Interactive Search and Exploration of Waveform Data with Searchlight

Alexander Kalinin
Brown University
akalinin@cs.brown.edu

Ugur Cetintemel
Brown University
ugur@cs.brown.edu

Stan Zdonik
Brown University
sbz@cs.brown.edu

ABSTRACT

Searchlight [9] enables search and exploration of large, multi-dimensional data sets interactively. It allows users to explore by specifying rich constraints for the “objects” they are interested in identifying. Constraints can express a variety of properties, including a shape of the object (e.g., a waveform interval of length 10-100ms), its aggregate properties (e.g., the average amplitude of the signal over the interval is greater than 10), and similarity to another object (e.g., the distance between the interval’s waveform and the query waveform is less than 5). Searchlight allows users to specify an arbitrary number of such constraints, with mixing different types of constraints in the same query.

Searchlight enhances the query execution engine of an array DBMS (currently SciDB) with the ability to perform sophisticated search using the power of Constraint Programming (CP). This allows an existing CP solver from OrTools (an open-source suite of operations research tools from Google) to directly access data inside the DBMS without the need to extract and transform it.

This demo will illustrate the rich search and exploration capabilities of Searchlight, and its innovative technical features, by using the real-world MIMIC II data set, which contains waveform data for multi-parameter recordings of ICU patients, such as ABP (Arterial Blood Pressure) and ECG (electrocardiogram). Users will be able to search for interesting waveform intervals by specifying aggregate properties of the corresponding signals. In addition, they will be able to search for intervals similar to already found, where similarity is defined as a distance between the signal sequences.

Categories and Subject Descriptors

H.2.4 [Database Management]: Systems—*query processing*

Keywords

Data exploration; Query processing; Constraint Programming; Array Databases; Interactive Results

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SIGMOD’16, June 26–July 01, 2016, San Francisco, CA, USA

© 2016 ACM. ISBN 978-1-4503-3531-7/16/06...\$15.00

DOI: <http://dx.doi.org/10.1145/2882903.2899404>

1. INTRODUCTION

In the era of big data, when the variety and volume of available data increases at a considerable pace, the need for ad-hoc analysis is becoming more and more evident. On the one hand, users often want to discover new knowledge hidden in the data, like interesting patterns or dependencies between attributes. For this, machine learning techniques have been used with great success in the recent years. On the other hand, users often have some idea about *what* they want to find, but have no idea *where* to look. The latter is clearly a *search* problem. An example might include a scientist who has a hypothesis and is looking for evidence to support it. In such a case she might want to find objects with specific properties, where the nature of the object and the properties depend on the area of research. While it might be possible to explore every possible object in the data and filter the required ones, this would certainly be infeasible in most common cases due to large search space size. The problem quickly gets out of hand when the user wants to try different properties or combinations of them, which is common for ad-hoc exploration. Moreover, the user might be more interested in getting *some* results quickly instead of *all* results after a considerable amount of time. Such interactivity is often unavailable in traditional DBMSs or might be hard to achieve for search queries due to their poor support in such systems.

Exploration via search queries. To give a more concrete example, let us assume a scientist working with the MIMIC database [2]. The data contains recordings of physiological signals from a number of ICU patients, including signal waveforms and accompanying metadata. The user might want to find waveform intervals satisfying some interesting properties, such as the average amplitude of the signal over the interval. Additionally, she might want to find intervals with waveforms similar to some pattern (possibly found via past queries). In this scenario the following types of queries are possible:

- *Basic search queries* search for an interval satisfying specified properties. For example, “find a time interval for patient 1 of length 10ms to 100ms where the average amplitude of the ABP (Arterial Blood Pressure) signal is below 20”.
- *Complex search queries* involve multiple intervals and constraints tying them together. For example, the user might add this constraint to the previous query: “the difference between the maximum amplitude of the signal over the current interval and its left and right 100ms neighborhoods is greater than 10”.

- *Similarity queries* involve searching for waveform intervals similar to the specified query waveform. It might have been obtained as a result of one of the previous search queries or found by looking at the data. In this case the user might ask for “waveform intervals with the distance from the query interval less than 20”, where the distance is defined as Euclidean distance between the corresponding sequences of signal values.

