symbol; a colored line represents the correlation value of a pair of stations. The correlation values from 0 to 0.5 are mapped from green to yellow, while correlations from 0.5 to 0.9 are mapped from yellow to red edges. A black edge represents a correlation value greater than 0.9. We notice that the correlated edges are appearing between station pairs as the earthquake wave reaches them. Similarly, edges become uncorrelated when the wave has passed through.

VII. Conclusion

We propose an algorithm to compute filtered and lagged correlation over streaming time series. FilCorr combines filtering and cross-correlation operations to compute the lagged correlation between streaming time series efficiently. FilCorr is faster than the state-of-the-art algorithm that computes correlation in parallel. We show a case study where FilCorr achieves promising results towards greater societal impact.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant #OIA-1757207.

References

[1] "The facilities of iris data services, and specifically the iris data management center, were used for access to waveforms, related metadata, and/or derived products used in this study." 2018. [Online]. Available: https://www.iris.edu/hq/

[2] Y. Sakurai, S. Papadimitriou, and C. Faloutsos, "Braid: Stream mining through group lag correlations," in *SIGMOD*, 2005, p. 610.

[3] Y. Ahmad and S. Nath, "Colr-tree: Communication-efficient spatio-temporal indexing for a sensor data web portal," in *ICDE*, 2008, pp. 784–793.

[4] Y. Zhu and D. Shasha, "StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time," in *VLDB*, vol. 54, no. 2, 2002, pp. 358–369.

[5] A. M. Sheng Zhong, Vinicius Souza, "Supporting website," https://sites.google.com/view/filcorr/.

[6] N. S. Senobari, G. J. Funning, E. Keogh, Y. Zhu, C. M. Yeh, Z. Zimmerman, and A. Mueen, "Super-Efficient Cross-Correlation (SEC-C): A Fast Matched Filtering Code Suitable for Desktop Computers," *Seismological Research Letters*, vol. 90, no. 1, pp. 322–334, 2019.

[7] J. F. Cavanagh, P. Kumar, A. A. Mueller, S. P. Richardson, and A. Mueen, "Diminished eeg habituation to novel events effectively classifies parkinson's patients," *Clinical Neurophysiology*, vol. 129, no. 2, pp. 409–418, 2018.

[8] J. Shieh and E. Keogh, "iSAX : Indexing and Mining Terabyte Sized Time Series," in *SIGKDD*, 2008, pp. 623–631.

[9] D. E. Yagoubi, R. Akbarinia, B. Kolev, O. Levchenko, F. Masseglia, P. Valduriez, and D. Shasha, "ParCorr: efficient parallel methods to identify similar time series pairs across sliding windows," *Data Mining and Knowledge Discovery*, vol. 32, no. 5, pp. 1481–1507, 9 2018.

[10] J. H. *et al.*, "A digital correlator upgrade for the Arcminute MicroKelvin Imager," *Monthly Notices of the Royal Astronomical Society*, vol. 475, no. 4, pp. 5677–5687, 2018.

[11] A. Mueen, S. Nath, and J. Liu, "Fast approximate correlation for massive time-series data," in *SIGMOD*, 2010, pp. 171–182.