dbTouch: Analytics at your Fingertips

Stratos Idreos[†]
[†]CWI, Amsterdam
stratos.idreos@cwi.nl

Erietta Liarou[⋆]
[⋆]EPFL, Lausanne
erietta.liarou@epfl.ch

ABSTRACT

As we enter the era of data deluge, turning data into knowledge has become the major challenge across most sciences and businesses that deal with data. In addition, as we increase our ability to create data, more and more people are confronted with data management problems on a daily basis for numerous aspects of every day life. A fundamental need is *data exploration through interactive tools*, i.e., being able to quickly and effortlessly determine data and patterns of interest. However, modern database systems have not been designed with data exploration and usability in mind; they require users with expert knowledge and skills, while they react in a strict and monolithic way to every user request, resulting in correct answers but slow response times.

In this paper, we introduce the vision of a new generation of data management systems, called *dbTouch*; our vision is to enable interactive and intuitive data exploration via *database kernels which are tailored for touch-based exploration.* No expert knowledge is needed. Data is represented in a visual format, e.g., a column shape for an attribute or a fat rectangle shape for a table, while users can touch those shapes and interact/query with gestures as opposed to firing complex SQL queries. The system does not try to consume all data; instead it analyzes only parts of the data at a time, continuously refining the answers and continuously reacting to user input. Every single touch on a data object can be seen as a request to run an operator or a collection of operators over part of the data. Users react to running results and continuously adjust the data exploration - they continuously determine the data to be processed next by adjusting the direction and speed of a gesture, i.e., a collection of touches; the database system does not have control on the data flow anymore. We discuss the various benefits that dbTouch systems bring for data analytics as well as the new and unique challenges for database research in combination with touch interfaces. In addition, we provide an initial architecture, implementation and evaluation (and demo) of a dbTouch prototype over IOs for IPad.

6th Biennial Conference on Innovative Data Systems Research (CIDR '13) January 6-9, 2013, Asilomar, California, USA.

1. INTRODUCTION

The Big Data Era. Turning data into knowledge has become an essential need for businesses and sciences. As more data is generated, knowledge hides better within the "pile of big data". With several innovative ideas, systems and research directions, the database community is making strong steps towards dealing with the new challenges of big data across several data management areas. For example, recent highlights include research on databases that exploit the map reduce and the cloud paradigms [1, 12, 34, 2, 38, 32], database kernels for approximate query processing [3, 31, 40], adaptive indexing [26, 27, 28, 18, 16, 29], adaptive data loading [24, 4], high performance column-store [43, 8, 25] and hybrid database kernels [6, 30, 11, 17, 13], crowdsourcing [35, 15, 39, 14] and anthropocentric systems [44], exploitation of modern hardware [20, 10, 36, 7], usability [37] and energy aware systems [21, 33]. Still though, the big data era poses several more unique challenges and opportunities.

Interactive Data Exploration. Here, we propose a novel research direction to address one of the most critical data management needs today, namely data exploration [9], i.e., when we are in search for interesting patterns often not knowing a priori exactly what we are looking for. For example, an astronomer wants to browse parts of the sky to look for interesting effects, while a data analyst of an IT business browses daily data of monitoring streams to figure out user behavior patterns. What both cases have in common is a daily stream of big data, i.e., in the order of multiple Terabytes and the need to observe *something interesting and useful.* The vision of interactive data exploration aims to allow for instant access to the data, i.e., without expensive initialization steps, while at the same time allowing the user to extract knowledge from the data by selectively and interactively investigating only parts of the data.

Vision: dbTouch. Our vision towards interactive data exploration is to bring together core database research and touch-interfaces, i.e., to enable users to *touch and manipulate data* in intuitive ways in search for interesting patterns. We capture the new category of such systems and research under the title *dbTouch* systems.

We propose to rethink database kernel designs towards architectures tailored for touch-based data exploration.

In a dbTouch system, data is visualized in various shapes and forms, while users can interactively explore the data

via touch input. For example, an attribute of a given table may be represented by a column shape. Users are able to apply gestures on this shape to get a feeling of the data stored in the column, i.e., to scan, to run aggregates on part of the data, or even to run complex queries.

The fundamental concepts of query, query plan and data flow are redefined. The user touch, the gesture evolution, i.e., the speed and the direction of the gesture, determine the data to be processed next and the actions that need to be performed. The system does not try to consume all data; instead, it analyzes only parts of the data at a time, continuously refining the answers and continuously reacting to user input. Every single user touch on a data object can be seen as a request to run an operator or a collection of operators over a part of the data. At the same time, as a gesture, i.e., a collection of touches, evolves and varies in direction and speed, users can determine the data they need to analyze, reacting to intermediate results as they appear on screen. The database system does not have control anymore on the data flow.

