Parallel and Distributed Data Series Processing on Modern and Emerging Hardware*

Panagiota Fatourou[1,2]

[1] Foundation for Research and Technology, Institute of Computer Science, Greece
[2] University of Crete, Department of Computer Science, Greece
faturu@csd.uoc.gr

FORTH ICS TR 494 December 2024

Abstract. This paper summarizes state-of-the-art results on data series processing with the emphasis on parallel and distributed data series indexes that exploit the computational power of modern computing platforms. The paper comprises a summary of the tutorial the author delivered at the 15th International Conference on Management of Digital EcoSystems (MEDES'23).

1 Introduction

Ordered sequences of data points, known as *data series*, are one of the most common types of data. They are heavily produced by several applications in many different domains, including finance, astrophysics, telecommunications, environmental sciences, engineering, multimedia, neuroscience, and many others [?,?]. Similarity search is a fundamental building block in data series processing, as many types of complex analytics, such as, clustering, classification, motif and outlier detection, etc., are heavily dependent on it. A similarity search query is also useful in itself. It searches for the data seires in the collection that has the smallest distance to the query series, given some distance metric (which can be e.g., Euclidean distance [?] or Dynamic-Time Warping [?]).

As data series collections grow larger, sequential data series indexing technologies turn out to be inadequate. For example, the state-of-the-art such index, ADS+ [?], requires more than 4min to answer a single query on a moderately sized 250GB sequence collection. Thus, ADS is inefficient in processing the enormous sequence collections that are produced in many domains.

We briefly present a sequence of state-of-the-art indexes that take advantage of multiple nodes and modern hardware parallelization in each node, in order to accelerate processing times and achieve scalability.

We focus on a specific family of indexes, namely the *iSAX indexes* [?,?,?,?,?,?]. An iSAX index first computes the *iSAX summary* [?] of each data series in the

* Supported by the Hellenic Foundation for Research and Innovation (HFRI) under the "Second Call for HFRI Research Projects to support Faculty Members and Researchers" (project number: 3684).

collection (*summarization phase*). Then, it builds a tree storing all summaries (*tree construction phase*), and uses it to answer similarity search queries. Given a query q, the similarity search algorithm first traverses a path of the tree to find a first approximate answer to q. This approximate answer is called *Best-So-Far* (*BSF*). Then, it traverses the collection of data series to prune those whose summaries have higher distance to q's summary than the value stored in BSF. The distance between the summaries of two data series is called *lower bound distance*. It has been proved that if the lower bound distance of a series DS from q is higher than BSF, the real distance of DS from q is also higher than BSF. Then, DS can be pruned. Finally, the actual (real) distances between each data series that cannot be pruned and q is calculated. Whenever a real distance computation results in a lower value than that of BSF, then BSF is updated. Eventually, the answer to q is the value stored in BSF. If the pruning degree is high, a vast amount of expensive real distance computations are avoided. This results in good performance.

We first discuss ParIS+ [**?**,**?**], a *disk-based* concurrent index, capable to manipulate big data series collections stored in secondary storage. We then continue to present MESSI [**?**,**?**], the state-of-the-art *in-memory* concurrent index. Next, we discuss the challenges originating from the utilization of Graphics Processing Units (GPUs) [**?**] for further improving performance during query answering. We also discuss SING [**?**], an index that utilizes GPUs for query answering to perform better than MESSI. These indexes improve performance drastically in comparison to ADS+, the state-of-the-art sequential index [**?**]: to answer a similarity search query on a 100GB (in memory) data series collection, SING requires 35 msec whereas ADS+ requires several tens of seconds.

Finally, we discuss Odyssey [**?**], a *distributed* data-series (DS) indexing and processing framework, that efficiently addresses the challenges for efficient and highly-scalable *multi-node* data series processing.

All the indexes that we discuss in the next sections, exploit the Single Instruction Multiple Data (SIMD) capabilities of modern CPUs to further parallelize the execution of individual instructions inside each core.