In all these cases, the user should start getting results as quickly as possible. The user might not even be interested in the final result at all. She might quickly switch between queries, interrupt, refine and restart them.

Existing solutions. Traditional DBMSs have very limited support for exploration via search queries. For relational DBMSs there is a framework called Semantic Windows (SW) [8], which supports interactive execution of a limited subset of “Basic search queries” from the example above. Users can search for multidimensional regions satisfying common aggregate properties. SW does not provide any support for more complex constraints. When it comes to array DBMSs, the only option seems to use built-in analytical query operators or write complex scripts, combining several queries. For instance, SciDB has the `window()` operator, which computes aggregates for every possible sub-array of the specified size. This might allow users to find interesting sub-arrays. However, the operator processes sub-arrays sequentially, in the specific order, without any interactivity. Moreover, since it has to compute every possible sub-array, it is infeasible for large search spaces. For more complex search problems, containing multiple constraints, users would have to write several queries and perform some form of a “join” or concatenation between intermediate results.

Searchlight. *Searchlight* [9] enables users to express a variety of search problems written as constraint programs. Users can use traditional constraints and search heuristics available in CP systems. Additionally, they can define their own, which is essential for bringing search to different exploration domains. Under the hood, Searchlight efficiently connects the solver with the DBMS query execution engine, allowing the former to enjoy the benefits of such features as data buffering, access to DBMS query operators and efficient management of data. The search can run in the distributed environment, where multiple solvers work in parallel on different parts of the search space. Searchlight makes efficient use of the data distribution and networking capabilities present in distributed DBMSs.

Searchlight is implemented as a fusion between Google’s Or-Tools [1], a suite of operations research tools, which contains a powerful CP solver, and SciDB [5, 3], an array DBMS. We demonstrate the practicality of our approach using the real-world MIMIC II [2] data set, allowing users to search for interesting waveform intervals defined by constraints, including complex neighborhood constraints and similarity.

2. SEARCHLIGHT OVERVIEW

In this section we briefly describe Searchlight, including its query model and system architecture. More thorough discussion of these topics, as well as the experimental evaluation, can be found in the Searchlight research paper [9].

2.1 Search Queries

Since Searchlight uses a CP solver to perform search, it takes queries in form of constraint programs. A query de-

Figure 1: Two-level search query processing: the solver on the first level, the validator on the second level. The router routes data accesses to the synopsis or the array, depending on the context.

finer a number of *decision variables* and *constraints* over the variables. Revisiting the example from the Introduction, the user might study ABP signal recordings for a particular patient. If she wants to find an interval of length from 10ms to 100ms, she can specify two variables: $x \in [0, 3600000]$, denoting the start of the interval and $lx \in [10, 100]$, denoting the interval’s length. The variables have *domains* associated with them, which define the search space. In this case we assume the patient’s recording starts at time 0 and lasts for 1 hour. Then the user can specify the following constraints: $x + lx - 1 \leq 3600000$, which keeps the entire interval inside the 1 hour recording, and $avg(x, lx, ABP) < 20$, which specifies the restriction for the average amplitude of the signal. The $avg()$ function is a Searchlight extension, used to access the ABP data in the DBMS.

2.2 Searchlight Query Processing

An overview of the Searchlight query processing is presented in Figure 1. The gist of our approach is two-level processing. At the first level, *Solvers* receive a constraint program, create the search model (i.e., decision variables and constraints) and perform *speculative* search on a *synopsis* instead of real data. The synopsis is a condensed representation of the data, containing information needed for verifying constraints, e.g., aggregate properties of different areas of the data, bitmaps for sparse areas, etc. As the result, the Solvers produce *candidate solutions* (*candidates*). Searchlight guarantees the absence of false negatives, however, the candidates might contain false positives. At the second level, *Validators* check the candidates over the original data, producing the real solutions. One of the most important features of the speculative search is the ability to prune parts of the data without reading the data itself, which significantly improves query completion times. The two-level processing is completely transparent for both Solvers and users, and is taken care by the *Router* component, which directs the API data calls to the appropriate data. Solvers and Validators do not require any additional knowledge about the query beyond general knowledge contained in the model.

