Automated Parallel Data Processing Engine with Application to Large-scale Feature Extraction

Xin Xing*, Bin Dong†, Jonathan Ajo-Franklin†, Kesheng Wu†,
*Georgia Institute of Technology, USA. Email: xxing33@gatech.edu
†Lawrence Berkeley National Laboratory, USA. Email: {dbin, JBAjo-Franklin, kwu}@lbl.gov

***Abstract*—As new scientific instruments generate ever more data, we need to parallelize advanced data analysis algorithms such as machine learning to harness the available computing power. The success of commercial Big Data systems demonstrated that it is possible to automatically parallelize many algorithms. However, these Big Data tools have trouble handling the complex analysis operations from scientific applications. To overcome this difficulty, we have started to build an automated parallel data processing engine for science, known as ArrayUDF. This paper provides an overview of this data processing engine, and a use case involving a feature extraction task from a large-scale seismic recording technology, called distributed acoustic sensing (DAS). The key challenge associated with DAS data sets is that they are vast in volume and noisy in data quality. The existing methods used by the DAS team for extracting useful signals like traveling seismic waves are complex and very time-consuming. Our parallel data processing engine reduces the job execution time from 10s of hours to 10s of seconds, and achieves 95% parallelization efficiency. ArrayUDF could be used to implement more advanced data processing algorithms including machine learning, and could work with many more applications.**

***Keywords*-ArrayUDF, distributed acoustic sensing, local similarity**

I. Introduction

Scientific experiments and simulations generate enormous amount of data that requires machine learning and other advanced data analyses to extract useful information (e.g., features or patterns). Extracting features from massive data sets requires the computing power of a large number of computers working in parallel. However, developing parallel data analysis programs is a very labor intensive process. In the recent decades, the Big Data systems including Google's MapReduce and Apache Spark have demonstrated a data parallel approach to automate the complex data analysis operations. This approach employs a simple data model to capture a large variety of application needs and enables application programmers to concentrate on the operations to be performed on a small unit of data, while leaving the common tasks of data management, parallelization, and error-recovery in the data processing engine to be developed by a relatively small group of experts. Though this approach has been wildly successful in commercial applications, its impact on scientific applications has been limited because of mismatching data model and the lack of support for complex analysis operations. In this paper, we provide an overview of the automated parallel data processing engine we are developing, and demonstrate its capability with a large-scale seismic monitoring application with distributed acoustic sensing (DAS).

DAS [13][12] is a new seismic recording technology that can record vibration amplitudes of every small segment of a telecommunication fiber-optic cable. With such dense recordings along long cables, the size of signal data from DAS system can easily reach to petabytes, even exabytes. These enormous amounts of DAS data are useful in many applications such as earthquake detection [11], seismic monitoring of the near surface [7], and permafrost condition change [2]. One of the key signal in DAS data is the traveling waves induced by seismic events like moving vehicles and earthquakes. Our work will concentrate on extract such a signal, which should pave the way for more advanced data analysis such as machine learning.

Several types of methods have been developed to locate traveling waves based on traditional seismic sensors and can be applied to DAS data as well. The first type is usually called "energy detector" and is based on capturing the rapid amplitude or energy increases in the recorded signals which are mostly induced by traveling waves. Related methods include short-term average over long-term average (STA/LTA) [8] and many of its variants. Secondly, noting that a traveling wave can induce similar recorded signals in independent seismic sensors, high waveform similarities of signals from independent sensors can also be used to locate traveling waves. Methods of this type [9][10][18] often use cross-correlations to quantify the waveform similarity but have different post processings of the calculated cross-correlations. Lastly, there are also attempts to use convolutional neural networks (CNN) [14] particularly for earthquake detections. However, CNN mostly requires labeled data to guide its model training while in many cases, there is no label in DAS data for locating general traveling waves. These state-of-art algorithms are very time-consuming, especially for the large DAS data. For example, a small sample (1 minute) of DAS recording is projected to take tens of hours with the existing matlab procedures used by the application scientists. Therefore, it is highly necessary to parallelize the analysis procedures to harness the power of many computers.

To simplify the development of parallel data analysis programs, the Big Data systems such as MapReduce [5] include a new generation of parallel data analysis engines. With a MapReduce system, users first define operations on an atomic data structure, i.e., *key-value* (KV) *pair*, and then pass these operations to either *Map* operator or *Reduce* operator to compose complex analysis procedures. Many higher level scalable data analysis systems such as Spark [17] and HBase [4] are then further built based on MapReduce. The basic KV pair structure of these MapReduce-based systems, however, does not match the multidimensional array structure used in DAS and other common scientific applications [15]. Meanwhile, Chaimov et al. [3] also shows that these MapReduce-based systems can not scale well on supercomputers. More recently, TensorFlow [1] uses array as basic data structure but it only supports customized operators. Using its built-in operators to define new operations may require code development, parallelization, performance tuning, and other challenging tasks.

To overcome these shortcomings, we develop ARRAYUDF. We demonstrate the effectiveness of ARRAYUDF with a use case of locating traveling waves in DAS data. More specifically, our work has following contributions:

- We introduce a newly-developed data processing engine ARRAYUDF that can automatically parallelize data processing methods that apply a fixed compute kernel to different local sub-blocks of multi-dimensional arrays. The ARRAYUDF only requires users to provide a few parallelization parameters and a compute kernel. It can automatically manage data partitioning and parallel execution of the compute kernel on a supercomputer with tens of thousands of computing nodes.
- We develop a DAS data analysis method based on the concept of local similarity [10] recently developed for conventional sensors. This method can effectively locate traveling waves among noisy DAS data and also turns out to have scalable performance on supercomputers.
- We experimentally show the effectiveness of our local-similarity based method and the scalability of ARRAYUDF on a supercomputer. Test results show that ARRAYUDF can achieve around 95% parallel efficiency in our data analysis calculation. More importantly, it reduces the compute time from tens of hours to tens of seconds.

Organization of this paper is as follows. Section II gives an overview of ARRAYUDF. In Section III, we introduce our target application DAS and its data analysis challenges. In Section IV, we present the local-similarity based method for traveling wave detection in DAS data and our approach to parallelize the local similarity calculation with the help of ARRAYUDF. Section V reports the experiment results. We conclude our paper with future works in Section VI.