2 ParIS+

ParIS+ [**?**,**?**] is a disk-based index that takes advantage of the computational power of multi-core architectures in order to execute in parallel the computations needed for both index creation and query answering. ParIS+ makes careful design choices in the coordination of the computational and I/O tasks, thus managing to completely hide the CPU cost during index creation under I/O.

ParIS+ is 2.6x faster than ADS+ [**?**] in index creation, and up to 1 order of magnitude faster in query answering. This makes ParIS+ a very efficient solution for disk-resident data. However, its answering time to a search query on a 100GB dataset is 15sec. This performance not only does not support interactive analysis (i.e., 100msec) [**?**], but also it is above the limit for keeping the user's attention.

To move data from disk to main memory, ParIS+ uses a double buffering scheme. While a coordinator thread moves data to one part of the buffer, a number of worker threads compute the iSAX summaries for the data series stored into the other part, and store them in a set of *iSAX buffers*. The use of the double buffering scheme enables ParIS+ to hide CPU computation under I/O.

The worker threads traverse the iSAX buffers storing pairs of iSAX summaries and the pointers to the corresponding data series, into the leaves of the index tree. The iSAX buffers are needed to ensure some form of data locality. All data series found in an iSAX buffer are stored in the same subtree of the index tree. ParIS+ uses a dedicated thread to build each subtree. This eliminates the need for costly synchronization and communication between the threads.

During query answering, ParIS+ first calculates BSF using the index tree (and never uses the tree again). Then, a number of threads concurrently traverse different parts of an array, called SAX, which contains the iSAX summaries of all data series of the collection, in order to create a list (*candidate list*) of those series that are good candidates to be the query's answer (i.e., those whose lower bound distance is smaller than BSF and cannot be pruned). Finally, a number of threads perform in parallel, using SIMD, the real-distance computations needed to find the closest data series to the query, among the candidate series.

3 MESSI

The second index, called MESSI [**?**,**?**], provides an efficient indexing and query answering scheme for *in-memory* data series processing. Fast in-memory data series computations often appear in real scenaria [**?**,**?**]. For instance, Airbus stores petabytes of data series, reasoning about the behavior of aircraft components or pilots [**?**], but requires experts to run analytics only on subsets of the data (e.g., on those relevant to landings from Air France pilots) that fit in memory. MESSI features a novel solution for answering similarity search queries which is 6-11x faster than an in-memory version of ParIS+, achieving for the first time interactive exact query answering times, at ∼60msec. It also provides redesigned algorithms that lead to a further ∼4x speedup in index construction time, in comparison to (in-memory) ParIS+.

The design decisions in ParIS+ were heavily influenced by the fact that its performance cost was mostly I/O bounded. Since MESSI copes with in-memory data series, no CPU cost can be hidden under I/O. Therefore, MESSI required more careful design choices and coordination of the parallel workers. This led to the development of a more subtle design for the index construction and new algorithms for answering similarity search queries on this index.

For query answering, in particular, the MESSI paper shows that an in memory version of ParIS+ is far from being optimal. Thus, MESSI proposes new solutions to achieve a good balance between the amount of communication among the parallel worker threads, and the effectiveness of each individual worker.

MESSI uses concurrent priority queues for storing the data series that cannot be pruned, and for processing them in order, starting from those that have

the smallest lower bound distance to the query series. In this way, the worker threads achieve a better degree of pruning. Moreover, MESSI assigns an iSAX summary to each node of the index tree. Then, it traverses the tree to decide which data series cannot be pruned based on these iSAX summaries. If the lower bound distance of a node from the query series is lower than BSF, the entire subtree rooted at the node can be pruned. In this way, the number of lower bound distance calculations performed is significantly reduced. To achieve load balancing, MESSI ensures that all priority queues have about the same number of elements, and workers use randomization to choose the priority queues they will work on.

