Computers & Geosciences 66 (2014) 145–154

Contents lists available at ScienceDirect

Computers & Geosciences

journal homepage: www.elsevier.com/locate/cageo

Large-scale seismic signal analysis with Hadoop

T.G. Addair [b], D.A. Dodge [a,*], W.R. Walter [a], S.D. Ruppert [a]

[a] Lawrence Livermore National Laboratory, 7000 East Avenue, MS 046, Livermore, CA 94550, USA
[b] Google Inc., 1600 Amphitheater Parkway, Mountain View, CA 94043, USA

ARTICLE INFO

Article history:
Received 12 October 2013
Received in revised form
9 January 2014
Accepted 29 January 2014
Available online 11 February 2014

Keywords:
Correlation
Hadoop
MapReduce
Seismology

ABSTRACT

In seismology, waveform cross correlation has been used for years to produce high-precision hypocenter locations and for sensitive detectors. Because correlated seismograms generally are found only at small hypocenter separation distances, correlation detectors have historically been reserved for spotlight purposes. However, many regions have been found to produce large numbers of correlated seismograms, and there is growing interest in building next-generation pipelines that employ correlation as a core part of their operation. In an effort to better understand the distribution and behavior of correlated seismic events, we have cross correlated a global dataset consisting of over 300 million seismograms. This was done using a conventional distributed cluster, and required 42 days. In anticipation of processing much larger datasets, we have re-architected the system to run as a series of MapReduce jobs on a Hadoop cluster. In doing so we achieved a factor of 19 performance increase on a test dataset. We found that fundamental algorithmic transformations were required to achieve the maximum performance increase. Whereas in the original IO-bound implementation, we went to great lengths to minimize IO, in the Hadoop implementation where IO is cheap, we were able to greatly increase the parallelism of our algorithms by performing a tiered series of very fine-grained (highly parallelizable) transformations on the data. Each of these MapReduce jobs required reading and writing large amounts of data. But, because IO is very fast, and because the fine-grained computations could be handled extremely quickly by the mappers, the net was a large performance gain.

1. Introduction

It has long been recognized that in collections of seismic data recorded by the same instrument from sources in similar locations there will be many similar seismograms (e.g. Geller and Mueller, 1980, Poupinet et al., 1984). Geller and Mueller (1980) attributed the similarity to the fact that for small magnitude earthquakes, propagation results in an effective low-pass-filtering of the signals to become essentially the Green's functions, so repeated ruptures of the same asperity produce the same seismogram.

Since the initial observations of doublets, researchers have exploited the phenomenon in a variety of different ways. Much work has centered on producing high-precision relocations of clustered seismicity by correlating the waveforms to obtain high-precision relative picks used to relocate the events. For example Fremont and Malone (1987) and Got et al. (1994) imaged structures underneath active volcanoes by relocation of multiplets. Rubin et al. (1999) imaged seismicity on creeping sections of the Hayward fault. Hauksson and Shearer (2005) relocated 327,000 Southern California earthquakes using waveform cross correlation.

Seismic tomography based on correlation of ambient seismic noise (Shapiro et al., 2005), seismic coda (Campillo and Paul, 2003), and higher-order methods e.g. C^3 (Stehly et al., 2006) has become a very important means of building high-resolution models of the Earth's crust and upper mantle. Although not the topic of this paper, these are data-intensive operations and could conceivably benefit from a Hadoop implementation at a sufficiently large scale of implementation.

Waveform correlation can also be used as the basis for highly sensitive detectors. This application has been known since at least the 1960s (Anstey, 1966) and has been employed on numerous occasions since. Because the correlation "footprint" of high frequency signals is generally quite small, correlation detectors have been used almost exclusively as "spotlight" detectors aimed at small regions of interest. Harris and Dodge (2011) used correlation in combination with subspace detectors (e.g. Harris, 2006) in an automated system to track events in an aftershock sequence.

However, interest in using correlation to process events over broader regions has grown. For example Schaff and Richards (2004, 2011) discovered that about 13% of 18,000 earthquakes recorded at regional distances in China were sufficiently well

* Corresponding author. Tel.: +1 925 423 4951.
E-mail addresses: tgaddair@gmail.com (T.G. Addair), dodge1@llnl.gov (D.A. Dodge), walter5@llnl.gov (W.R. Walter), ruppert1@llnl.gov (S.D. Ruppert).

http://dx.doi.org/10.1016/j.cageo.2014.01.014
0098-3004

correlated that they could be detected and located using waveform correlation.

To make better use of correlation relationships in seismic data analysis we need to understand what fraction of observed seismicity displays correlation relationships as functions of observed properties (e.g. source-receiver distance, window-length, bandwidth), and correlated source differences (e.g. location, depth, magnitude). The longer-term goal is to understand the physics underlying observed correlation behavior in terms of source similarity and path effects.

However, an impediment to our ability to investigate seismogram correlation is the computational costs of determining these relationships for the tens of thousands of seismic events each year observed at each of the tens of thousands of stations where data is available. Without an extremely efficient way of performing these computations, it is simply too difficult and time consuming to perform the many correlation runs that may be required to achieve a deep understanding of the phenomena under study.

This paper presents a methodology to significantly improve the speed at which massive amounts of seismic event data can be processed to look for correlation behavior using a Hadoop architecture. A result is to explore and expose the correlation behavior of large collections of seismic data. An expected outcome after analysis of these results is completed will be to make correlation a more useful tool in both geophysical research and seismic event cataloging operational systems.

2. Applying correlation to next-generation seismic pipelines

Organizations tasked with monitoring seismicity around the world (e.g. the United States Geological Survey National Earthquake Information Center, the Comprehensive nuclear-Test-Ban Treaty (CTBT) International Data Center, the US National Data Center, the International Seismological Center) and many in specific regions use a processing paradigm that was developed in the 1980s when average computer processing power was a tiny fraction of what is commonly available now. A single pass is made over the waveform data to extract a compact set of parameters such as seismic phase arrival time, amplitude and period. These are input to a phase associator, and the associated phases are fed to a locator, which produces the hypocenter solution. Although numerous refinements have been added over time, the basic procedure is unchanged.

For a variety of purposes, from improving the monitoring of the CTBT, to better characterization of earthquake hazard, to natural resource extraction and reservoir monitoring, it is necessary to detect, locate, and identify seismic events down to very low magnitudes even in the presence of interfering signals. As we near the limits of what can be done using the approach discussed above, pattern-matching-based processing looks increasingly attractive. A correlation between a new signal and a reference one, with a sufficiently high statistic, is at once a detection, location, and identification, assuming those properties are known for the template. A correlation detection and identification can be made with as little as one channel, without needing an associator. Since correlation detectors are much more sensitive than the power detectors used in current systems, it is likely that the detection threshold could be pushed down in all regions for which correlators are available.