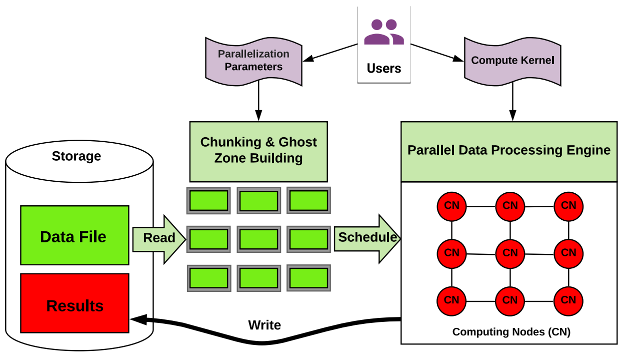

Parallelization Parameters
Users
Compute Kernel
Chunking & Ghost Zone Building
Parallel Data Processing Engine
Storage
Data File
Results
Read
Schedule
Write
CN
Computing Nodes (CN)

Figure 1: High-level overview of ARRAYUDF for automated parallel data processing.

II. AUTOMATED PARALLELIZATION FOR DATA PROCESSING

The fundamental idea behind the current Big Data systems such as MapReduce and Apache Spark is to define a data model that allows users to define a variety of analysis operations on a basic unit of data. The basic unit of data in a MapReduce system is a key-value pair, while most of the scientific data is organized as arrays. To directly support operations on array elements, we introduce a new data processing engine ARRAYUDF [6].With ARRAYUDF, the processing power of a supercomputer can be easily harnessed to accelerate many scientific applications.

A high-level overview of ARRAYUDF is presented in Figure 1. ARRAYUDF only requires a compute kernel and a few parallelization parameters from users. The compute kernel describes a universal data processing operation applied to every local sub-block centered at different entries of the original array. The parallelization parameters include chunk size and ghost zone size, which are used to split the original array for parallel processing. After splitting the array, all the chunks with their ghost zones are read and scheduled onto available computing nodes using round-robin method. Each computing node then executes the given compute kernel to all the local sub-blocks in its assigned chunks.

Ghost zone is a classic technique in parallel computing to keep the computation local at each node and avoid expensive communications. Specifically, in ARRAYUDF, applying the compute kernel to a sub-block centered at an entry near chunk boundary usually needs to access entries from a neighboring chunk. If without ghost zone, this calculation requires expensive and also complicated node-to-node communications during the execution of the compute kernel.

ARRAYUDF utilizes a generic user-defined function interface for users to define the compute kernel. The user-defined function interface is commonly used by database systems and MapReduce to allow diverse operations. The parallel execution of this user-defined function to different local sub-blocks is then automatically parallelized and completed by

ARRAYUDF. More specifically, ARRAYUDF provides the following two major C++ classes for easy-to-use:

- The ARRAY class provides an in-memory pointer and an abstraction for any multi-dimensional array stored on disks. Its initialization requires a storage path of the array and also chunk and ghost zone sizes to configure the splitting of the array. The APPLY function is the main method of ARRAY that takes a user-defined function and then applies it to local sub-blocks centered at each entry of the ARRAY. In order to support more diverse operations, users can also specify a stride parameter to skip applying the user-defined function to certain entries, e.g., every other entry along one dimension. One example of using APPLY is

$$B = Apply(A, f, \vec{c}, \vec{g}),$$

  where A and B are input and output ARRAY, respectively, f is a pointer of the user-defined function, and $\vec{c}$ and $\vec{g}$ are two vectors that specify chunk and ghost zone sizes.
- The STENCIL class is an abstract data structure to represent a general sub-block of an ARRAY. It is for user-defined functions to define the compute kernel over a set of neighboring entries around a reference entry. Just like the *view* concept in database, a STENCIL instance provides a logical view of a data array and there is no overhead to maintain multiple STENCILs over a physical array which is important for the parallel application of the compute kernel. In more details, STENCIL has a "center" being the reference entry of a sub-block which the user-defined function is applied to. Users can then use relative offsets from the center entry to specify the neighboring entries involved in the compute kernel. For example, consider the 5-points moving average operation applied over a time series. At each entry, the moving average operation needs 2 neighboring entries in both directions. Denote a STENCIL variable as s. The user-defined function for moving average operation can be expressed as

```
$f(s)\{$
    $\textbf{return} \ \frac{s(-2)+s(-1)+s(0)+s(1)+s(2)}{5}$
$\}$
```

  where $s(i)$ represents the value at the ith offset from the center and all these five entries together form a logical view of the array. When the APPLY method of ARRAY class above applies the user-defined function to a sub-block, it automatically creates a STENCIL instance as the input for the user-defined function.

III. DAS: DISTRIBUTED ACOUSTIC SENSING

Distributed acoustic sensing can re-purpose unused telecommunication fiber-optic cables as a series of single-component seismic sensors, with sensing point separations as fine as 1 meter or less. Comparing to the conventional spatially-discrete electronic sensors, DAS system has a large number of seismic sensors at low cost. With the improved spatial resolution of sensors, DAS has much larger capabilities in various seismic applications [11][7][2].

The DAS system we work on is based on a 25 kilometer fiber-optic cable connecting two cities in California. It provides 10,000 sensors (called channels in DAS) along the cable and adjacent channels are 2 meters away. These channels are indexed from 1 to 10000 in order. The sampling rate of each channel, as a sensor, is 500 Hz. An overview of this DAS system is shown in Figure 2a. As illustrated, the cable used by DAS goes through very diverse surroundings, leading to different and complicated noises at different DAS channels. An overall illustration of the recorded signals by DAS channels is plotted in Figure 2b which contains 6 minutes records around the time when an M4.4 earthquake happened near this DAS system.