The vision is that through the integration of touch gestures and database kernels we enable new ways for users to have a *quick look and feel* of their data in a natural and interactive way. At the same time, new challenges arise for touch-oriented database architectures.

Touch Input. Touch interfaces have already had a tremendous impact at mobile computing. They literally transformed the industry during the past 5 years, starting with Apple's IOs in 2008 and following up with Android from Google, Windows 8 from Microsoft and webOS from Palm (now HP). Although some were skeptic during the initial stages, the immediate and wide adoption of mobile touch based platforms clearly shows the huge potential. The key element is the simplicity and the interactive nature of touch interfaces which result in two fundamental side-effects; (a) more people use and interact with touch interfaces, i.e., people that are not necessarily very familiar with other forms of computing and (b) new kinds of applications appear. This is exactly the same trend we want to achieve with dbTouch. Imagine the simplicity of sliding a finger over a rectangle shape, representing a table on a touch tablet, to scan or to run aggregates over the table instead of trying to figure out the schema and to write SQL queries in text mode.

From Gestures to Query Processing. Naturally, the dbTouch direction creates new research opportunities both in the area of database architectures and in the area of visualization. From a database researcher's point of view there are several fundamental questions to answer. Most critical questions have to do with how we translate touch gestures to database query processing algorithms/operators, i.e., how does dbTouch react in terms of storage and access patterns when we slide a finger or when we zoom-in with two fingers over a column or a table? What is the equivalent of a query? Several challenges arise. For example, when sliding a finger to scan or to run an aggregation over a column, users may choose to slide faster or slower or change the slide speed numerous times or even pause for some time, while they are observing the running results.

The interactive and continuously changing mode in dbTouch is drastically different than what state-of-the-art database systems support; it requires rethinking of core algorithms, while new concepts arise when it comes to interactive query processing. In traditional systems, once a query is posed, the database controls the data flow, i.e., it is in full control regarding which data it processes and in what order, such as to compute the result to the user query. In dbTouch, however, these concepts are blurred. In dbTouch, a query is a session of one ore more continuous gestures and the system needs to react to every touch, while the user is now in control of the data flow.

In addition, other than translating gestures to database algorithms, there are numerous optimization issues that appear in dbTouch. For example, when a user slides a finger over a column with a varying slide speed, then we would like to find a good way and timing to extrapolate the gesture movement and to efficiently access, prefetch and precompute the anticipated data to avoid stalling once the query session resumes or when it moves faster.

Exploiting dbTouch. A dbTouch system is primarily meant to work as an exploration tool, i.e., to speed up understanding of data. For example, a user may choose to load a small data sample directly on a tablet mobile dbTouch system or use the tablet as an interface to a cloud based dbTouch system over the complete data set.

Contributions. Our contributions are as follows.

1. We introduce the vision of dbTouch systems for interactive touch-based data exploration.
2. We introduce algorithms and functionalities for several basic gestures such as single finger slide to scan or to run aggregates, two fingers zoom-in to progressively get more detailed samples of data, rotate to switch the physical design from row-store to column-store, etc.
3. We identify several optimizations, challenges and opportunities in dbTouch systems such as storing and accessing data in several granularities, prefetching data and precomputing results during gestures, incrementally changing the storage layout, etc.
4. We provide the first implementation, evaluation and demo of an early prototype dbTouch system over IOs for IPad.

Outline. The rest of the paper is organized as follows. Section 2 discusses the dbTouch vision in more detail along with descriptions of an initial set of gestures and operators for data exploration as well as optimization issues and problems that arise when designing an early dbTouch prototype. Section 3 discusses implementation details and an early evaluation of a prototype over IOs for IPad. Then, Section 4 discusses new opportunities and research directions opened by the dbTouch vision. Section 5 discusses related work and Section 6 concludes the paper. Finally, Appendix A discusses a demo proposal.

2. DBTOUCH

This section gives more details on the dbTouch vision such as what is the desired behavior and functionality. We discuss several topics one has to address when designing dbTouch systems such as data and result visualization, mapping of touch to data, query processing via slide gestures, optimizations to enable quick reaction to touch input, data storage and access options and optimizations as well as formulation of complex queries.

2.1 Interactive Exploration

One of the main requirements when inspecting data is to get a quick feeling regarding the quality of the data and possible patterns and properties. Not necessarily all data is required at this stage, while at the same time this is not an exact science; it involves a lot of intuition from the analyst which is why many people claim that (big) data analysis is a kind of art [19]. In this way, the key goal of dbTouch is to assist users with *data exploration*.