Before considering the engineering aspects of a large-scale correlation-based seismic pipeline, it is crucial to understand how effective we can expect it to be. We need to know how much of the Earth's seismicity is correlated and how it is distributed. It may be that an appropriate system could only usefully target specific regions, and therefore its scope should be limited accordingly.

At Lawrence Livermore National Laboratory (LLNL) we operate a research database of seismic events and waveforms for nuclear explosion monitoring and other applications. The waveforms are digital time series of ground motion recorded by seismometers installed at the seismic stations. Typically, the seismometers produce output on multiple channels corresponding to different orientations and pass bands, so often a single event will be recorded on multiple channels at many stations. The LLNL database contains several million events associated with more than 300 million waveforms at thousands of stations (Fig. 1). We have correlated the waveforms in this database as a first step towards understanding the global distribution of correlated seismicity and to begin construction of a library of pattern detectors that could be used in a template-based seismic processing pipeline. We processed all channels to ensure no data are missed even though this entails some redundancy.

In this exploratory effort we correlated catalog events in a number of specific seismic phase windows (e.g. P, S) and the entire signal length, as well as in a number of frequency bands for each window. For each of the distinct station-channels (STA-CHAN) for which we have waveforms, we found all events whose catalog locations are separated by 50 km or less ("islands") and with average event-station distance $<=90^{\circ}$. For each pair of waveforms we applied the processing illustrated in Fig. 2. In all $\sim$6.8 billion waveform windows were processed.

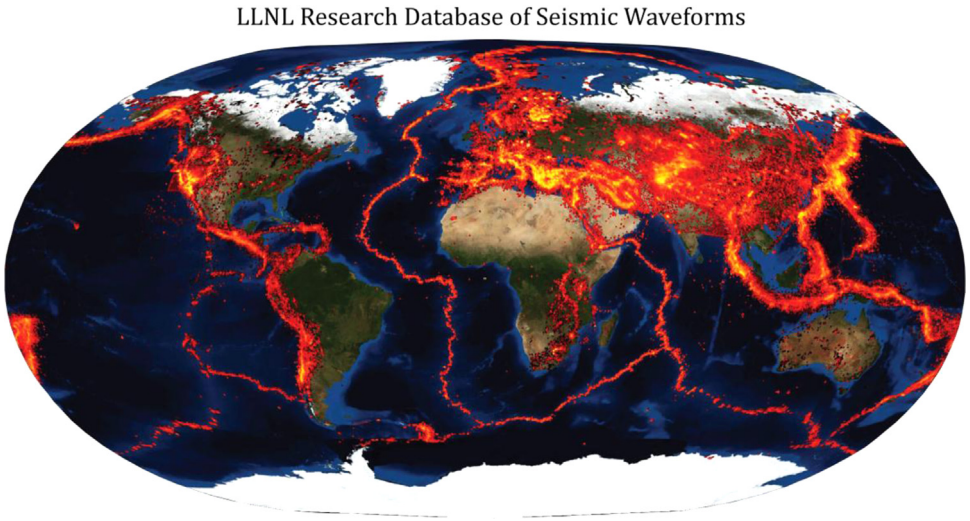

Fig. 1. The waveform density (number of waveforms in database per cell divided by the total number of waveforms). Color is proportional to log(density) with black lowest and white highest. Note that although the data set has global coverage, the density is highest in the Middle East, Eurasia, and Western North America.

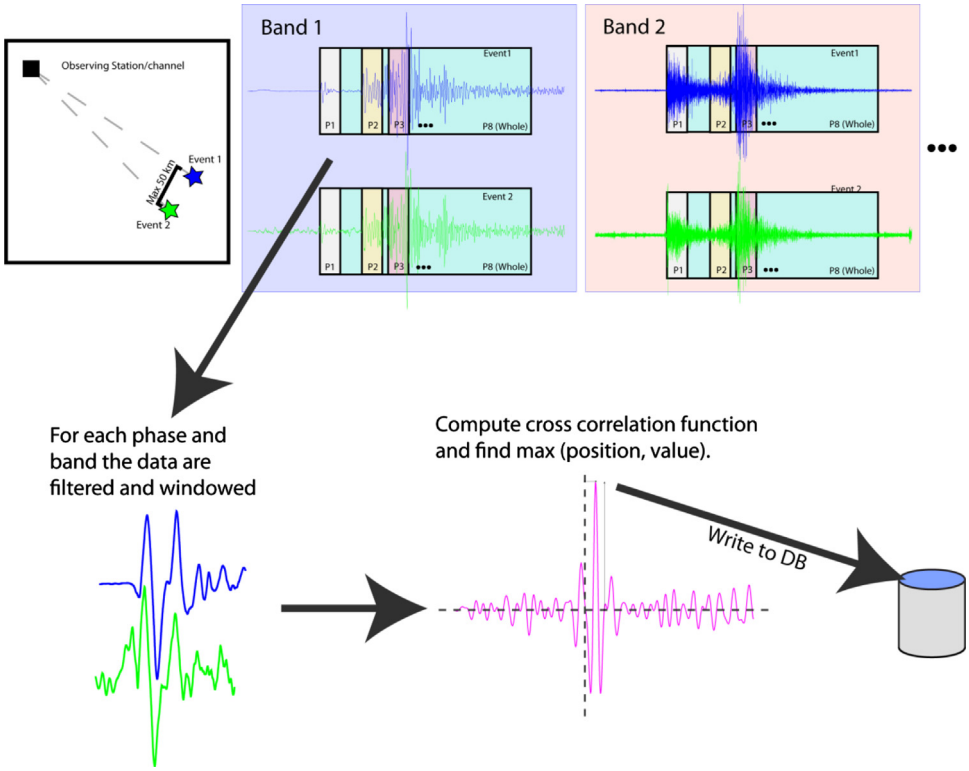

Fig. 2. A schematic illustration of the processing applied to a single channel for a pair of events observed by a single station. For each of B bands the seismograms are filtered and cut into W phase windows. For each window pair, the cross correlation function is computed and the max and its associated shift are recorded in the database.

A significant step in processing each window was identification of low SNR windows and artifacts. Low SNR is only a problem because it wastes CPU time, but corrupted and/or bad data (e.g. glitches, dropouts, severe clipping, no recorded signal, etc.) correlate quite well with each other and produce invalid results.