The feature of interest in this paper is the recorded signals of traveling waves induced by seismic events like moving vehicles and earthquakes. Figure 3 shows a detailed example of the digital signals recorded by three neighboring channels and a traveling wave captured. The basic assumption is that recorded signals at each channel are a combination of noise and traveling waves, i.e.,

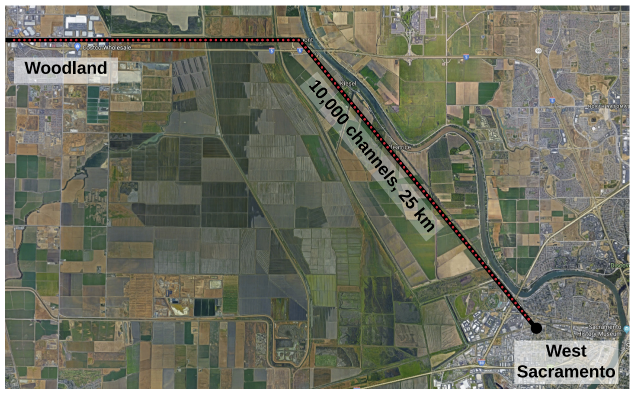
$$\text{DAS recorded signals} = \text{Noise} + \text{Traveling waves}.$$

In most of the time, the recorded signals are noises such as those flat signals shown in Figure 3 and the blue background in Figure 2b. The traveling waves may be strong enough to induce amplitude spikes in the recorded signals but may also be buried in the noise.

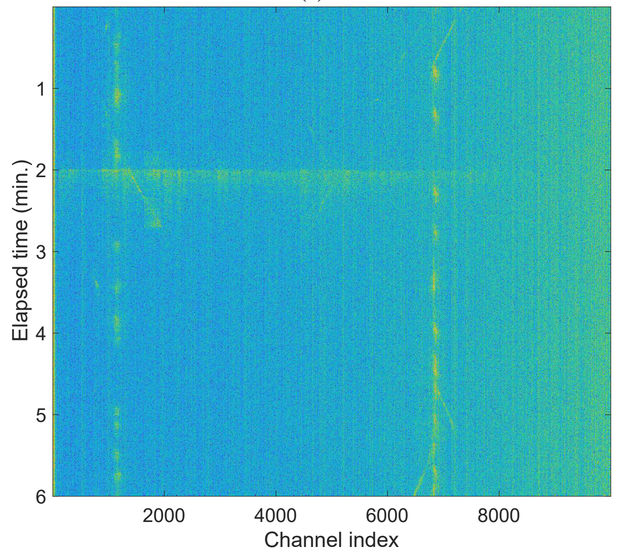
Of all the signal analysis of traveling waves, the most critical and basic task is to locate traveling waves in the records of each channel. Taking Figure 3 as an example, the same traveling wave arrives at these three well separated channels at different times. If knowing these arrival times, it is possible to estimate the speed and direction of this traveling wave. The main challenges of locating traveling waves in DAS data include:

- The large number of channels in DAS system generate enormous amount of data even for a short time period, e.g., around 12 terabytes data for one-week records with the DAS system in this study. Any data analysis method applied has to be computationally efficient and scalable.
- The DAS system has diverse noises at different channels which are induced by different surroundings as shown in Figure 2 and also by the optic-based measurement technique used in DAS.

In this work, we focus on analyzing the DAS data to locate traveling waves while trying to address the above challenges. In following discussions, we denote x_n as the



(a)



(b)

Figure 2: Deployment of the fiber-optic cable used by the DAS system from West Sacramento to Woodland in California. (a) The dotted line depicts the 25-kilometer cable going through high ways, bridges, farms, and other diverse surroundings. (b) Example of a 6-minute noisy DAS records of all the channels. Brighter color stands for larger signal amplitudes. Note that different channels have different background noise amplitudes. The horizontal bright line at the 2nd minute is associated with an earthquake.

signal vector recorded by channel n. We follow standard MATLAB routines to denote array entries and sub-blocks. For example, $x_n(i)$ is the recorded signal at wall time $t = i\delta$ and $x_n(i:j)$ is the recorded discrete signals in the time interval $[i\delta, j\delta]$ where $\delta = 0.002$ second is the unit time difference between two consecutive recorded signals according to the 500 Hz sampling rate of channels.

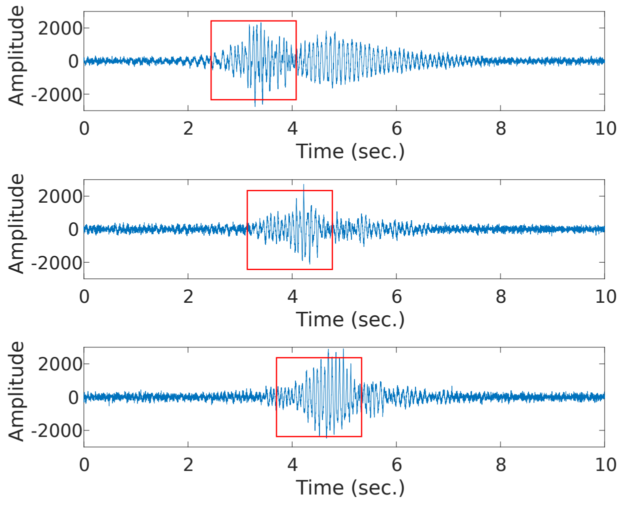


Figure 3: Example of recorded signals at the 7990th, 8000th, and 8010th channel in a 10-second time period. Red boxes mark a traveling wave arriving at the three channels with time lags.

IV. DAS DATA ANALYSIS AND ITS PARALLEL IMPLEMENTATION

To develop a scalable method for locating traveling waves in large and noisy DAS data, we introduce a local-similarity based method and its parallel implementation based on the data processing engine ARRAYUDF.

A. *Local similarity method*

The concept of local similarity [10] was introduced quite recently for seismic data analysis and is originally used for large arrays of conventional spatially-discrete electronic sensors. The basic idea is that if a strong seismic wave arrives at two different channels, the two channels should record "similar" signals at the corresponding time intervals. The similarity between two signal segments, say two vectors x and y, can be quantified by their absolute correlation without subtracting their mean values, i.e.,

$$\text{AbsCorr}(x, y) = \frac{|x^T y|}{\|x\|_2 \|y\|_2} = |\cos(\theta(x, y))|, \qquad (1)$$

where $\theta(x, y)$ is the angle between x and y. The two vectors are similar if x is close to y or $-y$ in terms of direction, which is equivalent to AbsCorr(x, y) being close to 1.

It is worth noting that two random noises from independent channels are always likely to have small absolute correlations and thus will not be identified to be highly similar. From another viewpoint, consider a signal segment $x_n\big((i-M) : (i+M)\big)$ at channel n recorded in the time interval $[(i-M)\delta, (i+M)\delta]$ around wall time $t = i\delta$ where M is an integer parameter to define the segment length. If there are signal segments in a neighboring independent channel similar to $x_n\big((i-M):(i+M)\big)$, we may identify the time interval $[(i-M)\delta, (i+M)\delta]$ at channel n to have some traveling waves.