In order to promote data exploration, a dbTouch system needs to provide an *interactive* feeling to the user, i.e., giving the illusion that the user is indeed "touching the data" in real time. This increases user satisfaction and ease of using the system. To achieve this goal, dbTouch actions should not be monolithic actions that consume big piles of data at a time, resulting in significant delays in response times. Instead, they should be *incremental* and *adaptive* actions that can give quick results back to the user. In turn, users react to those results and adjust their gestures accordingly, effectively promoting interactive data exploration through the continuous interaction of the user with base data and results produced.

The above goal of interactive data exploration requires a combination of both low level query processing actions and visualization techniques for properly visualizing data and intermediate results. Good query processing techniques improve the response times of the system when reacting to user requests (gestures), while good visualization techniques improve the response time of the user when reacting to results produced by the system.

2.2 Front-end

We begin our description of dbTouch systems by discussing the front-end part of the system, i.e, what users experience and what they expect when interacting with a dbTouch system. This is drastically different than the standard SQL interfaces and impacts design decisions for the underlying dbTouch kernel.

Data Objects. The main interaction of a user with a dbTouch system comes by interacting with *data objects*. Objects appear on the touch screen, while a user can apply gestures on these objects. For simplicity and ease of presentation, in the rest of this section, we assume one of the most straightforward visualization options for data objects, i.e., data appear in a relational-like way; tables are represented as (fat) rectangles and attributes are represented as columns. For example, Figure 1 shows several examples of column representations and gestures. Data objects are abstract representations of the data. For example, in Figure 1 we see that the actual data is not visible. A single column of

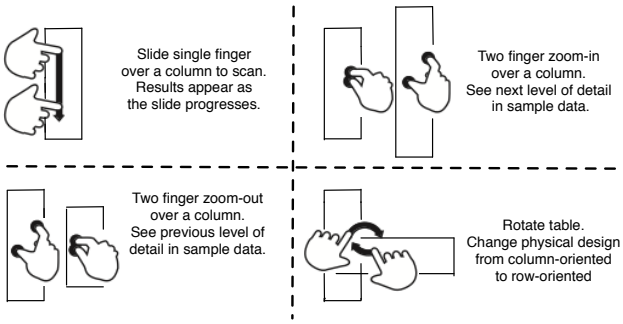

Figure 1: Examples of dbTouch gestures.

a height of only a few centimeters may represent an attribute in a table with several millions of tuples. The actual data become visible only during query processing, i.e., by applying one or more gestures on a data object. As we discuss later on, the fact that a small object represents many tuples has implications on data storage and data access options when designing a dbTouch kernel for data exploration.

dbTouch in Action. A snapshot of our dbTouch prototype in action is shown in Figure 2. In general, several objects may be visible at any time, representing data (columns and tables) stored in the database. The user has the option to touch and manipulate whole tables or to visualize and work on the columns of a table independently for more fine grained access and analysis. In the example of Figure 2, the user sees three columns of the same table, each one visualized as a separate rectangle and with a different color. In the right hand-side screen-shot of Figure 2 the user has applied a zoom-in gesture over the blue data object for a more fine grained exploration on this column.

Schema-less Querying. Contrary to traditional database systems, in a dbTouch system users do not need to know the database schema. The various data objects convey the schema information at a high level, giving a glimpse to the user regarding what kind of data is available for exploration; just by glancing at the touch screen, users can immediately know how many tables and columns exist. Applying query processing gestures is possible without any further knowledge. Discovering the schema in more detail is part of the exploration process. For example, a single tap anywhere on a column data object reveals a single column value, allowing to easily recognize the data type of the column. The exact value that appears depends on the exact location touched (to be discussed later on). Similarly, a single tap anywhere on a table data object reveals a full tuple, allowing to easily recognize what kind of attributes are contained in this table. As we discuss later on, users can break down tables into a several single columns or groups of columns, affecting the underlying storage. Other important properties such as attribute and table names as well as foreign key relationships between tables can be visualized on demand or permanently depending on user preference. Overall, contrary to traditional systems, exact schema information in a dbTouch system is less important to the user; one simply needs to look at the screen for a column or table which might be interesting and start touching it.