We used a decision tree classifier framework for quality control (QC) to automatically remove the problem signals from the correlation results. In the initial implementation we performed this step as part of results post-processing, and it resulted in the removal of nearly 280 million correlation pairs. In the Hadoop implementation we used the classifier to remove problem data prior to the correlation step. This gives a significant performance improvement, because for STA-CHAN "islands" with large N, each segment removed for QC reasons, results in N fewer correlations being computed.

Our first full-scale attempt at this process was run on an architecture consisting of 4 servers with 44 cores and 613 GB of RAM. All metadata and results were stored in an Oracle database, and the ~50 TB of waveform data were managed by a 2-head Hitachi file server. The time required to process the data was about 42 days. Over 370 million correlated waveform segments were found, and these results are still being analyzed.

Because we anticipate repeating this processing with even larger datasets and with variations such as multi-channel correlations, we must reduce the processing time significantly. This motivated our decision to move the entire workflow to a MapReduce architecture, which is the subject of the rest of this paper.

The transition of architectures from client–server to Hadoop is just the latest in a series of optimizations aimed at making the correlation processing fast. The client–server architecture used to process the dataset was itself the product of months of tuning. We knew at the outset that the correlation processing would have to be efficient, so we implemented a frequency-domain correlator that cached FFTs and autocorrelations. This made the cost of correlation $O(N)$ rather than the $O(N^2)$ expected in a naive implementation. We also parallelized the STA-CHAN processing at the same time. Later improvements included switching to an FFT object that used pre-computed butterflies and caching those objects.

Profiling showed that often the cores were mostly idle because the threads were blocked reading waveform data. We were able to partially address this by replacing our simple cell-based nearest-neighbor calculation with a calculation using an R-Tree (Guttman, 1984). This elimination of redundancy gave about a factor of 3 speedup. At this point, most of the remaining tuning addressed 2nd order issues, but we did discover one major and somewhat surprising performance hit. The application was creating and destroying objects at such a high rate, that the garbage collector started to soak up a significant fraction of CPU. In the end we had to revert to primitive types for a number of classes that were particularly troublesome.

3. Hadoop software framework

Apache Hadoop is an open source software framework for deploying data-intensive distributed applications. It implements a computational paradigm called MapReduce and the Hadoop Distributed File System (HDFS) derived from Google's MapReduce and Google File System (GFS) respectively (Dean and Ghemawat, 2004; Ghemawat et al., 2003).

The primary motivation of the MapReduce programming model is to create independent tasks that operate on arbitrary partitions of the input dataset in parallel during the map phase. During the subsequent reduce phase, data that must be rejoined or summarized are shuffled together and processed. When run over a distributed file system such as HDFS with nodes connected to dedicated storage, this framework naturally leverages data locality, whereby tasks running on a particular node process data resident to that node. By moving computation to the data, Hadoop allows

exceedingly fast IO and compute performance for highly parallelized algorithms.

Hadoop is written in Java, and thus writing MapReduce jobs in Hadoop most commonly involves writing explicit mappers and reducers in Java as well. This deliberate approach to MapReduce programming can encourage the developer to attempt to find a simple mapping of the problem to a single mapper and a single reducer. For many applications, including the waveform correlator, limiting the implementation to one MapReduce job means limiting parallelism where it could be of the greatest benefit. Moreover, the standard Java approach requires significant boilerplate for each job and provides little guidance for chaining jobs into a workflow. Consequently, we explored Pig (Natkovich, 2008) in an effort to find a higher-level tool for applying the MapReduce model.

Apache Pig is a platform for creating MapReduce workflows with Hadoop. These workflows are expressed as directed acyclic graphs (DAGs) of tasks that exist at a conceptually higher level than their implementations as series of MapReduce jobs. Pig Latin is the procedural language used for building these workflows, providing syntax similar to the declarative SQL commonly used for relational database systems. In addition to standard SQL operations, Pig can be extended with user-defined functions (UDFs) commonly written in Java. We adopted Pig for our implementation of the correlator to speed up development time, allow for ad hoc workflow changes, and to embrace the Hadoop community's migration away from MapReduce towards more generalized DAG processing (Mayer, 2013). Specifically, in the event that future versions of Hadoop are optimized to support paradigms other than MapReduce, Pig scripts could take advantage of these advances without recoding, whereas explicit Java MapReduce jobs would need to be rewritten.

3.1. First proof of concept

As part of a pilot project facilitated by Livermore Computing (LC), LLNL's institutional high-performance computing group, an effort began to explore porting the waveform correlator to Hadoop. LC's development cluster consisted of 10 Westmere nodes with 12 cores and 96 GB of RAM each. Most importantly for the IO bound waveform correlator, each node also featured dedicated storage to promote data locality. To compare the performance of the existing solution to Hadoop, a 1 terabyte (TB) test dataset was derived from a seismically dense region of the western United States and copied into HDFS.

Taking full advantage of the increased computational power and IO throughput of a Hadoop cluster requires significant parallelism of the application-level algorithms. Because every STA-CHAN in the original waveform correlator implementation was an independent task, and there were over 10,000 such tasks to perform, it would seem that the work was more than distributable enough to keep 10 hard-drives and 120 compute cores busy. In practice, however, there was significant imbalance in the effort performed by each task, as illustrated by Fig. 3, which shows the original implementation's time to completion for each task on the 1 TB test dataset.

As the histogram shows, most tasks in the original implementation executed in a matter of seconds. For these tasks, IO is the natural bottleneck as they do very little beyond reading a waveform, rejecting it, and requesting the next candidate. However, a handful of STA-CHANs containing dense clusters of high-quality data required several hours to fully process, imposing clear computational constraints on the performance. Any single job MapReduce implementation would be similarly constrained by these outliers, given that each mapper and reducer is run within its own single thread of execution. In order to increase the parallelism of the Hadoop correlator, and consequently improve performance, the problem needed to be broken up into finer-grained sub-problems that could be more evenly distributed across the cluster topology.

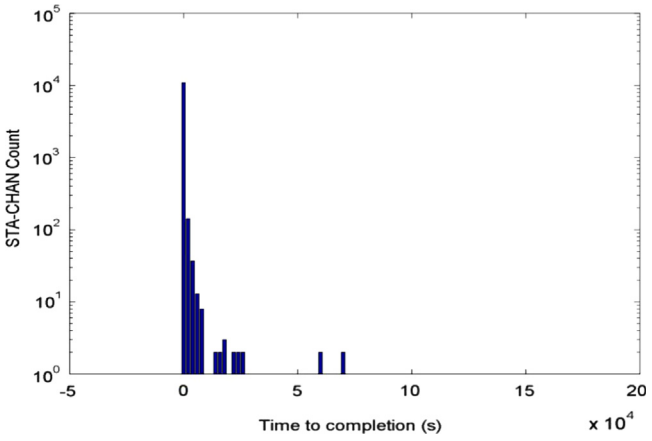

Fig. 3. The time to completion for each STA-CHAN task on the original architecture for the test dataset. Outlier tasks take several orders of magnitude longer to complete than average.