Based on this idea, the **local similarity** between the target channel n and a neighboring channel, taking channel $(n+K)$ with a separation parameter K as an example, at wall time $t = i\delta$ is defined as,

$$S_{n,n+K}(i) = \max_{-L\leqslant l\leqslant L} \text{AbsCorr}\Big(x_n\big((i-M):(i+M)\big), x_{n+K}\big((i-M+l):(i+M+l)\big)\Big) \quad (2)$$

where time shift $l \in [-L, L]$ is introduced to capture traveling waves that arrive at the two channels with time lags. L is an integer parameter to define the maximum time shift range. $S_{n,n+K}(i)$ exactly captures the maximum similarity between the target signal segment $x_n\big((i-M):(i+M)\big)$ and all the signal segments at channel $(n + K)$ with small time shifts.

Symmetrically, $S_{n,n-K}(i)$ can be calculated between channel n and $(n-K)$. The local similarity of channel n at wall time $t = i\delta$ with all the defined neighboring channels, channel $(n - K)$ and $(n + K)$, is then defined as

$$S_n(i) = \frac{1}{2}\big(S_{n,n-K}(i) + S_{n,n+K}(i)\big). \quad (3)$$

Time window length $2M\delta$ and maximum time shift $L\delta$ are usually set as 2 and 0.6 seconds, respectively, leading to $M = 500$ and $L = 300$. Channel separation number K needs to be at least 5 since the recorded signals of any two channels that are less than 5 channels away are intrinsically correlated due to the measurement technique used in DAS. Using the recorded signal examples in Figure 3, the calculated local similarity vector $S_n = \big(S_n(i)\big)$ at the 8000th channel with the 7990th and 8010th channel as the neighboring channels is illustrated in Figure 4. As expected, the local similarity dramatically increases when the marked traveling wave arrives at the channel. Thus, for each channel, the common practice is to set up a threshold based on the calculated local similarities at all the times. Any time point with its local similarity above the threshold is then identified to have a traveling wave. This technique is referred to as *thresholding* in later discussions.

The original local similarity method, only for earthquake detection, further stacks the similarity vectors $\{S_n\}$ of all the sensors (channels in DAS case), i.e.,

$$\text{SLS}(i) = \frac{1}{N}\sum_{n=1}^{N} S_n(i), \quad N = \text{\# of channels}, \quad (4)$$

to smooth out high local similarities induced by local traveling waves. The method then applies thresholding to the stacked local similarity vector $\text{SLS} = \big(\text{SLS}(i)\big)$ for earthquake detections.

The main challenge of this local-similarity based method is the intensive calculation required since $S_n(i)$ at each

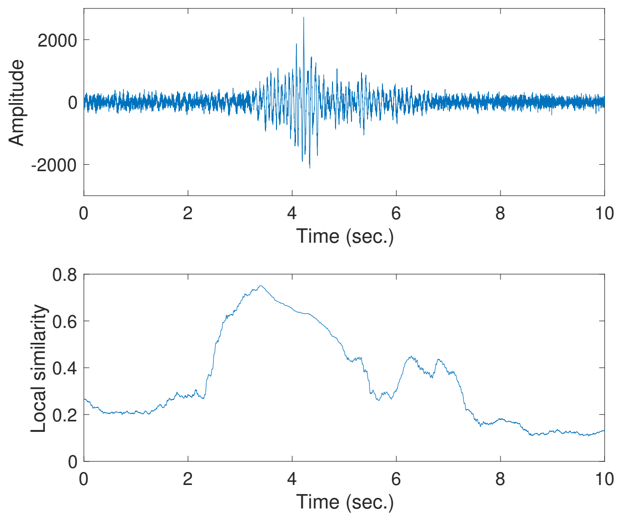


Figure 4: Calculated local similarity vector S_n (bottom) of the 8000th channel with $M = 500$, $L = 300$, and $K = 10$ using the recorded signals shown in Figure 3. The recorded signal at the 8000th channel is plotted (top) for reference.

channel n and at each time point $t = i\delta$ is calculated. Denote the number of channels as N and the number of recording time points as T. Ignoring $S_n(i)$ with indices n and i near boundaries, there are around NT local similarities $\{S_n(i)\}$ to be calculated. For each $S_n(i)$, its calculation takes around $48ML$ [1] arithmetic operations. In total, the computational complexity of this method is around $48MLNT$ which is quite large due to the magnitudes of N and T.

Based on fraction tests of our preliminary implementations, a sequential local similarity calculation in MATLAB for an one-minute DAS data is projected to take more than 200 hours. Meanwhile, a 64-thread shared memory python implementation will still take more than 40 hours. Thus, it is critical to parallelize and accelerate the local similarity calculation of DAS data.

B. Parallel scheme for local similarity calculation

DAS data is stored as a 2D array, denoted as D, whose columns are the discrete signal vectors $\{x_n\}$ recorded by all the channels. For any channel n, consider the calculation of $S_n(i)$ at wall time $t = i\delta$. By the definition (3), calculating $S_n(i)$ needs to access sub-arrays $D((i-M):(i+M), n)$ and $D((i-M-L):(i+M+L), n \pm K)$, which are centered at $x_n(i)$, or equivalently at $D(i, n)$. A simple illustration of this calculation locality is plotted in Figure 5.

It can be observed that the calculation of $S_n(i)$ only depends on entries with some fixed offsets to the center entry $D(i, n)$ and thus can be regarded as applying a

[1]The calculation mainly contains $(4L+2)$ times of AbsCorr evaluations betwxeen vectors of length $(2M + 1)$. One such AbsCorr evaluation has three vector inner-products and takes $3\times2\times(2M+1)$ arithmetic operations.