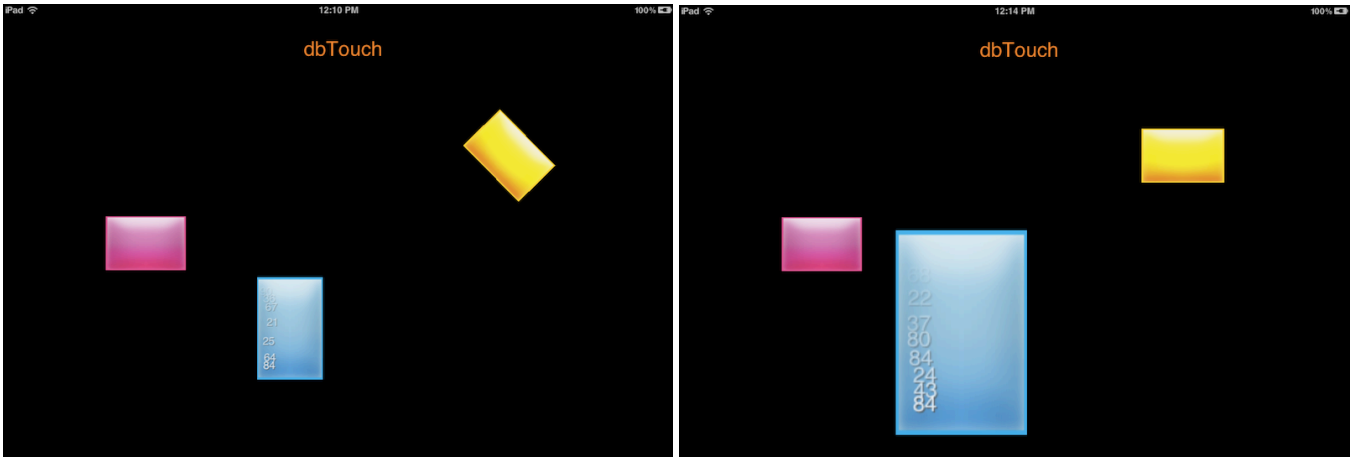

Figure 2: Screen-shot of dbTouch during an interactive scan via a slide gesture. Before (left side) and after (right side) a zoom-in gesture on the column represented by the blue object.

Visualizing Data. Naturally, the kind of visual representation used can vary. Here, we use a simple, relational-like, representation to introduce the dbTouch vision but other options are definitely worth studying and may better match certain data sets depending on the application. This issue is discussed more in Section 4.

2.3 Slide to Explore

Having discussed how data is visualized, we now continue to describe how one can start exploring the data via simple gestures.

Slide. The main query processing gesture in dbTouch is the *slide* gesture. It brings several opportunities as well as several optimization problems regarding both data storage and data access decisions when designing a dbTouch system. The upper left part of Figure 1 depicts an example of the slide gesture; the idea is that a user slides a single finger over a data object to *point* at data to be processed. In fact, a slide gesture is a collection of several single touch actions. dbTouch reacts to every single touch and processes data continuously as the gesture evolves. There are no restrictions on how long a slide gesture lasts or on what is the start and the end point of the gesture. In addition, there are not restrictions on what is the direction and the speed of the slide; these are all "query parameters" that the user may vary and adjust on-the-fly.

Scan and Aggregates. The most basic action when having a first exploratory look at a new set of data is achieved either by having access at the actual data values, i.e., by a plain scan, or by running a few simple aggregations. In both cases, dbTouch accesses base data and either delivers the actual data as is (in the case of a plain scan) or it computes a running aggregate and continuously updates this result (in the case that an aggregation query is chosen). Both queries are realized via a slide gesture over the data object the user is interested in. The slide speed, the data object size and the exact area touched determine the actual data of the underlying base column or table to be processed. More complex queries and considerations regarding data access and storage are discussed later on.

Query Processing. Contrary to traditional systems, querying dbTouch is not a monolithic action; not all data are processed in dbTouch, while query processing may stop at any time. The points touched by the user define the data to be processed. Users do not declare a query, then pose it and then wait for an answer. Instead, users define the query they wish to run by choosing a few query actions (say an scan or an aggregate for simplicity) and then they start a slide gesture over a column or a table.

In analogy with traditional database kernels, the slide gesture is equivalent to the *next* operation where an operator requests the next tuple to process. The difference is that instead of having a predefined plan on how to go through the data, now the users are the ones who trigger those *next* actions. They do so at a varying speed and with the option to go back and forth over the same data areas or even pause and begin again from an arbitrary data entry; now users have full control on the data flow of what would be the equivalent of a query plan in a traditional system.

Slide Example. Figure 2 shows a slide gesture in action in our dbTouch prototype. The gesture is applied on the blue data object in Figure 2 and results appear as the gesture progresses. The screen-shot is taken directly from the IPad where we tested our prototype using the screen-shot functionality of XCode. Naturally, the actual user finger is not visible in Figure 2; in this case the user slides a finger starting from the top of the blue object all the way to the bottom of the object with a single continuous movement.