3.2. Overview of Pig workflow

The IO limitation of the original waveform correlator hardware lent itself to an architecture that attempted to minimize IO, going so far as to ensure that each waveform was read no more than once. In contrast, the Hadoop implementation takes advantage of the improved IO throughput by trading reads and writes to gain increased parallelism. As Fig. 4 shows, the first proof of concept Hadoop implementation added many additional reads, writes, and transformations to the data that did not exist in the original implementation. Despite this increase in design complexity, IO, and computation, a dramatic increase in parallelism across each of the steps allowed us to take full advantage of the cluster hardware. This increased problem granularity meant reduced work done in the bottlenecks, and spread the most computationally intensive procedures over many more threads of execution. A discussion of the major processing steps implemented in this pipeline follows.

3.2.1. Join metadata

For the purpose of minimizing disk usage and promoting consistency, LLNL maintains a highly normalized relational database management system (RDBMS) for storing seismic STA-CHAN and waveform metadata. This solution has worked well for our more typical interactive, single-workstation use cases, but creates an immediate bottleneck for highly parallel applications. Specifically, any application spread across tens of nodes and hundreds of cores can quickly saturate both the database's CPU and the bandwidth of the network interconnect. So it makes sense in an environment such as Hadoop to denormalize the dataset up front. We pre-join all tables to produce a single table containing all the columns required by the algorithm. By doing so, the need to request data randomly is eliminated, and the dataset can easily be partitioned and sent to independent tasks for processing. In contrast, a normalized solution would require each task to make requests for missing chunks of metadata, imposing a network limitation on the task.

Pig provides a native “join” function for taking two structured datasets and denormalizing them by some join predicate. Using Apache Sqoop, a command-line tool for transferring data between relational databases and Hadoop, we ingested all necessary Oracle tables into HDFS, and then used Pig to join the datasets into one denormalized metadata table containing the origin and station information for all waveforms as well as the waveform metadata.

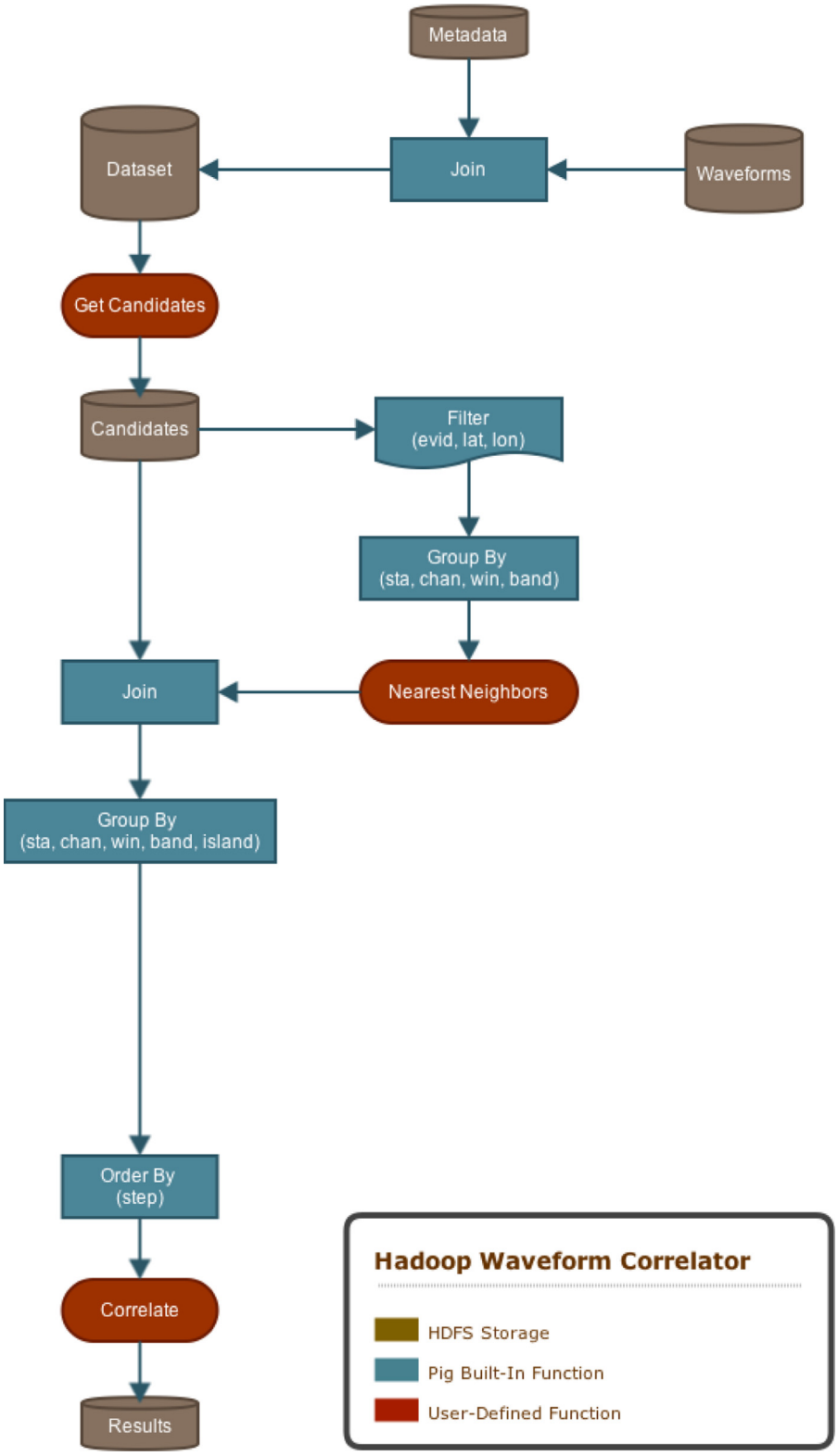

Fig. 4. The first proof of concept implementation of the waveform correlator as a Pig workflow consisting of many finely-grained passes over the data.

Fully denormalized, our metadata took just over 50 GB of usable space on disk.