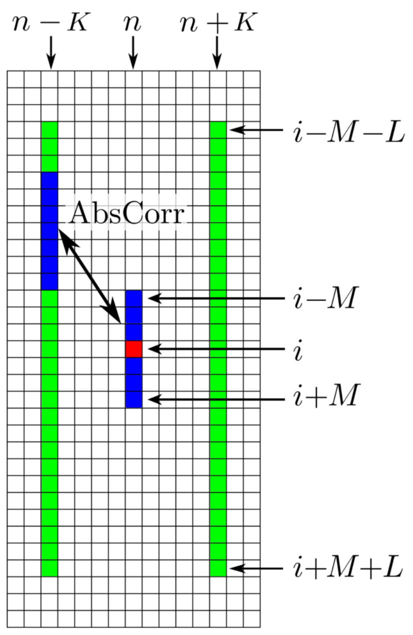


Figure 5: Illustration of data accesses in D for the calculation of $S_n(i)$. All colored entries are accessed. The red entry is $D(i,n)$ where the whole calculation is centered at. The blue bands on the $(n{-}K)$th column is an example of $D((i{-}M{+}l):(i{+}M{+}l), n{+}K)$. Important row/column indices are labeled.

fixed compute kernel to $D(i,n)$ and its neighboring entries. Specifically, for any fixed i and n, denote $\tilde{D}$ as the local sub-block of D centered at $D(i,n)$ of size $(2M+2L+1)\times(2K+1)$, i.e., the sub-block between the two green bands in Figure 5. Index $\tilde{D}$ in a symmetric way so that $\tilde{D}(0,0)=D(i,n)$ and the ranges of row and column indices of $\tilde{D}$ are $[-M-L, M+L]$ and $[-K,K]$ respectively. The compute kernel applied to $D(i,n)$ and its neighboring sub-block $\tilde{D}$ to calculate $S_n(i)$ is then specified in Algorithm 1.

Based on this interpretation, we can apply the same compute kernel to any entry of D and calculate the corresponding local similarity in parallel. The detailed idea of the parallel scheme for local similarity calculations of DAS data is illustrated in Figure 6. We first can split the DAS array D uniformly into sub-blocks, called chunks, at certain columns (for simplicity). For each chunk, add two ghost zones of which each contains K columns of D that are next to the chunk. Assign each chunk and its associated ghost zones to one computing node which applies the compute kernel Algorithm 1 to each entry in the chunk. Lastly, collect the partial local similarity results from each computing node together.

C. *Parallel local similarity calculation with* ArrayUDF

Combining the abstract parallel scheme in Section IV-B and the automated parallel data processing engine ArrayUDF in Section II, the calculation of all local similarities $\{S_n(i)\}$ of DAS data can be easily parallelized using ArrayUDF. The pseudocode of our implementation using ArrayUDF is presented in Figure 7. The user-defined function in ArrayUDF is defined exactly as the compute kernel in Algorithm 1 applied to each entry of DAS data D. Each entry $D(i,n)$ and its neighboring sub-block $\tilde{D}$ can be exactly abstracted as a Stencil variable centered at $D(i,n)$. Then, the local similarity can be computed by applying the user-defined function to one stencil. The benefit of using ArrayUDF in DAS data analysis is obvious. The execution engine in ArrayUDF can easily distribute our intensive local similarity calculation task onto many computing nodes while we, as users, only need to implement the compute kernel in Algorithm 1 and set up the chunk and ghost zone sizes properly.

Algorithm 1 Compute kernel for local similarity calculation

Note: $\tilde{D}$ is a sub-block of D centered at the target $D(i,n)$ and is of size $(2M+2L+1)\times(2K+1)$. It is indexed in a symmetric way so that $\tilde{D}(0,0)=D(i,n)$.

```
function LocalSimilarityKernel($\tilde{D}, M, L, K$)
    $W_n = \tilde{D}(-M:M, 0)$                         ▷ segment at channel $n$
    $S_{n,n+K} = S_{n,n-K} = 0$                        ▷ initialization
    for $l = -L : L$ do
        $W_1 = \tilde{D}((l-M):(l+M), +K)$
                                      ▷ segment at channel $n+K$
        $W_2 = \tilde{D}((l-M):(l+M), -K)$
                                      ▷ segment at channel $n-K$
        $S_{n,n+K} = \max\{S_{n,n+K}, \text{AbsCorr}(W_n, W_1)\}$
        $S_{n,n-K} = \max\{S_{n,n-K}, \text{AbsCorr}(W_n, W_2)\}$
                         ▷ maximum of the absolute correlations
    end for
    return $\frac{1}{2}(S_{n,n+K} + S_{n,n-K})$ ▷ local similarity $S_n(i)$
end function
```

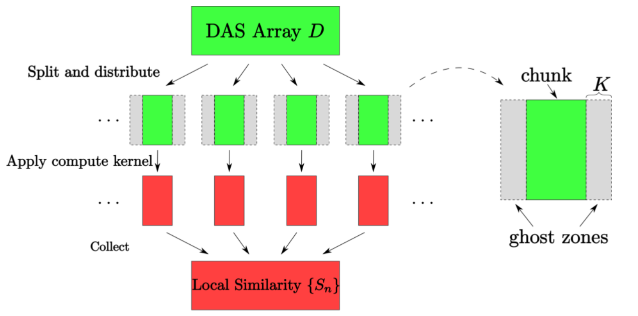


Figure 6: Diagram of the parallel scheme for the local similarity calculation of DAS data.

V. Results

We work on a 6-minute DAS data around time when an M4.4 earthquake happened near this DAS system, which

```
#define  M  500  // time window offset
#define  L  300  // time shift offset
#define  K  10   // neighbor channel offset
//User defined function on a Stencil
//to calculate local similarity.
float DAS_UDF(STENCIL s){
  //w0 has current signal segment
  w0 = s(−M:M, 0);
  //nw1 has entries of all shifted signal segments
  //on the Kth channel after the current channel
  nw1 = s(−(M+L):+(M+L), K));
  //nw2 has entries of all shifted signal segments
  //on the Kth channel before the current channel
  nw2 = s(−(M+L):+(M+L), −K);
  //Maximum AbsCorr with shifted signal segments
  for(i = 0; i < 2 ∗ L; i++){
    s1 = MAX(s1, AbsCorr(w0, nw1[i:i+2∗M]));
    s2 = MAX(s2, AbsCorr(w0, nw2[i:i+2∗M]));
  }
  return (s1+s2)/2;
}

int main(){
  vector<int> cs(2)={30000,10};  //Chunk size
  vector<int> gs(2)={0,10};
//Ghost zone size
  ARRAY<float> A("data.h5");
  ARRAY<float> B("result.h5");
  //Run DAS_UDF function using Apply method
  //Store the result in B
  B = A−>Apply(DAS_UDF, cs, gs);
}
```

Figure 7: Pseudocode for calculating local similarities $\{S_n(i)\}$ of DAS data using ARRAYUDF. The DAS data is stored in file "data.h5" and the result is stored in file "result.h5". DAS_UDF function implements the compute kernel in Algorithm 1 with a STENCIL variable. The chunk and ghost zone sizes specified in this code are examples for an one minute DAS data, i.e. the array is of dimension 30000×10000, with $K = 10$. The whole array is split into 1000 chunks. Each chunk has two ghost zones with extra 10 columns added to its two sides. For example, the chunk $D(:, 21 : 30)$ assigned to a computing node needs to be augmented with two ghost zones $D(:, 11 : 20)$ and $D(:, 31 : 40)$.

has also been plotted in Figure 2b. The data has dimension 180000×10000 and is of size 3.5 GB in HDF5 file format [16]. For simplicity, we fix the parameter $M = 500$ and $L = 300$ and test different channel separation number K.