Inspecting Results. A key element is how results are delivered to the user. Figure 2 depicts an example where results appear as the user touches the data object. Results appear *in place*, i.e., as if every single result value pops up from the position in the data object where the raw value responsible for this result lies. In fact, result values are typically shifted slightly sideways from the exact touch location such as to avoid being hidden below the user finger. Soon after a result value becomes visible, it subsequently fades away, making room for more results to be inspected. For example, in the screen-shot of Figure 2, the result values

which appear at the upper part of the blue data object are almost faded out; the effect is most visible to the upper most data values. This is because the slide gesture in this particular example began from the upper part of the blue object and continued to the bottom part. In this way, the most recently touched data entry is responsible for the most bold result value visible. With results appearing and disappearing in place and dynamically as the gesture evolves, dbTouch gives the feeling of interactive exploration, i.e., only data entries which are touched are being processed and only when touched; this is fully under the control of the user.

Challenges. There are several issues regarding how to design and optimize a dbTouch system with a strong exploration behavior; How should data be stored and accessed? What happens when the slide patterns such as speed and directions change? Which data tuples exactly do we process with every touch? How to maintain quick response times? The rest of this section discusses initial ideas for those issues and brings up open research problems for the efficient design of dbTouch kernels.

2.4 From Touch to Tuple Identifiers

A key step in a dbTouch system is in translating the location of a touch over a data object to a tuple identifier of the table or the column represented by the object, i.e., determining which data entry corresponds to the touch and thus which data entry should be processed next. For ease of presentation, consider a simplification of the slide gesture, i.e., a single tap gesture. With a single tap over a data object, dbTouch accesses a single data entry, i.e., in the case of a plain scan query, a single result value appears and fades away (as in Figure 2). A slide gesture can be seen as multiple continuous single taps in successive positions of a data object.

Object Views. In order to translate the location of a touch to a tuple identifier, dbTouch exploits the *view* concept of modern touch-based operating systems. Views are placeholders for visual objects. In turn, each view can be placed in a master view, forming hierarchies. Each view has a set of properties associated with it which are readily accessible by the touch OS, such as the size of the view, the location of the view within its master view, what kind of gestures are allowed over the view, etc. Within each view, visual objects and gestures can be treated independently to the master view. The touch OS allows for exact identification of the location where a touch appears within a view.

Mapping a Touch to a RowID. In dbTouch, each data object visualized corresponds to a different view. dbTouch adds a number of properties to each view, e.g., the number of data entries in the underlying column or table, the data type(s), the data size, etc. In this way, once a touch is detected in a given view corresponding to a dbTouch data object, dbTouch determines the location of the touch within the corresponding view. Subsequently, it determines the respective tuple identifier using the knowledge of the current size of the view and the knowledge of the total number of tuples in the actual data object; once we know the relative location based on the current view size, we can calculate the relative location (i.e., the tuple identifier) within the actual data object by applying the Rule of Three. Thus, if the touch location is t, the size of the data object is o and the number of total tuples is n, then the tuple identifier we are looking for is $id = n * t/o$.

When a data object refers to a single column, then only the height dimension is used to determine tuple identifiers; this is both for the touch location and for the view size. When a data object refers to a full table or to a collection of columns of a given table, then often both dimensions are needed. For example, with a vertical slide over a full table the user would see a similar result as in the example of Figure 2; the difference is that now dbTouch returns tuple entries as opposed to single column entries. In this case only the height dimension is needed. With a horizontal slide, on the other hand, we use both dimensions to slide through the attributes values of a given tuple entry; the tuple identifier is determined via the height, while the attribute seen is determined by the relative width of the touch location within the view. In addition, if a data object is rotated such as it lies horizontally, then a horizontal slide is used to scan through the data. However, nothing changes regarding the mapping of touches to tuple identifiers; when we rotate an object, then we only change its positioning within its master view; thus touches and identifiers calculated relative to the object view are not affected.

2.5 Data Access and Touch Granularity

We now proceed to discuss in more detail how data is accessed when applying gestures.

Touch to Explore. The goal of data exploration is not a complete view of the data, rather a quick glance at the data and a quick discovery of possible interesting patterns. The main idea is to guide users to quickly and easily figure out interesting data areas. Thus, running scans and queries over complete columns and tables brings slow response times which hinder exploration.

Touching Samples. In this way, a slide gesture in dbTouch results in only a *sample* of the data being processed even if a user slides through the whole area of a data object. The sample is distributed over the complete data set depending on how the slide gesture progresses, i.e., the touch locations registered determine the data to be processed. Every single data object visualized as an object of a few centimeters may correspond to data of say several million entries. However, for each possible size of a visual object, there is a limited amount of touch locations which can be registered and thus there is a limited amount of tuples which can be mapped and processed. These limitations are there purely due to physical constraints (e.g., finger and object size). Of course, a dbTouch system can give the option to the user to on demand vary the touch granularity, i.e., how many tuples correspond to each touch, as well as to change the size of the object. In addition, dbTouch systems need to take into account the above considerations when deciding how to store and how to access data. We discuss these issues and options later on.