We also joined the metadata to the binary waveform signal data directly by attaching a STA-CHAN-EVENT key onto the waveform data itself. By defining a simple schema consisting only of two fields (the unique key and the binary waveform data), we were able to perform a similar join in Pig to create a fully denormalized table containing all the necessary data for a given waveform in a single row along with the trace itself.

3.2.2. *Get candidates*

Having the dataset fully denormalized into rows of waveforms along with all relevant STA-CHAN and event metadata, we next

approached the issue of cutting and filtering the waveforms into valid candidates for correlation. This was in contrast to the original processing pipeline that took raw waveforms, checked for nearest neighbors, then cut and filtered the set of neighbors together. In a scenario where IO is the primary performance consideration, it makes sense to avoid cutting and filtering as a preprocessing step, as it necessarily requires reading all raw waveforms once, then writing and reading back in all cut and filtered waveforms in a second pass.

But, when IO is cheap, additional passes over the data to massage them into a more workable state can offer significant parallelization benefits. Cutting and filtering a waveform is an independent operation, and can thus be parallelized more finely than the complete set of operations on a STA-CHAN. For the 1 TB test dataset consisting of nearly 10 million traces on about 10 thousand STA-CHANs, that represents a factor of 1000 more tasks to be spread across the cluster. Such smaller tasks are necessarily less variable in the amount of work to be done, and thus distribute the computational effort more evenly.

Another benefit of performing the candidate generation task early in the processing is that it creates an opportunity to filter the input dataset down to something smaller and more manageable. This may seem counter-intuitive considering that S waveforms cut into W windows and filtered into B bands yields as many as SWB candidates. In practice, however, the majority of these preliminary candidates are discarded for having low signal-to-noise ratios or otherwise failing one of the decision tree classifiers used to assess the quality of the data. For the 1 TB test dataset, only about 1/2 S, or 4.5 million candidates were output from the candidate generation task.

3.2.3. Nearest neighbors

Left with only the cut and filtered waveforms that passed the initial quality control tests, the nearest neighbor calculation, too, can be parallelized beyond the STA-CHAN granularity with fewer events per task. This is because we only wish to correlate waveforms that are cut into the same window and filtered into the same band, and many of the events for a particular window and band were discarded in the previous step. Thus we can have a separate task for every STA-CHAN-WINDOW-BAND with only a fraction of the events to be processed per task.

An important implementation detail of the nearest neighbor calculation is that every task requires all of its event spatial data (latitude/longitude coordinates) to be in memory at once. At this point in the processing, this event metadata is currently in a denormalized table with all the binary waveform data, which will certainly not all fit in memory at once for a given STA-CHAN-WINDOW-BAND. Fortunately, Pig provides a mechanism to specify a subset of the fields (or columns, in relational database terms) to be sent to a given function. To reconcile the fact that the spatial event data are small enough to fit in memory, but the waveform data are not, we calculate the nearest neighbors only once.

The output from the nearest neighbor calculation will provide two additional pieces of metadata for each waveform: an “island” identifier and a step number. An island of events is a connected component in graph terminology. Given our criterion that a neighboring event is within 50 km, we can say that no two events in separate islands are within 50 km of each other. This is useful for the purpose of defining finer-grained sub-problems, as individual STA-CHAN-WINDOW-BAND-ISLANDs can now be correlated entirely independently, in parallel. However, even islands of events can be too large to fit entirely in memory, and so we need one additional piece of information to constrain the problem.

As mentioned previously, we output a step number for each waveform, representing the order in which we traversed the waveform's event within its island in the nearest neighbor calculation. We traverse the events within a task by first drawing an event from the set at random. We then use an R-Tree (Guttman, 1984) to fetch all neighboring events within 50 km with logarithmic asymptotic time complexity, and add the neighboring events to a queue. The events on the queue are the next to be processed in a similar manner, adding their previously unseen neighbors to the queue until no more neighbors exist, at which point we have discovered an island. Concretely, we say that for an island of N events, the waveform with step 1 corresponds to the first event we draw at random to seed the island, and the waveform with step N corresponds to the last event drawn from the queue that has no previously unseen neighbors. By processing events in a queue, instead of simply drawing them at random, we perform a breadth-first traversal of the island's corresponding graphical representation. This enables us to process neighboring events together, keeping them in memory while we process their neighbors, then discarding them as we migrate towards sections of the island farther away. Outputting the step value enables us to recreate this migratory processing without needing to keep all the spatial event information in memory during future processing tasks.

With the island and step metadata generated, this information is then rejoined to the complete waveform dataset, and grouped together by STA-CHAN-WINDOW-BAND-ISLAND using standard Pig functions.

3.2.4. Correlate

Every STA-CHAN-WINDOW-BAND-ISLAND can be processed independently. First, all events within an island are ordered by ascending step number. This ordered bag of waveform data is then fed to a user-defined function: CORRELATE. This is a special kind of user-defined function called an “accumulator” that processes elements within the ordered bag individually as opposed to loading them into memory at once. In this way, the CORRELATE function is stateful, invoked with one waveform at a time, and outputting to a separate bag of correlations as neighboring events are fed in. These correlations form the final output of the pipeline once flattened into a conventional comma-delimited text file with appropriate metadata attached.

Before we accumulate our first waveform, we define an event queue we will add to in the same order as we accumulate events, and we set a variable “current” to define the waveform against which all incoming waveforms will be correlated until they are no longer neighboring events, at which point we can safely discard the current event. This works because of the way we have ordered our events by step number, such that if an event follows another, either it is a neighbor or there are no more neighbors to process.

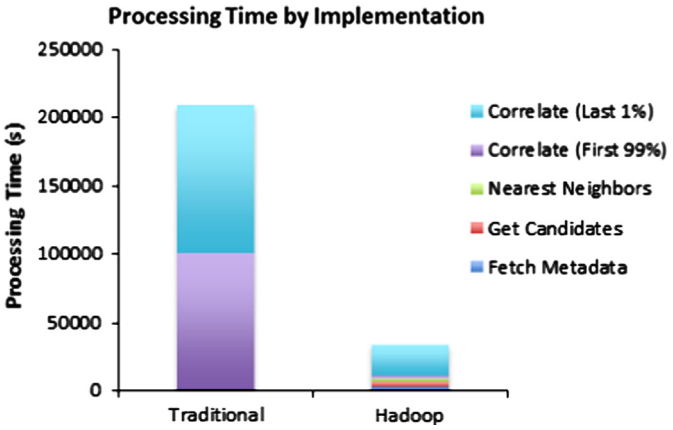

Fig. 5. A comparison of processing times for the test dataset on the original architecture (left) and the first Hadoop implementation (right). Times are in seconds.

