RINSE: Interactive Data Series Exploration with ADS+

Kostas Zoumpatianos
University of Trento
zoumpatianos@disi.unitn.it

Stratos Idreos
Harvard University
stratos@seas.harvard.edu

Themis Palpanas
Paris Descartes University
themis@mi.parisdescartes.fr

ABSTRACT

Numerous applications continuously produce big amounts of data series, and in several time critical scenarios analysts need to be able to query these data as soon as they become available. An adaptive index data structure, ADS+, which is specifically tailored to solve the problem of indexing and querying very large data series collections has been recently proposed as a solution to this problem. The main idea is that instead of building the complete index over the complete data set up-front and querying only later, we interactively and adaptively build parts of the index, only for the parts of the data on which the users pose queries. The net effect is that instead of waiting for extended periods of time for the index creation, users can immediately start exploring the data series. In this work, we present a demonstration of ADS+; we introduce RINSE, a system that allows users to experience the benefits of the ADS+ adaptive index through an intuitive web interface. Users can explore large datasets and find patterns of interest, using nearest neighbor search. They can draw queries (data series) using a mouse, or touch screen, or they can select from a predefined list of data series. RINSE can scale to large data sizes, while drastically reducing the data to query delay: by the time state-of-the-art indexing techniques finish indexing 1 billion data series (and before answering even a single query), adaptive data series indexing can already answer $3 * 10^5$ queries.

1. INTRODUCTION

Big Data Series. The need for accessing, exploring and analyzing large collections of data series concerns a big number of diverse domains, affecting both science and industry. Formally, a data series $D = (e_1, ..., e_n)$ is defined as a sequence of elements $e_i = (v_i, p_i)$, where each element consists of a value v_i and a position p_i that implies its ordering. Such domains include meteorology (e.g., temperature), chemistry (e.g., mass spectroscopy), finance (e.g., stock quotes), smart cities (e.g., road traffic), marketing (e.g., opinion evolution), and others. These data have to be analyzed, in order to identify patterns, gain insights, detect abnormalities, and extract useful knowledge. In order to perform such complex data mining tasks, there is a need of efficiently processing similarity queries, such as range and nearest neighbor search.

Proceedings of the VLDB Endowment, Vol. 8, No. 12
Copyright 2015 VLDB Endowment 2150-8097/15/08.

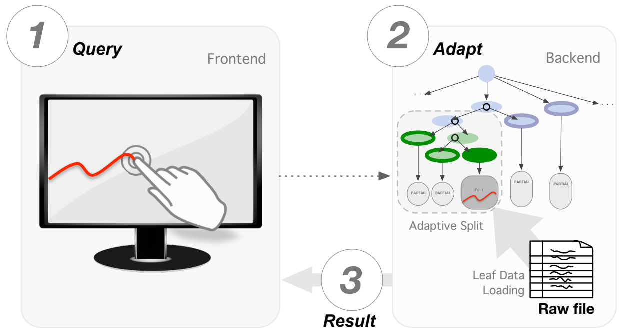

Figure 1: Interactive Data Series Exploration

In addition, analysts and scientists need to explore the data by firing sets of exploratory queries, which are not known a priori [12]. In such cases, performing tuning and initialization actions up-front suffers from the lack of knowledge about which data parts are of interest, e.g., [9, 10].

Data Series Indexing. To enable this kind of analysis though, query answering times have to be kept at an interactive speed. A common approach for optimizing similarity search queries using a distance measure e.g., Dynamic Time Warping and Euclidean Distance, is to perform a dimensionality reduction technique such as *Discrete Fourier Transforms (DFT) [1], Discrete Wavelet Transforms (DWT)* [5], *Piecewise Aggregate Approximation (PAA)* [13], or *Symbolic Aggregate approXimation (SAX)* [14] and use this representation for indexing using various spatial [6, 2] or specialized indexes, such as the iSAX family [15, 3, 4] and DS-Tree [16].

Problem Definition. The problem with such methods, is that they take hours or even days to be built. This can be a show-stopper for many applications that either require immediate access to the data or the amount of exploratory queries does not justify the cost that has to be paid upfront.