A. Computational scalability

We use the NERSC CORI [2] supercomputer which is a Cray XC40 system with over 2000 computing nodes for the numerical results reported in this section. Each node has 32 Intel Xeon "Haswell" CPU cores. For each channel n, we only calculate local similarities $S_n(i)$ at every 10th recording

[2]https://www.nersc.gov/users/computational-systems/cori/

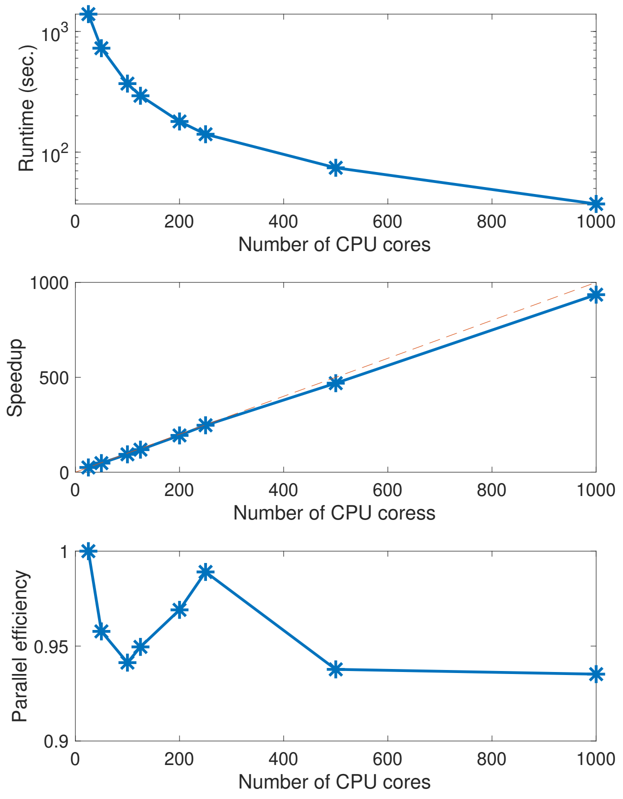


Figure 8: Runtime (top), speedup (middle), and parallel efficiency (bottom) of the local similarity calculation with different numbers of CPU cores. Number of CPU cores tested ranges from 25 to 1000. Runtime does not include the data reading and result writing processes. Speedup and parallel efficiency are defined as $\frac{25\times(\text{Runtime of 25 CPU cores})}{\text{Runtime of } N \text{ CPU cores}}$ and $\frac{25\times(\text{Runtime of 25 CPU cores})}{N\times(\text{Runtime of } N \text{ CPU cores})}$, respectively. The runtime of a serial calculation for this problem is projected to take nearly 10 hours through a fraction test.

time point since 0.02 second time resolution should be fine enough for locating traveling waves. In ARRAYUDF, chunk size is set as 180000×10 and thus the data array is split into 1000 chunks. Each computing node is uniformly assigned with multiple chunks for local similarity calculation. Figure 8 shows the runtime, speedup, and parallel efficiency of the whole computation using different numbers of CPU cores with $K = 10$. As can be noted, the parallel implementation based on ARRAYUDF scales very well and the execution is 935 times faster when using 1000 CPU cores.

B. Earthquake detection by thresholding

First, we follow the original local similarity method to calculate the stacked local similarity vector $\text{SLS} = (\text{SLS}(i))$ defined in (4) from $\{S_n\}$ of all the channels. With different channel separation number K, the stacked local similarity is plotted in Figure 9. The original method then sets up a threshold as the median of SLS plus ten times the median absolute deviation (MAD) of SLS, where MAD of SLS is defined as

$$\mathrm{MAD}(\mathrm{SLS}) = \mathrm{median}(|\mathrm{SLS}(i) - \mathrm{median}(\mathrm{SLS})|).$$

Even from direct observations, the only spike of each curve in Figure 9 clearly shows the time interval of the target earthquake. These results corroborate the validity of detecting earthquakes by applying the original local similarity method to DAS data.

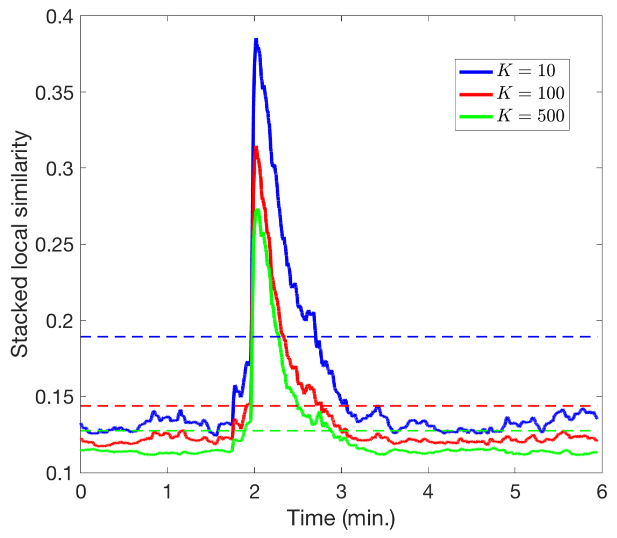


Figure 9: Stacked local similarity over all channels in DAS. The estimated detection thresholds for $K = 10, 100, 500$ are $0.19, 0.14, 0.12$ respectively and plotted in dash lines.

