Distributed Algorithms to Find Similar Time Series

Oleksandra Levchenko, Boyan Kolev, Djamel-Edine Edine Yagoubi, Dennis Shasha, Themis Palpanas, Patrick Valduriez, Reza Akbarinia, Florent Masseglia

Distributed Algorithms to Find Similar Time Series *

Oleksandra Levchenko[1], Boyan Kolev[1], Djamel-Edine Yagoubi[1], Dennis Shasha[1,2], Themis Palpanas[3], Patrick Valduriez[1], Reza Akbarinia[1], and Florent Masseglia[1]

[1] Inria & LIRMM, Univ. Montpellier, France
first.last@inria.fr
[2] Dep. of Computer Sc., New York University
shasha@cs.nyu.edu
[3] University of Paris, France
themis@mi.parisdescartes.fr

Abstract. As sensors improve in both bandwidth and quantity over time, the need for high performance sensor fusion increases. This requires both better (quasi-linear time if possible) algorithms and parallelism. This demonstration uses financial and seismic data to show how two state-of-the-art algorithms construct indexes and answer similarity queries using Spark. Demo visitors will be able to choose query time series, see how each algorithm approximates nearest neighbors and compare times in a parallel environment.

Keywords: Time series · Indexing · Similarity search · Distributed data processing · Spark.

1 Introduction

As hardware technology improves for sensors, the need for efficient and scaleable algorithms increases to fuse the resulting time series. Sensors produce thousands and up to billions of time series, so the first step in fusion is often to find similar time series. Applications include statistical arbitrage strategies in finance and the detection of earthquakes in seismic data.

To handle such large numbers of time series, algorithms require high performance indexing. Creating an index over billions of time series by using traditional centralized approaches is highly time consuming.

An appealing opportunity for improving performance of the index construction and similarity search on such massive sets of time series, therefore, is to take advantage of the computing power of distributed systems and parallel frameworks. However, a naive parallel implementation of existing techniques would

* The research leading to these results has received funds from the European Union's Horizon 2020 Framework Programme for Research and Innovation, under grant agreement No. 732051.

under-exploit the available computing power. We have implemented parallel algorithms for two state-of-the-art approaches to construct indexes and to provide similarity search on large sets of time series by carefully distributing the work load. Our solution takes advantage of the computing power of distributed systems by using parallel frameworks, in this case Spark.

2 Parallel Similarity Search Methods

This section reviews similarity search methods with specific attention to parallel index construction both to increase speed and improve quality.

2.1 parSketch

parSketch [3] is a parallel implementation of the sketch / random projection-based method, both for index construction and querying. The basic idea is to multiply each time series in a database (or in a sliding window context, each window of a time series) with a set of random vectors, yielding a dot product. The vector of those dot products is a "sketch" for each time series. Then two time series can be compared by comparing sketches with approximation guarantees[1] that improve the more random vectors there are.

In our implementation of this idea, given a length m time series or a window of a time series, $\mathbf{t} \in R^m$, we compute its dot product with N -1/+1 random vectors $\mathbf{r}_i \in \{1, -1\}^m$. This results in N inner products (dot products) called the *sketch* (or random projection) of t_i. Specifically, $sketch(t_i) = (\mathbf{t_i} \bullet \mathbf{r_1}, \mathbf{t_i} \bullet \mathbf{r_2}, ..., \mathbf{t_i} \bullet \mathbf{r_N})$. We compute sketches for $t_1, ..., t_b$ using the same random vectors $r_1, ..., r_N$. By the Johnson-Lindenstrauss lemma [1], the distance $\|\mathbf{sketch(t_i)} - \mathbf{sketch(t_j)}\|$ is a good appproximation of $\|\mathbf{t_i} - \mathbf{t_j}\|$. Specifically, if $\|\mathbf{sketch(t_i)} - \mathbf{sketch(t_j)}\| < \|\mathbf{sketch(t_k)} - \mathbf{sketch(t_m)}\|$, then it's very likely that $\|\mathbf{t_i} - \mathbf{t_j}\| < \|\mathbf{t_k} - \mathbf{t_m}\|$. Our index is a set of grid structures to hold the time series sketches. Each grid maintains the sketch values corresponding to a specific set of random vectors over all time series.

Our implementation of the sketch-based approach *parSketch* parallelizes every step of algorithm: the computation of sketches, the creation of multiple grid structures, and the computation of pairwise similarity, thus exploiting each available core and taking full advantage of parallel data processing

2.2 DPiSAX

DPiSAX [4] is a parallel solution to construct the state-of-the-art iSAX-based index [2]. The iSAX representation is based on the PAA representation which allows for dimensionality reduction while providing the important lower bounding property. The idea of PAA is to have a fixed segment size, and minimize dimensionality by using the mean values of each segment.

The SAX representation takes as input the reduced time series obtained using PAA. It discretizes this representation into a predefined set of symbols, with a

given cardinality, where a symbol is a binary number. The iSAX representation uses a variable cardinality for each symbol of SAX representation, each symbol is accompanied by a number that denotes its cardinality.

Our parallel partitioned version of iSAX algorithm is based on a sampling phase that allows anticipating the distribution of time series among the computing nodes. *DPiSAX* splits the full dataset for distribution into partitions using the partition table constructed at the sampling stage. Then each worker builds an independent iSAX index on its partition.