A formal proof of this claim, with accompanying pseudo-code of the CORRELATE algorithm, can be found in Appendix.

Once there are no more neighbors to process, we remove the next event from the queue and declare it to be the current event. Necessarily, the incoming event must be a neighbor of the newly declared current event, or there are no unseen events for the current event. This is shown by Lemma 1 (Appendix). Once there are no more incoming events to accumulate, we drain the queue of its remaining events: correlating each newly removed event against all remaining events on the queue. In practice, we again speed up the neighbor calculation with an R-Tree.

3.3. Performance improvements and bottlenecks

Implementing the MapReduce workflow described above and deploying it on the Hadoop test cluster yielded a significant performance improvement over the original implementation of the correlator. However, certain bottlenecks in the processing pipeline remained. As Fig. 5 illustrates, processing time for 99% of the 1 TB test dataset went from nearly 28 h on existing hardware to 45 min on Hadoop. However, we observe that for the last 1% of the 1 TB test dataset the performance gap does not close as much between the two implementations: 30 h on existing hardware to over 6 hours on Hadoop.

Our initial suspicion was that the loose connectivity constraint imposed on the islands was leading to sprawling, yet sparse, islands of events that could be broken up into smaller, and minimally overlapping components. However, experimentation with algorithms including minimum graph cuts (Hao and Orlin, 1994) and the density-clustering DBSCAN (Ester et al., 1996) showed that our dataset contained a handful of large, highly dense islands of quality events. This made the prospect of further

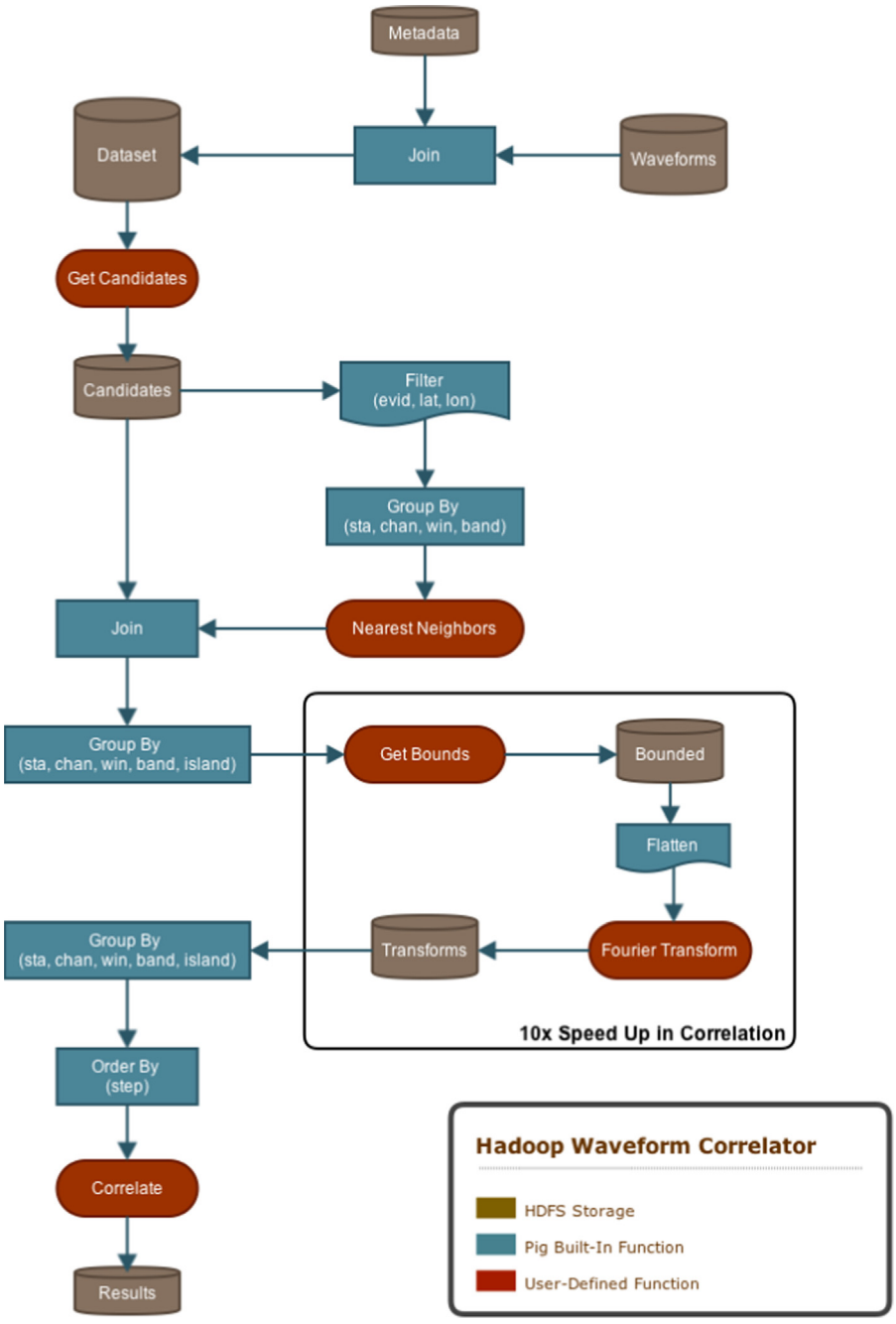

Fig. 6. The revised Hadoop processing flow with the added "Get Bounds" and "Fourier Transform" processing steps.

refining the granularity of our tasks difficult, since a fully connected graph of events requires that every event be correlated against every other event. In other words, in places where seismicity concentrated there can be a very large number of events within 50 km of each other that require all possible pairs of correlations to be generated.

In graph theory terms we can think of an island of neighboring events as a connected component of V vertices (events) and E edges (neighboring relations). In the CORRELATE process described above, we take time-domain waveform segments, and align them with their neighboring event segments before converting them into the frequency domain via a Fast Fourier Transform (FFT). Once the event and all its neighbors are in the frequency domain, we correlate them to produce a correlation coefficient between 0 and 1. Thus we must perform $O(E)$ FFTs per STA-CHAN-WINDOW-BAND-ISLAND task. For the extremely dense outlier islands mentioned above, $E \approx V^2$, which implies that $O(E) = O(V^2)$. For our test dataset, the largest complete island consisted of about 10,000 events, and thus roughly 100,000,000 FFTs confined to a single thread of execution. Naturally, this led us to consider ways to do better.