In addition, a subtle difference between these three curves and their thresholds in Figure 9 is that with larger K, the stacked local similarity has less variations in non-earthquake time period and thus can help to estimate earthquake arrival time more precisely through thresholding. A further explanation of this difference will be discussed later.

C. Constant threshold for general traveling wave detection

Different from estimating a threshold using median and MAD in the original method, we find that, from our numerical experiments, a constant threshold around $c_0 = 0.18$ works very well over the local similarities of all the channels to locate general traveling waves in DAS data. More specifically, any time point $t = i\delta$ at any channel n can be marked to have some traveling wave if the local similarity $S_n(i)$ is above c_0. An example of the general traveling wave detection with this constant threshold is shown in Figure 10.

To explain this new finding, Figure 11 shows the local similarities of two completely independent channels with

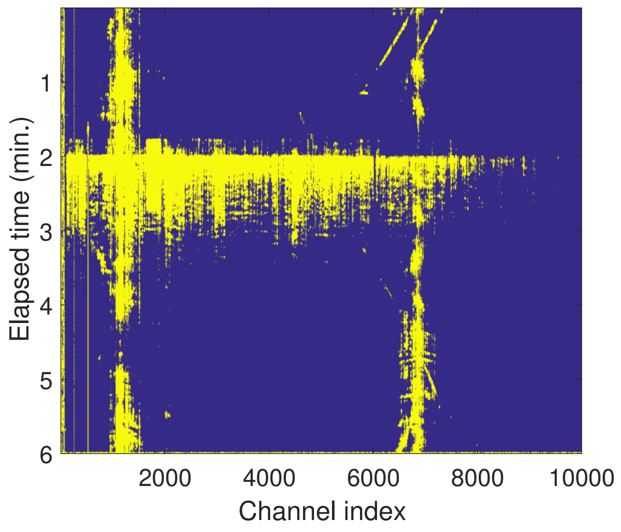


Figure 10: Traveling wave detection by local similarity thresholding with $c_0 = 0.18$ and $K = 10$. Yellow dots are marked time points at corresponding channels. Observations: (1) the two lines in area $[6000, 8000] \times [0, 1]$ are waves induced by two vehicles driving along the cable; (2) the two vertical bands in channels $[1000, 1500]$ and $[6000, 7000]$ are waves induced by some persistent vibrating sources; (3) the horizontal band at time $[2, 3]$ is associated with the target earthquake.

$K = 10$ and $K = 500$. Note that the local similarity baselines of these two independent channels with both tested K are quite close to a constant around 0.12 when there is no traveling wave. From all our numerical tests, it is always the case for any channels and any K as long as M and L are fixed. Also, different settings of M and L will lead to a different constant other than 0.12. Mathematically speaking, it is likely that this constant might be the mean value of the local similarities between two independent random background noises. This phenomenon is worth of further study. Therefore, taking local similarity variations into account, we heuristically choose the constant threshold $c_0 = 0.18$ for general traveling wave detections.

D. Screen local seismic events for earthquake detection

In terms of earthquake detections, the local traveling waves, such as those induced by vehicles and persistent vibrating sources detected in Figure 10 using constant thresholding, are also noises and need to be screened by further data processing. The main difference between these local traveling waves and earthquake wave is that local traveling waves can only reach small number of channels while earthquake wave can reach almost all the channels in DAS within a short time period.

Taking advantage of this difference, the original method stacks the local similarities $\{S_n\}$ of all channels to smooth

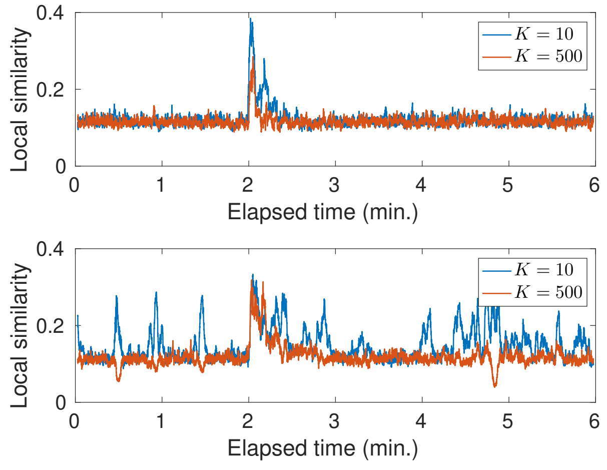


Figure 11: Example of local similarities at two independent and far apart channels with $K = 10$ and $K = 500$, respectively. Baselines of the four curves are all close to 0.12.

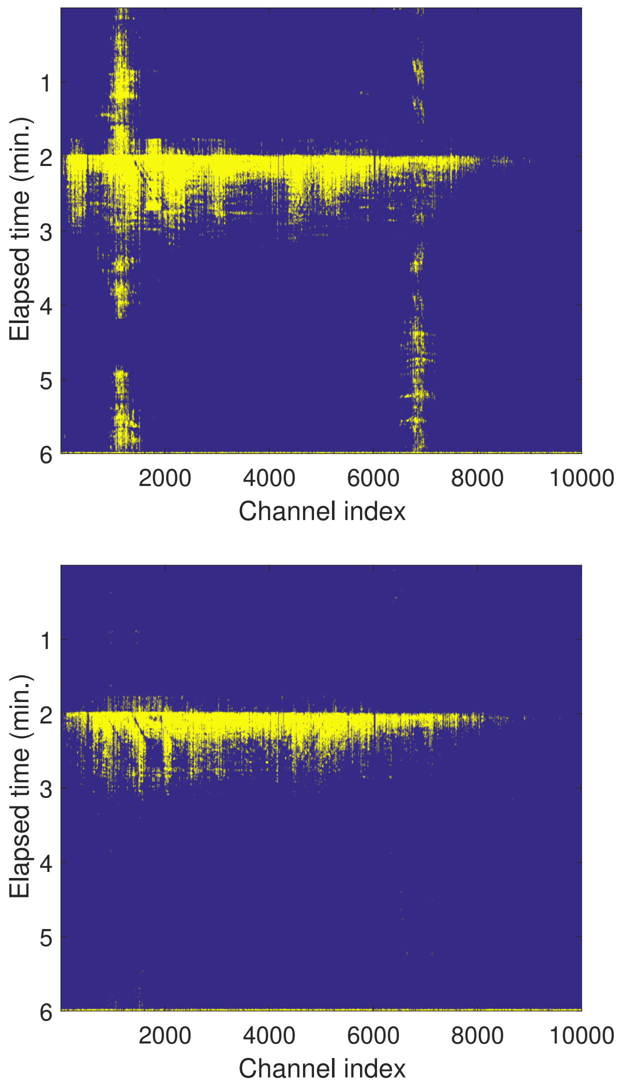


Figure 12: Traveling wave detection by local similarity thresholding with $K = 100$ (top) and $K = 500$ (bottom).