Adaptive Data Series Indexing. In this work, we study the data to query time bottleneck and the index creation bottleneck for interactive exploration of very large collections of data series. We present a demonstration of Adaptive Data Series index (ADS+) [17], the first adaptive indexing method for data series. ADS+ minimizes the index creation time allowing users to query the data soon after its generation and several times faster compared to state-of-the-art indexing approaches. ADS+ is based on

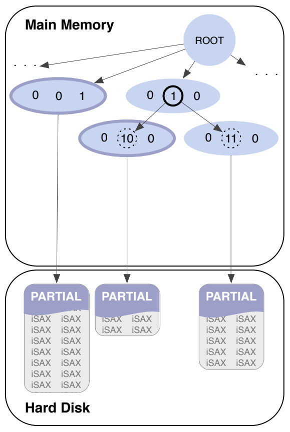

Figure 2: ADS+ initial state

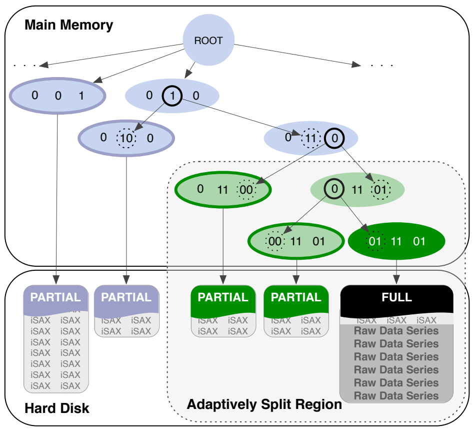

Figure 3: ADS+ adaptive split

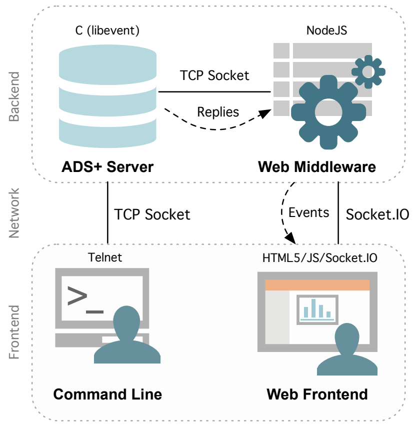

Figure 4: RINSE Architecture

iSAX 2.0 [3], where each data series is represented by words of multi-resolution characters (depicted as binary words in Figures 2 and 3). As more queries are posed, the index is continuously refined and subsequent queries enjoy even better execution times. ADS+ has been shown to work well for high-dimensional data series [17].

ADS+ introduces several novel techniques for adaptive data series indexing such as creating only a partial tree structure deep enough to not penalize the first queries with a lot of splits, and filling it on demand, as well as adapting leaf sizes on-the-fly and with varying leaf sizes across the index. During indexing, ADS+ performs only a few basic steps, mainly creating the basic skeleton of a tree, which contains condensed information on the input data series. Its leaves do not contain any raw data series and remain unmaterialized until relevant queries come. When queries arrive, ADS+ fetches data series from the raw data and moves only those data series inside the index. In addition, ADS+ does not require a fixed leaf size; it dynamically and adaptively adjusts the leaf size in hot areas of the index; all leaves start with a reasonably big size to guarantee fast indexing times but the more a given area is queried, the more the respective leaves are split into smaller ones to enhance query times. The net effect is that users do not have to wait for extended periods of time before getting access to the data: in our experiments with 1 billion data series, ADS+ allows users to answer several hundreds of thousands of queries, while the state-of-the art indexing approaches are still in the indexing phase [17]. Adaptive indexing was originally introduced in the context of column-store databases [11, 7, 8]. We share the main intuition: instead of building database indexes upfront, indexes are built during query processing, adapting to the workload. However, contrary to indexing relational data where a global ordering can be imposed, i.e., incrementally creating a range index, in the case of data series similarity search, global orderings do not exist.

Interactive Data Series Exploration. We describe the RINSE system (a recursive acronym: RINSE Interactive Series Explorer), which is built around ADS+. RINSE provides a user interface that manifests the benefits of adaptive indexing for interactive exploration of large data series collections. The exploration process can be seen in Figure 1. Users can explore large multi-gigabyte datasets in seconds, pose exact or approximate similarity queries by drawing data series using the mouse or touch screen and issue random queries on the click of a button. These queries guide the ADS+ index that lies inside RINSE to perform adaptive operations. Users can experience how the index adapts by looking at statistics that are updated on the fly. Additionally, they can compare query answering times, memory footprint, etc. across different access methods, such as a simple scan or the use of a complete (non adaptive) index. Finally, they can also experience the differences in terms of data-to-query time by building either a complete or an adaptive index. Users can pose queries using their mouse (or touch screen) or select them from other data collections.

The rest of the paper is organized as follows. In Section 2 we briefly describe how ADS+ works, in Section 3 we describe RINSE and the demonstration scenarios, and finally, in Section 4 we give a short summary of this work.