Exploration Speed. Assuming for simplicity slide gestures that start from one end of a data object and go with a steady speed all the way to the opposite end, then the data entries processed are equally distributed over the complete

data set. A faster slide results in fewer tuples processed as less touch inputs are registered. On the contrary, a slower slide over the same data object results in more tuples accessed; this is because more touches are recognized and more successive positions can be mapped to tuple identifiers of the underlying data. In this way, the slide speed determines the granularity of the data observed.

In addition, there are no restrictions to how a gesture progresses. For example, after starting with a given speed, users may pause the gesture, e.g., because an interesting value is observed. Then, they can go back and forth in this data area at a much slower speed to observe the data at a more fine grained granularity. Overall the user has the freedom to "walk" over the data in any direction and speed. This is drastically different to how a typical database system processes data, i.e., one tuple after the other in the order they are stored in base data. dbTouch systems need to follow the user touch which might mean that some data entries are skipped during query processing but there is no guarantee that the user will not decide to go back and request those entries a few seconds later. This has implications to both how we should store and how we should access data in a dbTouch system. This issue is discussed in more detail in the next sections.

Zoom-in/Zoom-out. As we discussed, a given object size restricts the amount of data a user can access. Users may change the size of an object using the zoom-in and the zoom-out gestures. This allows to increase and to decrease respectively the granularity of the data processed from a given data object, i.e., it can be seen as accessing varying samples of the same data. Figure 1 depicts examples of those gestures. With a zoom-in gesture the respective data object becomes bigger. As a result, there is more area to touch and thus a more fine grained access to the underlying data; with a bigger visual size, there is a bigger number of positions to map to tuple identifiers. For example, the right hand-side screen-shot in Figure 2 depicts a slide over the blue column after a zoom-in operation; more data results appear compared to the slide in the left hand-side screen-shot in Figure 2. The zoom-out gesture works in exactly the opposite way, resulting to a smaller data object and thus to a more high level view of the data when applying slide gestures. Zoom-in and zoom-out gestures can be used in combination with slide gestures of a varying speed for significant flexibility when exploring data, allowing the user to drill down on selected data areas adaptively and on demand.

2.6 Storing and Accessing Data

We continue the dbTouch introduction with a discussion on ideas regarding how data should be stored.

Gesture Evolution. A key element in dbTouch is how we scroll through the data. Underlying storage and access methods need to be able to adjust to gesture patterns and variations. The goal is to improve the data access time a dbTouch system needs in order to react to a single touch. The progression of the user gestures defines how fast we go from one value entry to the next; users may change the slide speed over time, they may change the direction of the slide or they may even pause the gesture temporarily. In addition, there are no restrictions regarding the start and the end point of a slide gesture. Similarly, there is no restriction regarding how many times a single gesture may go over the same area of a data object.

All the parameters above call for highly adaptive methods for storing and accessing data in order to be able to cope with the numerous variations in gesture characteristics; any data may be accessed at any time.

Physical Layout. dbTouch does not pose any particular restrictions on the underlying storage model. It can be row-store, column-store or a hybrid format. One useful design decision we use in our current prototype is that fields are fixed-width per attribute. This technique has been pioneered in modern column-stores and is currently applied to hybrid systems as well. Essentially, using fixed-width values per attribute allows for a much easier calculation of data locations without having to access metadata information as it happens with more traditional slotted pages layouts. In dbTouch, fixed-width fields allow for a faster mapping of touch locations to tuple identifiers. In this way, the underlying storage layout used in our current dbTouch is matrixes. Each matrix may contain one or more columns and each column contains fixed-width fields. The matrixes are dense and each matrix is associated with a given data object.

Sample-based Storage. As we discussed, query processing in dbTouch via slide gestures is equivalent to processing a sample of the underlying data. Even when a slide touches the whole area of the target data object, the size of the object and the speed of the gesture progression result in a given sample factor. In a traditional system, data is stored in a single base column or table. However, in dbTouch, accessing data at a coarse granularity directly from the base data may result in loading a significant amount of data which is not needed for the current query. As a result performance is hindered and the user does not get all the performance benefits of not touching the whole data set.

A better approach would be to store separately various different samples of the base data and depending on the object size and gesture speed feed from the proper copy, minimizing the auxiliary data reads. Similar ideas have appeared in Sciborg [40] about storing hierarchies of samples. In dbTouch, however, the problem becomes more complex as, given the potential continuous variation of the slide parameters, dynamically choosing and switching between various copies of the data becomes a critical issue.

Prefetching Data. An interesting optimization is that of *prefetching* when a slide gesture pauses or slows down. dbTouch can extrapolate the gesture progression (speed and direction) and fetch the expected entries such that they are readily available if the gesture resumes. Prefetching is more important when a slide gesture is used for aggregation operations where more computation is involved or of course if the slide gesture is used for more complex queries involving several operators and data objects (to be discussed later on).