4 SING

Although ParIS+ [**?**,**?**] and MESSI [**?**,**?**] exhibit advanced performance by exploiting the parallelism opportunities offered by the multi-core and SIMD architectures, they ignore the computational power of Graphics Processing Units (GPUs). SING [**?**] is the first (in-memory) data series indexing scheme that combines the GPU's parallelization opportunities with those of multi-core architectures (and SIMD), to further accelerate exact similarity search. SING outperforms MESSI in a variety of settings, reducing the MESSI cost for query answering to (almost) half.

Data series processing with GPUs is challenging for several reasons. First, the GPU memory is rather limited, so storing the entire raw data set in it is impossible. Moreover, the slow interconnect speeds disallows processing raw data in the GPU memory, as moving even small subsets of such data (i.e., those data series that are not pruned) in the GPU memory incurs a prohibitively high cost. Last, GPUs employ non-sophisticated cores, which are not readily suited to processing tree indexes and algorithms on top of them that frequently involve branching. These considerations imply that it is just (parts of) the query answering computation that can be performed efficiently in the GPU. Furthermore, the SING paper provides experimental evidence that simple GPU adaptations of the techniques employed by previously-presented tree-based indices for implementing those parts cannot outperform state-of-the-art solutions, such as MESSI.

To address these challenges, SING provides a new similarity search algorithm that runs on top of the tree structure created by MESSI. The algorithm ensures the efficient collaboration of both the CPU and GPU cores to answer queries. The main ideas on which it is based are the following. SING stores in the GPU memory, an array of (just) the iSAX summaries of the data series in the collection. This array is sorted in the order the data series appear in the leaves of the index tree. SING performs an inital pruning phase to prune entire root subtrees, thus reducing the number of lower bound distance computations that the GPU executes. Moreover, it employs a simple polynomial function to compute lower bound distances. This enables the computation of lower bound distances in their entirety within the GPU. SING employs streaming to effectively synchronize the parallel execution of CPU and GPU threads.

5 Odyssey

The data series indexes discussed in the previous sections, operate on a single node. Thus, they do not take advantage of the full computational power of modern distributed systems comprised of multiple multi-core nodes. Odyssey [**?**] is a state-of-the-art *distributed* data-series processing framework, which ensures good speedup and high scalability, thus addressing two major challenges of distributed data series indexing. An *ideal* distributed data series index should achieve linear speedup or be able to process data collections whose size is proportional to the number of nodes in the system. Experiments show [**?**] that Odyssey does not notably depart from this ideal behavior.

Odyssey provides a collection of scheduling schemes for query batches, which result in good performance under several different workloads. These schemes are based on predicting the execution time of each query. Odyssey provides a query analysis that shows correlation between the total execution time of a query and its initial best-so-far value. This analysis enables the calculation of the predictions.

Odyssey achieves a good degree of load-balancing among the nodes even in settings where the execution time predictions are not accurate. Odyssey's load balancing algorithm illustrates how to efficiently implement the work-stealing approach [**?**,**?**,**?**,**?**,**?**,**?**,**?**] in distributed data series indexing settings. In the work-stealing approach, nodes sitting idle may steal work from busy nodes. The difficulty here is how to achieve this without ever moving any data around, as moving data between nodes would be prohibitively expensive. In Odyssey, this is esured by employing a (partial) replication scheme, which is flexible enough to allow Odyssey to navigate through a fundamental trade-off between data scalability, and good performance during query answering. Specifically, no data replication would minimize space overhead, but it would dissallow a cheap load-balancing solution, thus resulting in higher execution times for answering queries. In Odyssey, a user may choose the replication degree that is the most relevant to its application based on its scalability and performance needs.