2. THE ADAPTIVE DATA SERIES INDEX

The ADS+ Index. In order to increase the exploration ability we need to decrease the data to query time. That is, we need to decrease the amount of time needed until a user can access and query new data with good response time. The main bottleneck is the index construction overhead. Our results in [17] show that a big part of these read and write costs is due to reading the raw data series from disk and to writing the leaves of the index tree back to disk (after insertions). ADS+ attacks the index construction bottleneck by shifting the construction of the leaf nodes of the index (the only nodes that can carry raw values for the data series, and have to be stored on disk) to query time.

ADS+ Index Creation. The index creation phase takes place before queries can be processed but it is kept very lightweight. ADS+ builds a minimal tree during this phase, which can be seen in Figure 2. The tree contains only iSAX representations, which are sufficient to build the index tree. The actual data series are only necessary during query time, i.e., in order to give a correct answer. In addition, not all data series are needed to answer a particular set of queries. In this way, ADS+ first creates all necessary iSAX representations (with a full scan on the raw file) and builds the index tree without inserting any data series. For data series we also record its offset in the raw data file so future queries can easily retrieve the raw values. ADS+ improves locality when inserting data using buffering.

Querying and Refining ADS+. When a query arrives (in the form of a data series), it is first converted to an iSAX

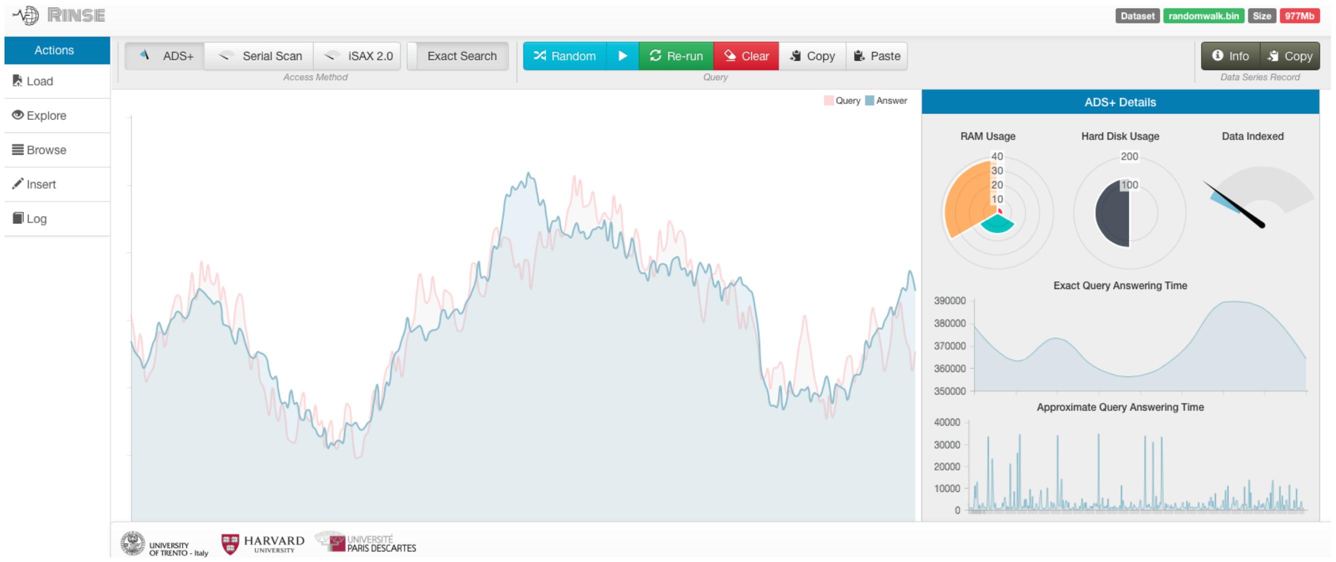

Figure 5: RINSE in action

representation. Then, the index tree is traversed searching for a leaf with an iSAX representation similar to that of the query. In the case that the leaf node where the search ends is in PARTIAL mode (as seen in Figure 2), i.e., it contains only iSAX representations but not any data series, then all missing data series are fetched from the raw file. At this point the leaf data is fully materialized and future queries that need to access the data series can find them in this leaf. The real distance from the query using the raw data is calculated. The minimum distance found in the leaf can be used as an approximate answer. If we need an exact answer, the process is repeated for the next most promising node until we can't improve our answer any more. To prune the search space, we use the minimum distance bounding function of iSAX, described in [15].