2.3 Distributed Searchlight

Searchlight supports distributed query processing on both levels of the execution. At the first level, the query’s search space is distributed among multiple Solvers on the same and different nodes, exploring both multi-core and multi-node parallelism. At the second level, Validators are assigned to different data partitions and are responsible for validating

candidates referencing the corresponding data. Validators can redistribute data, if needed. There can be an arbitrary number of Solvers and Validators, since the two levels of query processing are completely disjoint, and no Solver-Validator collocation is needed.

When Searchlight distributes a query across multiple machines in a cluster, there might be a natural bias on each level. Some Solvers might produce more candidates than others, if they hit a “hot-spot” part of the search space. In the same way, since different Validators are responsible for different data partitions, some of them might have to validate more candidates than others. Searchlight supports both static and dynamic balancing on both levels. During the static phase the search space and the data is distributed between Solvers and Validators before the query begins. During the query execution Searchlight redistributes work between idle and busy Solvers to take care of the search bias. If some of the Validators become struggling with candidates and fall behind, Searchlight starts more CPU threads to help with the load. If increasing the number of threads does not help, Searchlight redistributes candidate solutions to idle Validators with possible data redistribution as the last resort. Since, both Solvers and Validators might use multiple threads Searchlight mediates resource utilization depending on the state of the query execution.

3. MIMIC DATA AND QUERIES

Searchlight uses SciDB [3, 5], an open-source array DBMS, as its underlying DBMS. An array is a collection of elements indexed by *dimensions*. Each element has its own tuple of *attributes*. We store the MIMIC II waveform data in a two-dimensional array indexed by the patient id and signal time. Each time point corresponds to 0.008s, since the measurements are taken with the frequency of 125Hz. The attributes represent signals stored for a particular patient at the particular time point. We store ABP (Arterial Blood Pressure), PAP (Pulmonary Artery Pressure) and ECG (electrocardiogram) for the purpose of the demonstration, although other signals can be easily added. If some signal was not recorded at the specified time for the specified patient, the value is considered *null* and is ignored during computations. Note, the original MIMIC data set might contain several recordings for a single patient. We have concatenated such recordings into a single timeline for each patient. To be able to find the original MIMIC record, we have an external index pointing to the original records.

Since Searchlight queries are constraint programs, let us discuss briefly how these programs might be composed for the MIMIC data. We use the following decision variables:

- *id* specifies the id of the patient. It can be set to the required value to explore a particular patient. Alternatively, it can be set to a suitable range.
- *x* defines the beginning of the resulting interval. *x* can be assigned a particular range, restricting the search to that range for each patient specified in the domain of *id*.
- *lx* denotes possible length of a resulting time interval. It can be set to a constant value to search for fixed length intervals, e.g., $lx = 125$ would look for 1 second intervals. Alternatively, it can be set to a range of suitable values, e.g., $lx \in [125, 1250]$ for 1-10 second intervals.

In addition, the user can use the following UDFs to access signal data in the DBMS:

- $avg/sum/count/min/max(id, x, lx, sig)$ compute the cor-

responding aggregate values for the specified signal and interval, e.g., $avg(id, x, lx, ABP)$ computes the average amplitude of the ABP signal over the specified interval.

- $sq_dist(id, x, lx, sig, seq)$, which computes the squared Euclidean distance between the *signal* values of the interval (id, x, lx) and the sequence *seq*. This function allows users to search for waveforms similar to the specified by constraining the distance to be less than some value.

To express more complex queries involving multiple intervals, more variables can be defined. For example, if there is need to compare the ABP signal over the interval with the average signal over its 1 second left neighborhood, this can be achieved via $avg(id, ln, 125, ABP)$, where $ln = x - 125$ is a new variable. Instead the constant 125 another variable can be defined, similar to *lx*, to express a range of neighborhood lengths. One of the main ideas behind Searchlight is generality: Searchlight does not restrict users in terms of variables and constraints.

Section 2.2 mentioned the synopsis structure used at the first level of query processing. For MIMIC we use two types of synopsis: aggregate and DFT (Discrete Fourier Transform). The first one is an aggregate grid, where the waveform of each patient is divided into disjoint sub-intervals (cells), and each cells stores aggregate information (min/max, sum, count). This synopsis is based on Multi-Resolution Aggregate (MRA) structures [10]. The DFT synopsis is used for the sq_dist function and stores MBRs (Minimum Bounding Rectangles) for DFT traces of waveforms. This synopsis is based on the ST-index for subsequence matching [6].