Odyssey is the first data series index that supports parallelization outside the boundaries of a single node, without sacrificing the good performance of state-of-the-art indexes [**?**,**?**,**?**] within a node. This was not a trivial task, as experiments show that simple solutions of using as many instances of a state-of-the-art data series index as the number of nodes would not result in good performance, mainly due to severe load balancing problems. Supporting work-stealing on top of a state-of-the-art index would require moving data around. Odyssey *single-node* indexing algorithm borrows techniques from MESSI [**?**,**?**], but it also provides new mechanisms to allow a node v to rebuild parts of the index tree of another node v' needed for executing the load that v steals from v'. This requires the utilization of a different pattern of parallelism in traversing the index tree to produce the set of data series that cannot be pruned, and new implementations for populating and processing the data structures needed for efficient query answering.

Experiments show that Odyssey's index creation exhibits perfect scalability as both the dataset size and the number of nodes increase. Moreover, Odyssey exhibits up to 6.6x times faster exact query answering times than other state of the art parallel or distributed data series indexes.

6 Discussion

We discussed state-of-the-art parallel and distributed indexes for answering similarity search queries on big collections of data series in a fast and scalable way. We started with concurrency techniques for disk-based indexes. We continued with efficient parallelization techniques for indexes built to work on in-memory data. We also focused on techniques for data series processing that utilize GPUs to further speed up computation. Finally, we touched upon the main challenges that are introduced when moving from a single multi-core node to a distributed system with many multi-core nodes, and summarized state-of-the-art techniques for addressing these challenges.

We focused on solving the problem of *exact* similarity search on big collections of *fixed-length* data series. Exact similarity search is a core operation needed in many critical data analytics tasks (including outlier detection, frequent pattern mining, clustering, classification, and others) [**?**]. An interesting open problem is whether the discussed techniques can be easily extended to efficiently support other types of similarity search queries, such as approximate search, without or with (deterministic or probabilistic) guarantees [**?**]. Also, it is interesting to study how these parallelization techniques can be extended to indexes that handle data series of variable length [**?**].

Concurrent iSAX indexes are *locality-aware*. They maintain some form of data locality, and achieve high parallelism with low synchronization cost by having threads working, independently, on different parts of the data as much as possible, to avoid the need of frequent communication between threads. However, they all employ locks, thus, they are blocking. If a thread holding a lock becomes slow (or crashes), the entire index blocks without being able to make any further progress. *Lock-freedom* [**?**] is a widely-studied property when designing concurrent trees [**?**,**?**,**?**,**?**] and other data structures [**?**,**?**,**?**]. It avoids the use of locks, ensuring that the system, as a whole, makes progress, independently of delays (or failures) of threads. A recent study [**?**] presented the first lock-free iSAX index, FreSh, together with Refresh, a generic approach that can be applied on top of any iSAX index to provide lock-freedom without adding any performance cost. An interesting open problem is to study the fundamental performance properties that govern other families of data series indexes, and come up with generic schemes for achieving locality-aware, lock-free synchronization on top of them.

The tree employed by an iSAX index has big root fan-out and its root subtrees are leaf-oriented binary trees. This type of trees support locality-aware parallelization, using locks, in a relatively easy way. Experimental work [**?**] has shown that no single indexing method is an overall winner in exact query an-

swering. For this reason, recent indexing schemes [**?**] incorporate key ideas from more than one data series indexing families with the goal of exhibiting the combined performance power of the different incorporated approaches. It would be interesting to see what kind of parallelization techniques would be suitable to types of indexes that are not in the iSAX family [**?**], or for indexes that follow combined approaches [**?**].

Next generation computer systems will utilize emerging memory technologies, such as Non-Volatile Memory (NVM), to address the high computation demands of modern applications and provide persistence [**?**,**?**]. The availability of non-volatile memory has increased the interest in the crash-recovery model, in which failed threads may be resurrected after the system crashes. Developing a recoverable data series index that will exploit the NVM performance benefits and offer recoverability, distilling the best from both approaches, disk-based [**?**] and in-memory [**?**] data series indices, is an interesting open problem.

Acknowledgements This work has been supported by the Hellenic Foundation for Research and Innovation (HFRI) under the "Second Call for HFRI Research Projects to support Faculty Members and Researchers" (project number: 3684).