Adaptive Leaf Size. During index construction split operations are expensive as they cause data transfer to and from disk (to update node data). The main parameter that affects split costs is the leaf size, i.e., a tree with a big leaf size has a smaller number of nodes overall, causing less splits. Thus, a big leaf size reduces index creation time. However, when reaching a big leaf during a search, we have to scan more data series than with a small leaf. State-of-the-art indexes rely on a fixed leaf size which needs to be set upfront. To further optimize the data to query time, we introduce a more transparent initialization step. ADS+ uses two different leaf sizes: a big build-time leaf size for optimal index construction, and a small query-time leaf size for optimal access costs. Intuitively the effect is that when a target leaf is accessed during query answering, it is split until it becomes small enough, while all leaves created due to split actions but are not needed for this query are then left untouched. If and only if the workload shifts and future queries need to query those leaves, then ADS+ automatically splits those leaves even further to reach a leaf size that gives good query processing times (see Figure 3).

3. RINSE DEMONSTRATION

We now describe RINSE, and the demonstration scenarios that showcase the functionality and benefits of adaptive indexing via the ADS+ index.

3.1 The RINSE System

The overall 3-tiered architecture of the RINSE demonstration system can be seen in Figure 4. More information can be found on the ADS+/RINSE website[1].

Basic Infrastructure. We developed ADS+ in C and compiled it using GCC 4.6.3 as a shared library (*libads*). We additionally created a TCP server, using *libevent*, to expose its functionality as a network service. This is seen as the ADS+ Server in Figure 4. In addition, we created language bindings for *libads* for the NodeJS JavaScript runtime environment. Users connect to the ADS+ Server using a telnet client or a web interface. The RINSE web interface is developed as a single page application in HTML5, JavaScript and CSS. It connects to a NodeJS middleware using the Socket.IO library. The middleware has an always active connection to the ADS+ Server. The HTML5 client listens for user events which are pushed to the middleware. The middleware then pushes results back to the client in a real-time event-based mode. This allows for an intuitive and responsive experience where users can draw queries on screen using the mouse (or a touch interface), and see the results appear on screen in near real-time. They can also generate random queries, or choose queries from a list. Data series query workloads [18] can also be used, in order to stress-test ADS+, and demonstrate its performance benefits.

Supported Features. RINSE allows users to index data and issue nearest neighbor queries, using three different access methods: 1) a simple serial scan, which reads the complete raw data file for every query; also employing a simple early abandoning technique for avoiding useless computations, 2) a complete iSAX 2.0 [3] index, which is built before the users can start posing queries, and 3) an adaptive ADS+ index, which takes considerably less time to construct. In all three cases, the distance measure used is the Euclidean Distance. Each query can be re-run using a different data structure such that users can see the differences. Furthermore, users have access to statistics measuring the performance of each access method, the amount of data currently ingested by the adaptive index and the memory footprint.

[1]http://daslab.seas.harvard.edu/rinse/

3.2 Demonstration Scenarios

We will provide a laptop with RINSE installed that exposes the web interface. A screenshot of the interface is shown in Figure 5. Various datasets of different sizes will also be made available to the users for exploration. The goal is to discover the speed benefits of using partial adaptive indexes over traditional ones for similarity search in large collections of data series, and observe how the system adapts to their queries during data exploration. Bellow we list 3 demo scenarios that showcase the functionality of RINSE.

1. Immediate Data Access. In our first demonstration scenario, the participants will experience how adaptive indexing allows for quick access to data by not having to wait for an extended period of time for indexing to be completed. We achieve this by directly comparing adaptive indexing to full indexing. The participants can choose between the two methods and witness how the indexing evolves by observing a set of dynamic graphic representations that record the behavior of the indexing structures. By repeating the process with increasing data sizes, it becomes evident that adaptive indexing is a scalable approach, while full indexing quickly becomes a bottleneck.

2. Data Exploration. In the second demonstration scenario, the audience will be given the ability to explore large datasets at interactive speeds. The goal of this scenario is to showcase the ability of ADS+ to provide answers at interactive speeds even though it starts from a partial index state compared to full indexing. Participants can issue queries in multiple ways such as interactively drawing data series on screen, as seen in the large plot area on the left hand side of Figure 5. For this task one can either use the mouse or a touch screen (e.g., on an iPad). A query is essentially a data series and the system tries to find the closest data series in the database. Additionally, queries can be selected from within existing files of data series; in this scenario the user browses existing files and selects a specific data series to see what is the closest one in the database. The result to each query, i.e., the nearest neighbor, is displayed on screen alongside the query itself as shown in Figure 5. This allows users to visually compare the shapes of the two data series. To demonstrate the benefits of using an adaptive index, participants can compare the query answering times of ADS+ [17] to that of iSAX 2.0 [3], and serial scan. Furthermore, we allow users to inspect the evolution of query answering times (i.e., for a sequence of queries) as seen at the bottom right in Figure 5. For iSAX 2.0 and ADS+, users are able to run both exact and approximate queries by selecting this option from the RINSE interface. In this way, users can also compare the answering times and accuracy of the various methods when using exact versus approximate processing modes.