3.4. Refined proof of concept

The goal we had in mind for improving the performance of CORRELATE was to reduce the $O(V^2)$ FFTs down to a more manageable $O(V)$. An ideal solution would be to simply compute all FFTs once per event (vertex) and use the transformed, frequency domain segments in the CORRELATE step. This solution would have the added benefit of being embarrassingly parallel, capable of being performed as a pre-processing step with one task per waveform segment. However, calculating the FFTs once per event requires additional information about all the events to be correlated that is missing a priori. Before we can transform the segmented data into the frequency domain we need to know the minimum pre- and post-picked arrival times for which all segments to be correlated together have data. Once this information is obtained for a given STA-CHAN-WINDOW-BAND-ISLAND, all the segments can be trimmed down to the same length in seconds. Without trimming the segments to the intersection of their lengths in this manner, we run the risk of producing separate power-of-two length output segments from the FFTs. For the frequency-domain multiplication to work, the basis must of course be the same, and so the minimum pre- and post-pick times must be known prior to performing the FFT.

Under the constraint that a given waveform could only be read from disk once, calculating island-wide pre- and post-pick times would be impossible without exhausting heap memory, so in our first implementation we opted instead for performing more FFTs in exchange for less IO. However, that implementation revealed that reading and writing the entire dataset to HDFS could be performed on the order of seconds and minutes. Consequently, our revised solution was simple: add two more passes over the data to the pipeline.

The first new task would accumulate an entire STA-CHAN-WINDOW-BAND-ISLAND and calculate the pre- and post-pick times (GetBounds). Those two pieces of additional metadata would then be appended onto the segment-level metadata, and all FFTs would be calculated (FourierTransform). Each task would take a single waveform and produce exactly one transformed segment, allowing this step to be massively parallelized. This revised workflow is shown in Fig. 6. Fig. 7 shows that adding the two additional passes over the data, in spite of the increased IO cost, yielded a factor of 10 performance improvement in the CORRELATE routine. The added cost incurred by the highly dense outlier islands was greatly diminished.

Both of the additional processing steps contributed negligibly to the overall runtime of the system (less than 15 minutes combined).

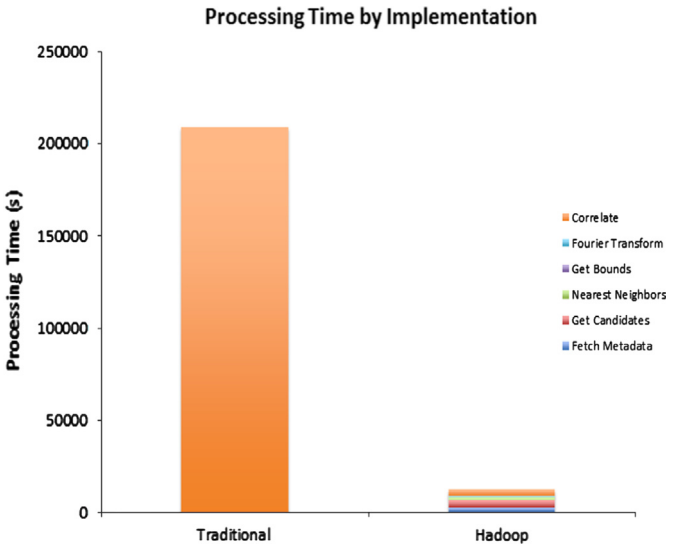

Fig. 7. The comparison of processing times for the test dataset on the original architecture (left) and the final Hadoop implementation (right). Times are in seconds.

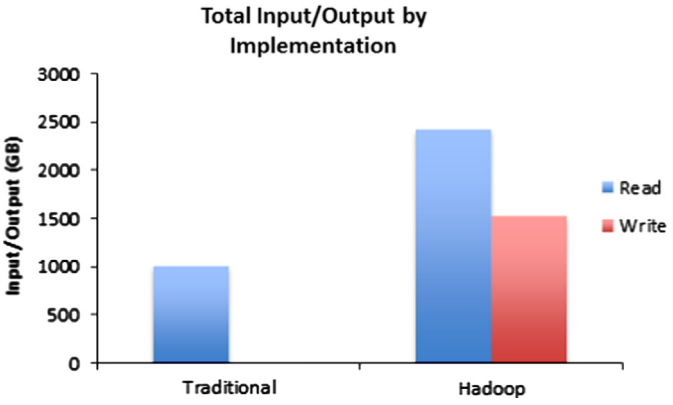

Fig. 8. A comparison of read/write times for the original implementation and the final Hadoop implementation. Times are in seconds.

Most importantly, the CORRELATE runtime was reduced to under an hour of processing. In aggregate, the refined Hadoop implementation yielded a factor of 19 improvement over the original waveform correlator, going from 48 h on the 1 TB test dataset to under 3 hours in total.

These performance gains include the time to read and write the data from and to HDFS, and were obtained in spite of the dramatic increase in total IO over the original implementation shown in Fig. 8.

4. Discussion

The Hadoop model of distributing the data with the computations presents a paradigm shift for the data-intensive scientific computing community. It demonstrates the need to change algorithmic priorities to fully take advantage of these powerful systems. Instead of asking ourselves how we can decrease the IO burden, as we did in our original implementation of the waveform correlator, we now find ourselves asking how we can increase the parallelism of our algorithms. Whereas before we had lots of CPU which could not be fully utilized, now we have blazingly fast IO and imbalanced CPU load. The process of finding new ways to break apart one's algorithms into finer-grained sub-problems is at the heart of the Hadoop philosophy: scale out, not up.

Based on our 1 TB test data set and the factor of 19 performance increase found by moving to the HADOOP architecture, we expect we will be able to re-correlate our entire ~50 TB, ~300 million waveform database in about 2 days instead of the original 42 days. This will dramatically improve our ability to conduct research on massive seismic datasets, and we intend to describe those results in future papers. The lessons of this study, making use of HADOOP to increase parallelism instead of reducing IO, can apply to many massive datasets of time series data, which are common in geophysics and other fields.

There is an important caveat, however: Namely the time cost of getting data into a Hadoop cluster. In our case the limitations are due to our NAS speed and to network throughput limitations. We estimate that one to two months may be required for an initial load of our complete holdings. For academic users who use data held at remote data centers, the latency would likely be much larger.

One solution may be to make the Hadoop cluster the only place holding waveforms and their metadata. In this model, users wishing to perform analytics submit jobs to the cluster, but users wishing to retrieve data make requests to a server which in turn retrieves the data from HDFS, packages it and returns it to the requester. We are studying this option for our own use.

The Hadoop ecosystem is evolving very rapidly with numerous SQL-over-Hadoop, streaming analytics, and time series databases under development. While this makes it hard to decide on an implementation right now, it seems very likely that in a year or two there will be some clear winners whose technology can be adopted for the seismological community.