out those high local similarities induced by local traveling waves as described in Section V-B. Our new idea for DAS system is to increase channel separation number K in local similarity calculation and then combine with the constant thresholding as illustrated in Figure 10. The reason is that the local similarities between channel n and $n + K$, i.e., $S_{n,n+K}$ in (2), will not detect any local traveling waves that cannot spread $2K$ meters (adjacent channels are 2 meters away). Thus, large separation number K can help screen out the information of local traveling waves in the calculated local similarities.

Figure 12 shows the traveling wave detections by constant thresholding with local similarities calculated with $K = 100$ and $K = 500$. These results clearly show that local traveling waves appearing in Figure 10 fade out when larger K is used. From another viewpoint, we can also tell that the vehicle-induced traveling waves can only spread out less than 200 meters while the waves of those two persistent vibrating sources both can spread more than 200 meters but less than 1000 meters.

VI. Conclusions

In this paper, we briefly introduce the automated parallel data processing engine ArrayUDF and demonstrate its effectiveness with a compute-intensive feature extraction method with large DAS data. Around 95% parallel efficiency has been achieved based on the engine while using up to 1000 CPU cores in our application. It was able to complete a task in tens of seconds that was projected to take over 10 hours in a serial calculation. Furthermore, because it only requires the scientists to provide a core compute kernel, it is very simple for the scientists to easily exploit the computing power of supercomputers without programming anything in MPI.

In addition, our work also shows that the local-similarity based method can effectively locate traveling waves out of the background noise in DAS data. We plan to further explore additional data analysis tasks with ArrayUDF for seismological studies. These tasks may include analyzing long-term DAS data (several hundreds terabytes of data) for earthquakes not recorded by other devices and developing sensitive real-time earthquake detection methods based on DAS.

Acknowledgment

This effort was supported by the U.S. Department of Energy (DOE), Office of Science, Office of Advanced Scientific Computing Research under contract number DE-AC02-

05CH11231 (program manager Dr. Laura Biven). This research used resources of the National Energy Research Scientific Computing Center (NERSC), a DOE Office of Science User Facility.

References

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, et al. Tensorflow: A system for large-scale machine learning. In *OSDI 2016*, 2016.

[2] J. Ajo-Franklin, S. Dou, T. Daley, B. Freifeld, M. Robertson, C. Ulrich, T. Wood, I. Eckblaw, N. Lindsey, E. Martin, et al. Time-lapse surface wave monitoring of permafrost thaw using distributed acoustic sensing and a permanent automated seismic source. In *SEG Technical Program Expanded Abstracts 2017*, pages 5223–5227. Society of Exploration Geophysicists, 2017.

[3] N. Chaimov, A. Malony, S. Canon, C. Iancu, and et al. Scaling Spark on HPC Systems. In *HPDC 2016*, 2016.

[4] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. In *7th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 205–218, 2006.

[5] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM*, 51(1):107–113, Jan. 2008.

[6] B. Dong, K. Wu, S. Byna, J. Liu, W. Zhao, and F. Rusu. Arrayudf: User-defined scientific data analysis on arrays. In *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing*, pages 53–64. ACM, 2017.

[7] S. Dou, N. Lindsey, A. M. Wagner, T. M. Daley, B. Freifeld, M. Robertson, J. Peterson, C. Ulrich, E. R. Martin, and J. B. Ajo-Franklin. Distributed acoustic sensing for seismic monitoring of the near surface: A traffic-noise interferometry case study. *Scientific reports*, 7(1):11620, 2017.

[8] P. S. Earle and P. M. Shearer. Characterization of global seismograms using an automatic-picking algorithm. *Bulletin of the Seismological Society of America*, 84(2):366–376, 1994.

[9] S. J. Gibbons and F. Ringdal. The detection of low magnitude seismic events using array-based waveform correlation. *Geophysical Journal International*, 165(1):149–166, 2006.

[10] Z. Li, Z. Peng, D. Hollis, L. Zhu, and J. McClellan. High-resolution seismic event detection using local similarity for large-n arrays. *Scientific reports*, 8(1):1646, 2018.

[11] N. J. Lindsey, E. R. Martin, D. S. Dreger, B. Freifeld, S. Cole, S. R. James, B. L. Biondi, and J. B. Ajo-Franklin. Fiber-optic network observations of earthquake wavefields. *Geophysical Research Letters*, 44(23):11–792, 2017.

[12] A. Mateeva, J. Lopez, H. Potters, J. Mestayer, B. Cox, D. Kiyashchenko, P. Wills, S. Grandi, K. Hornman, B. Kuvshinov, et al. Distributed acoustic sensing for reservoir monitoring with vertical seismic profiling. *Geophysical Prospecting*, 62(4):679–692, 2014.

[13] T. Parker, S. Shatalin, and M. Farhadiroushan. Distributed acoustic sensing–a new tool for seismic applications. *first break*, 32(2):61–69, 2014.

[14] T. Perol, M. Gharbi, and M. Denolle. Convolutional neural network for earthquake detection and location. *Science Advances*, 4(2):e1700578, 2018.

[15] A. Shoshani and D. Rotem. *Scientific data management: challenges, technology, and deployment*. Chapman and Hall/CRC, 2009.

[16] H. Tang, X. Zou, J. Jenkins, D. A. Boyuka, S. Ranshous, D. Kimpe, S. Klasky, and N. F. Samatova. Improving read performance with online access pattern analysis and prefetching. In F. Silva, I. Dutra, and V. Santos Costa, editors, *Euro-Par 2014 Parallel Processing*, pages 246–257, Cham, 2014. Springer International Publishing.

[17] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *NSDI 2012*, 2012.

[18] J. Zhang, H. Zhang, E. Chen, Y. Zheng, W. Kuang, and X. Zhang. Real-time earthquake monitoring using a search engine method. *Nature communications*, 5:5664, 2014.