Caching Data. In addition, *caching* can be exploited such that dbTouch is ready if the user decides to re-examine a data area already seen. dbTouch needs to observe the gesture patterns and adjust the caching policy according to the

expected progression of the gesture. In addition, caching may be used in order to create a new copy (sample) of the data which will allow dbTouch to answer future queries requesting data at a similar granularity.

Indexing. Creating and exploiting indices for dbTouch systems is an exciting topic. When querying an indexed column or sets of columns, then the slide gesture becomes the equivalent of an index scan. Having a hierarchy of samples directly affects indexing decisions; for example, dbTouch can maintain a separate index for each sample level, treating each copy separately depending on how often index support is needed for this copy. Similarly, switching between the various indexes during a single slide of a varying speed becomes a challenging issue.

2.7 Interactive Summaries

A more advanced use of the slide gesture in dbTouch is the exploitation of *interactive summaries*. The idea is that when sliding through a data object, dbTouch returns a summary of x items as opposed to simply returning a single data entry which corresponds to the exact touch location. The summary is defined as an aggregate value of several consecutive data entries. When during a slide we register position p which corresponds to tuple identifier id_p, then dbTouch scans all entries within the tuple identifier range $[id_p - k, id_p + k]$ and calculates a single aggregate value. Parameter k can be defined by the users according to their exploration requirements as well as by system parameters. For example, when fetching a given data item, we would like to also exploit all items within the same cache line. Similarly, the kind of aggregation used can be defined by the user. A good default choice is to perform an *average* aggregation.

Essentially, interactive summaries open two opportunities for data exploration. First, they allow users to "touch" and inspect more data with each single touch. Second, they allow for a quick inspection of properties and patterns on small/controlled groups of data, while also observing differences in patterns across different data areas of the same object on demand.

2.8 Schema and Storage Layout Gestures

Part of the exploration process involves changing the schema or the layout of the data for organization or for performance reasons. With the data visualized as objects in a touch screen, users can move data around using pan gestures and group or ungroup specific data objects. For example, one can create a table by drag and drop actions in a table placeholder object where independent columns can be dropped.

More interestingly, dbTouch allows users to vary the layout of individual tables using the rotate gesture (see Figure 1). Rotating a row-oriented table changes its physical layout to a column-store structure by projecting all attributes to individual arrays (and vice versa). The effects of the rotation gesture are also applied by simply rotating the tablet device itself for an even more interactive experience. Similarly, a user can project a specific column out of a fat table by dragging the column out of the table. It can be sent to an individual array or simply to a smaller table allowing the user to experience faster response times by going only through the needed data. The net result is that the dbTouch user can take control over performance parameters in an intuitive and interactive way.

A critical parameter is the interactive and adaptive behavior that dbTouch systems should provide. Changing the layout can be done in steps as it is in general an expensive operation, requiring a full copy of the data. Depending on the size of the current object, dbTouch should choose to create the new format for only a sample of the data, giving back to the user a quick response and new data object(s) to query. When and if the user requests for more detail within the new object, e.g., with a zoom-in gesture, then more data can be retrieved from the old layout.

2.9 Query Plans and Complex Queries

Up to now, we mainly discussed about simple scans using the slide gesture. However, any kind of complex query is possible.

Complex Queries. For example, the slide gesture can be used in order to run any kind of aggregate over a column object or to perform selections by posing a *where* restriction to the scan. In addition, multi-column query plans are also possible. Again it is a matter of enabling the proper actions. For example, we can enable *where* and *select* actions on one or more columns as well as we can enable a *join* for a pair of columns. Then, with the slide gesture over one of the columns used with a *where* action, a user can go through the data and drive the query processing steps. The tuple identifiers captured in the object where we apply the slide gesture define the data processed. Essentially, the same discussion as the one we did for the slide gesture for scans holds here as well, i.e., regarding slide speed, direction, object sizes, etc; the only difference is that here the system needs to perform a more complex computation per data entry as opposed to simply visualizing the touched entry as in the case of a scan.

Joins. Maintaining the interactive behavior during join queries is a challenging topic. The join is primarily a blocking operator as the hash-join is the typical choice. The same is true for hash-based grouping. However, in dbTouch we do not know up front all the data we are going to process. This is only known as the gesture evolves. Subsequently, we cannot use a hash-join as we do not know which data we should use to build the hash table. Of course, we could use the whole input but this would result in a significant delay. As such, exploiting non blocking options is a necessary path in dbTouch. Similarly, caching of hash tables across the various sample copies can enhance future queries.