Acknowledgments

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC. This is LLNL Contribution LLNL-JRNL-644626.

Appendix

The CORRELATE function, in pseudo code, works as follows:

```
current=null
queue=new Queue()

CORRELATE(incoming):
        if current==null:
                current=incoming
        else:
                queue.add(incoming)
                if neighbors(current, incoming):
                        correlate(current, incoming)
                else:
                        while not neighbors(current, incoming):
                                current=queue.dequeue()
                                for w in queue:
                                        if neighbors(current, w):
                                                correlate(current, w)
```

Lemma 1. *In the CORRELATE calculation, there is at all times a current event being correlated against all incoming events being removed from the queue. If an incoming event is not the neighbor of the current event, then all of the current event's neighbors have been correlated against the current event.*

Proof. In the NEAREST NEIGHBORS calculation, there is at all times a current event whose step number corresponds to the order it was removed from the queue in the breadth-first search. All of the current event's unseen neighbors are added to the queue together and assigned a step number greater than that of the current event. All previously unseen neighbors will be processed in a sequence together called the unseen sequence. □

By induction on the current event in the CORRELATE calculation, we will show that CORRELATE has correlated all of the current event's neighbors (the current candidates) by the time an incoming event is accumulated that is not a current candidate.

In the base case, at step 1, we accumulate the first incoming event from the bag of island events ordered by step number and promote it to be the current event. There are two possibilities: either the current event has an unseen sequence of current candidates to correlate, or it does not. If there is no remaining unseen sequence to process, then our claim is correct by definition. Thus we assume that the current event has an unseen sequence that needs to be processed. If the next incoming event is not a current candidate, then it must be part of another event's unseen sequence by definition of the connectivity of an island. By definition of the current event, the incoming event must be part of the unseen sequence of an event with a higher step number than the current event. But the current event's unseen sequence must necessarily come before the unseen sequence of any subsequent events by definition of the step number, and so this is a contradiction. Thus there are no more neighbors to correlate against the current event.

Suppose the claim holds for current events up to step $k-1$. At step k, we remove the kth event accumulated from the queue and promote it to be the current event. We then proceed to correlate it against all the elements on the queue. If the current event had any neighbors with a lower step number than itself, they were correlated earlier in the process by our assumption. Thus the only current candidates remaining must come from incoming events. Without loss of generality, we can apply the same reasoning to incoming events as applied in the base case to demonstrate that all remaining neighbors must be part of the next incoming sequence, or there are none left to correlate against the current event, and so the CORRELATE algorithm does not miss any potential correlations.

References

Anstey, N.A., 1966. Correlation techniques—a review. Can. J. Expl. Geophys. 2, 55–82.

Campillo, M., Paul, A., 2003. Long-range correlations in the diffuse seismic coda. Science 299, 547–549, http://dx.doi.org/10.1126/science.1078551.

Dean, J., Ghemawat, S., 2004. MapReduce: simplified data processing on large clusters. In: Proceedings of the OSDI'04, 6th Symposium on Operating Systems Design and Implementation, Sponsored by USENIX, in Cooperation with ACM SIGOPS, pp. 137–150.

Ester, M., Kriegel, H., Sander, J., Xu, X., 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In: Evangelos Simoudis, Jiawei Han, Usama M. Fayyad (Eds.). Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96), AAAI Press, pp. 226–231.

Fremont, M.J., Malone, S.D., 1987. High precision relative locations of earthquakes at Mount St. Helens, Washington. J. Geophys. Res. 92, 10,233–10,236.

Geller, R.J., Mueller, C.S., 1980. Four similar earthquakes in central California. Geophys. Res. Lett. 7, 821–824.

Ghemawat, S., Gobioff, H., Leung, S., 2003. The google file system. In: Proceedings of the 19th Symposium on Operating Systems Principles (Conference), Lake George, The Association for Computing Machinery, NY.

Got, J.L., Frechet, J., Klein, F.W., 1994. Deep fault plane geometry inferred from multiplet relative relocation beneath the south flank of Kilauea. J. Geophys. Res. 99, 15,375–15,386.

Guttman, A., 1984. R-Trees: a dynamic index structure for spatial searching. In: Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data–SIGMOD'84. p. 47. http://dx.doi.org/10.1145/602259.602266.

Harris, D. 2006, Subspace Detectors: Theory. UCRL-TR-222758, Lawrence Livermore National Laboratory, Livermore, California, pp. 46.

Harris, D., Dodge, D., 2011. An autonomous system for grouping events in a developing aftershock sequence. Bull. Seism. Soc. Am. 101, 763–774, http://dx.doi.org/10.1785/0120100103.

Hauksson, E., Shearer, P., 2005. Southern California hypocenter relocation with waveform cross-correlation, part 1: results using the double-difference method, Bull. Seism. Soc. Am. 95, 896–903.

Hao, J.X., Orlin, J.B., 1994. A faster algorithm for finding the minimum cut in a directed graph. J. Algorithms 17 (3), 424, http://dx.doi.org/10.1006/jagm.1994.1043.

Mayer, C., 2013. Hortonworks announce Stinger to solve Hadoop's real-time headache. ⟨http://jaxenter.com/hortonworks-announce-stinger-to-solve-hadoop-s-real-time-headache-46261.html⟩.

Natkovich, O., 2008. Pig—The Road to an Efficient High-level language for Hadoop, Hadoop Blog. ⟨http://developer.yahoo.com/blogs/hadoop/pig-road-efficient-high-level-language-hadoop-413.html⟩.

Poupinet, G., Ellsworth, W.L., Frechet, J., 1984. Monitoring velocity variations in the crust using earthquake doublets: an application to the Calaveras fault, California. J. Geophys. Res. 89, 5719–5731.

Rubin, A.M., Gillard, D., Got, J.-L., 1999. Streaks of microearthquakes along creeping faults. Nature 400, 635–641.

Schaff, D.P., Richards, P.G., 2004. Repeating seismic events in China. Science 303, 1176–1178.

Schaff, D.P., Richards, P.G., 2011. On finding and using repeating seismic events in and near China. J. Geophys. Res., 116, http://dx.doi.org/10.1029/2010JB007895.

Shapiro, N.M., Campillo, M., Stehly, L., Ritzwoller, M.H., 2005. High-resolution surface wave tomography from ambient seismic noise. Science 307, 1615–1618.

Stehly, L., Campillo, M., Shapiro, N.M., 2006. A study of the seismic noise from its long-range correlation properties. J. Geophys. Res. 111, B10306, http://dx.doi.org/10.1029/2005JB004237.