4. DEMONSTRATION

We demonstrate all three types of queries described in the Introduction. The users do not directly write constraint programs, but use the GUI presented in Figure 2. The GUI converts the user-specified parameters into the constraint program and sends it to Searchlight. The GUI allows users to perform search queries over waveforms of all the patients loaded in the system. The user can specify the required length of the interval (including a range), the signal (e.g., ABP), the range of average values for the amplitude and the search range to restrict the timeline. She is also able to specify the *resolution* of the search, which restricts intervals to be aligned with the specified value. This might significantly improve query times, since increasing the resolution parameter results in decreasing the search space.

Searchlight is an interactive system. It starts outputting results in a table as soon as it finds them. The user can click on any results to asynchronously request the corresponding waveform. At any point during the query execution the user can interrupt the query and issue a new one.

To demonstrate more complex constraints, the GUI also allows users to search for anomalies, defined as the difference in the maximum signal amplitudes between the intervals and its neighborhoods. The user is able to specify the lengths of the neighborhoods and the required minimum differences of the maximum amplitude values.

The GUI also supports similarity queries (not shown in the Figure). When the user explores a visualized waveform, she can highlight a portion of it and ask for intervals with similar waveforms by specifying the maximum allowed distance between the corresponding sequences of signal values.

To compare the performance of Searchlight with the native SciDB, the users are able to run the same search queries

Figure 2: Searchlight GUI screenshot for exploring MIMIC.

on another screen by specifying parameters for predefined query scripts. The scripts access SciDB via its Python API and issue a series of AQL (Array Query Language) queries corresponding to the main search query.

5. DISCUSSION

There has been a growing interest in interactive query processing and data exploration, which resulted in a number of systems. Some systems, such as Tableau [4] or Dice [7], allow users to perform *navigation-like* data exploration with interactive speeds. Such exploration usually involves queries similar to SQL group-bys, joins, ranges, etc. In contrast, Searchlight performs *search* queries, expressed as a number of constraints for the result. We believe these systems can complement each other, where the user can use Searchlight to find interesting regions in the data, and then continue further with the navigational exploration.

Other systems, such as Rinse [11], allow users to execute specific search queries, such as sub-sequence matching. Rinse uses adaptive index building techniques, which significantly reduces index building times and makes the data quickly available for querying after the import. Searchlight can certainly benefit from similar techniques, for example by employing adaptive synopsis building. With respect to queries, however, Searchlight supports a richer set of constraints, going beyond sub-sequence matching.

6. ACKNOWLEDGEMENTS

This research was supported in part by the Intel Science and Technology Center for Big Data, NSF IIS-1111423, and NSF IIS-1526639.

7. REFERENCES

- [1] Google or-tools. <https://code.google.com/p/or-tools/>.
- [2] Multiparameter intelligent monitoring in intensive care (mimic ii). <https://physionet.org/mimic2/>.
- [3] Scidb. <http://www.scidb.org/>.
- [4] Tableau. <http://www.tableau.com/>.
- [5] P. G. Brown. Overview of scidb: large scale array storage, processing and analysis. In *SIGMOD*, pages 963–968, 2010.
- [6] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *SIGMOD*, pages 419–429, 1994.
- [7] P. Jayachandran, K. Tunga, N. Kamat, and A. Nandi. Combining user interaction, speculative query execution and sampling in the dice system. *VLDB*, 7(13):1697–1700, 2014.
- [8] A. Kalinin, U. Cetintemel, and S. Zdonik. Interactive data exploration using semantic windows. In *SIGMOD*, pages 505–516, 2014.
- [9] A. Kalinin, U. Cetintemel, and S. Zdonik. Searchlight: Enabling integrated search and exploration over large multidimensional data. *VLDB*, 8(10):1094–1105, 2015.
- [10] I. Lazaridis and S. Mehrotra. Progressive approximate aggregate queries with a multi-resolution tree structure. In *SIGMOD*, pages 401–412, 2001.
- [11] K. Zoumpatianos, S. Idreos, and T. Palpanas. Rinse: Interactive data series exploration with ads+. *VLDB*, 8(12):1912–1915, 2015.