3 Experimental evaluation

In order to provide an unbiased comparison, (i) all methods were implemented using the same tools, (ii) all the experiments were run in the same pre-deployed computing environment, and (iii) on the same datasets. Applications were implemented with Scala and Apache Spark. A distributed relational storage, set up as a number of PostgreSQL instances, is used for *parSketch* to store indexed data (grids). The implementation makes use of indexes to achieve efficient query processing. The *DPiSAX* implementation uses an HDFS cluster to keep index data in distributed files, so that partitions of the index are stored and retrieved in parallel.

Experiments were conducted on a cluster[4] of 16 compute nodes with two 8 cores Intel Xeon E5-2630 v3 CPUs, 128 GB RAM, 2x558GB capacity storage per node. The cluster is running under Hadoop version 2.7, Spark v. 2.4 and PostgreSQL v. 9.4 as a relational database system.

Search methods were evaluated over two real datasets and two synthetic ones. The real datasets are: Seismic that contains 40 million time series, and Finance with 72 million time series. For the purpose of experimentation, we generated synthetic Random Walk input datasets, whose sizes/volumes vary between 50M to 500M time series size of 256 points. At each time point, a random walk generator cumulatively adds to the value of the previous time point a random number drawn from a Gaussian distribution N(0,1). Another synthetic dataset is Random, containing 200M series, each of which is a close approximation to "white noise."

4 Demonstration

The user can observe and compare search method performances on a range of input datasets. The demonstration GUI enables the user to use drop-downs to choose the input dataset and set of queries, to vary specific parameters for methods: grid cell size (affects only the output of *parSketch*), Search type (only for *DPiSAX*) and then to observe the difference in performance (Figure 4).

A bar chart compares 3 methods in terms of time performance per batch of queries. We use three quality metrics: (i) Quality Ratio is defined to be correlation of the 10th time series found by a particular method divided by the

[4] http://www.grid5000.fr

10th closest time series found by direct computation of correlation. (ii) Recall is calculated as a fraction of relevant items in the top 10 time series found by particular method over the top 10 time series found by direct correlations. (iii) Mean Average Precision which considers the order of top 10 time series found by particular search method over ranked sequence of time series returned by direct correlations.

Line charts on the right side of the screen depict the top 10 time series found for the given input. The scroll bar allows the user to examine each query in the batch using visual plots and the quality ratio, for the different search methods.

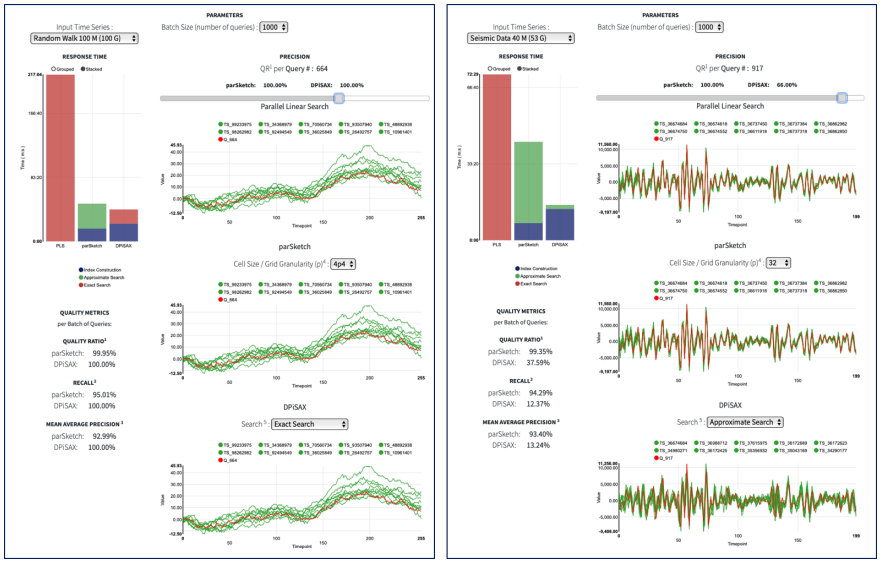

Fig. 1. Users can select input dataset and number of queries to execute, and can examine individual queries and answers, as well as quality and execution time of searches. The demonstration GUI and video are available at: http://imitates.gforge.inria.fr/

References

1. Achlioptas, D.: Database-friendly random projections: Johnson-lindenstrauss with binary coins. J. Comput. Syst. Sci. (2003)
2. Camerra, A., Shieh, J., Palpanas, T., Rakthanmanon, T., Keogh, E.J.: Beyond one billion time series: indexing and mining very large time series collections with iSAX2+. Knowl. Inf. Syst. (2014)
3. Yagoubi, D.E., Akbarinia, R., Kolev, B., Levchenko, O., Masseglia, F., Valduriez, P., Shasha, D.E.: Parcorr: efficient parallel methods to identify similar time series pairs across sliding windows. DMKD **32**(5), 1481–1507 (2018)
4. Yagoubi, D.E., Akbarinia, R., Masseglia, F., Palpanas, T.: Dpisax: Massively distributed partitioned isax. In: Int. Conf. on Data Mining (ICDM) (2017)