3. Adaptivity Benefits. In this scenario, the participants will experience the adaptive nature of ADS+. In particular, we highlight how ADS+ grows incrementally and adaptively as more queries arrive. To achieve this we provide a visual way of monitoring various statistics such as RAM and disk usage. Users can also see the percentage of data indexed by ADS+ at any point in time. In addition, they can observe the index expand as more queries are issued, and also observe the impact that this has on memory usage. Breakdowns for partial and raw data in the index are also provided. All these measures can be seen in Figure 5 on the right hand side. Finally, we provide both a manual and an automatic query process: (a) in the manual case, the index is enriched through user queries; (b) in the automatic case, the system automatically executes random queries and enriches the index. To support this scenario, a Play/Pause button is additionally provided to control the automatic query execution. While the system continuously executes random queries, the user can observe the changing index characteristics by following the evolution of the statistics reported graphically, i.e., memory overhead, percentage of data indexed, and query answering times.

4. CONCLUSIONS

In this demo, we demonstrate the benefits of the adaptive data series index ADS+ [17] through RINSE, a tool that provides visual access and querying of large data series collections. This demo allows the audience to experience how adaptive indexing provides quick access to data.

References

[1] R. Agrawal, C. Faloutsos, and A. N. Swami. Efficient similarity search in sequence databases. In *FODO*, 1993.

[2] S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-tree: An index structure for high-dimensional data. In *VLDB*, 1996.

[3] A. Camerra, T. Palpanas, J. Shieh, and E. Keogh. iSAX 2.0: Indexing and mining one billion time series. In *ICDM*, 2010.

[4] A. Camerra, J. Shieh, T. Palpanas, T. Rakthanmanon, and E. Keogh. Beyond One Billion Time Series: Indexing and Mining Very Large Time Series Collections with iSAX2+. *KAIS*, 39(1):123–151, 2014.

[5] K.-P. Chan and A.-C. Fu. Efficient time series matching by wavelets. In *ICDE*, 1999.

[6] A. Guttman. R-Trees A Dynamic Structure for Spatial Searching. In *SIGMOD*, 1984.

[7] F. Halim, S. Idreos, P. Karras, and R. H. C. Yap. Stochastic database cracking: Towards robust adaptive indexing in main-memory column-stores. *PVLDB*, 5(6):502–513, 2012.

[8] S. Idreos. Database Cracking: Towards Auto-tuning Database Kernels. PhD Thesis, 2010.

[9] S. Idreos, I. Alagiannis, R. Johnson, and A. Ailamaki. Here are my Data Files. Here are my Queries. Where are my Results? In *CIDR*, 2011.

[10] S. Idreos and E. Liarou. dbtouch: Analytics at your fingertips. In *CIDR*, 2013.

[11] S. Idreos, S. Manegold, H. Kuno, and G. Graefe. Merging what's cracked, cracking what's merged: Adaptive indexing in main-memory column-stores. *PVLDB*, 4(9):586–597, 2011.

[12] S. Idreos, O. Papaemmanouil, and S. Chaudhuri. Overview of Data Exploration Techniques. In *SIGMOD, Tutorial*, 2015.

[13] E. Keogh, K. Chakrabarti, and M. Pazzani. Dimensionality reduction for fast similarity search in large time series databases. *KAIS*, 3(3):263–286, 2000.

[14] J. Lin, E. Keogh, and S. Lonardi. A symbolic representation of time series, with implications for streaming algorithms. In *DMKD Workshop*, 2003.

[15] J. Shieh and E. Keogh. iSAX: Indexing and Mining Terabyte Sized Time Series. In *SIGKDD*, 2008.

[16] Y. Wang, P. Wang, J. Pei, W. Wang, and S. Huang. A data-adaptive and dynamic segmentation index for whole matching on time series. *PVLDB*, 6(10):793–804, 2013.

[17] K. Zoumpatianos, S. Idreos, and T. Palpanas. Indexing for interactive exploration of big data series. In *SIGMOD*, 2014.

[18] K. Zoumpatianos, Y. Lou, T. Palpanas, and J. Gehrke. Query workloads for data series indexes. In *SIGKDD*, 2015.