Optimization. With more complex queries than plain scans and aggregates, optimization plays a crucial role. This is for the same reasons as in traditional databases. The order of operators and the actual physical operators used can have a significant impact. However, contrary to traditional databases, in a dbTouch system we do not know up front how much data we are going to process. When a user initiates a slide gesture for a complex query, only a small part of the data may be processed or even all of the data may be processed.

Not knowing which part of the data is going to be processed makes optimization decisions challenging as for dif-

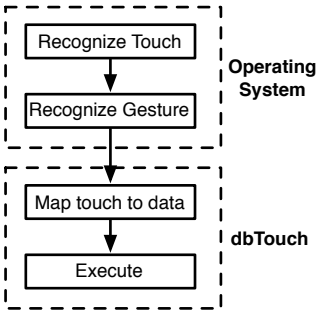

Figure 3: dbTouch system layers.

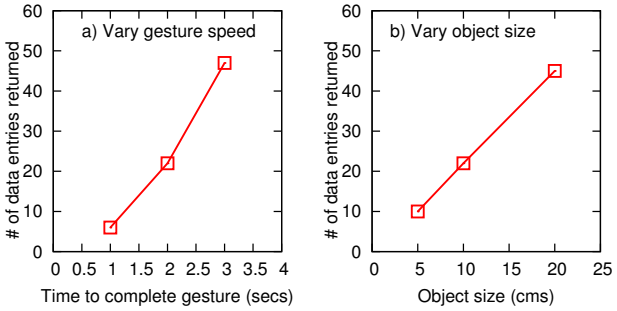

Figure 4: Effect of varying object size and slide gesture speed during a slide for interactive summaries.

ferent parts of the data in the same table, different properties may apply. In this way, dbTouch brings an interesting scenario for adaptive optimization approaches that interleave with query execution. dbTouch needs to figure out the proper optimization decisions on-the-fly, while still maintaining good response times. At the same time, as the slide direction and speed evolves dbTouch needs to adapt these decisions to the new data which is processed. Contrary to adaptive approaches for traditional databases, dbTouch does not control the data flow and thus it is much harder to make reliable decisions regarding when to switch to a different query plan.

3. DBTOUCH PROTOTYPE

In this section, we briefly discuss the details of our implementation over IOs for IPad and we provide an early evaluation. We build on top of IOs SDK 5.1 (9B176) and we test on IPad 1, while our development environment is XCode 4.3.2 (4E2002).

Implementation. Our current dbTouch prototype is essentially a small prototype database kernel implemented on top of IOs in objective C. In many ways it resembles a hybrid kernel where data is stored in fixed-width dense arrays or matrixes. Similarly to the typical SQL input, parser, optimizer, execution flow of modern database systems, the dbTouch prototype goes through a flow that begins with a touch input, and continues with gesture recognition as opposed to parsing. Then, depending on the gesture, the location touched, the size of the object touched, etc., the proper dbTouch execution methods are called.

Figure 3 shows a high level view of the dbTouch stack and how the various system layers interact. dbTouch feeds from the operating system; once a touch is registered by the operating system, dbTouch takes action to find which data should be processed and what kind of actions should be performed. This flow is not per query as it is in database systems; instead, dbTouch goes through these steps for every touch input on a data object. Our current prototype supports several of the gestures/operators seen in the previous section such as slide or single tap for scan, aggregation and interactive summaries as well as zoom-in/zoom-out for a more detailed or a more high level exploration.

Evaluation. Evaluating a dbTouch system is a challenge by itself mainly due to the interactive nature of the paradigm where a user continuously gives input to the system and drives query processing actions via touch gestures. To give a glimpse on the interaction and exploration properties of our current dbTouch prototype, we provide two experimentation examples which are characteristic of the exploration process.

Varying Gesture Speed. The first experiment observes what happens with a varying speed of applying the basic slide gesture for an interactive summaries query, using an *average* aggregation and 10 data entries for each summary. The set-up is as follows. A rectangle object in a vertical position represents a column of 10^7 integer values. The height of the object is 10 centimeters. We apply the slide gesture starting from the top end of the object and finishing at the bottom end. We repeat the gesture 3 times. Each time the gesture is completed with a different speed and each time we measure the amount of data entries that appear. Figure 4(a) shows the results. As we slow down the speed of the gesture, we are able to observe/process more data. dbTouch captures more touch input and it can map this input to object identifiers of the underlying data. In this way, a varying gesture speed allows drilling down to detail or getting a high level view of the data. It gives users the power to explore the data and to continuously adjust the exploration properties in an interactive way as results appear.

Varying Object Size. In our second experiment, we test what happens as the size of a data object changes. The set-up is the same as before. This time we apply a zoom-in gesture to progressively increase the size of the data object. For each size, we perform a slide gesture with the same speed (from top to bottom) and observe the amount of data processed. At each step we double the size of the object and we take double the time to complete the slide gesture. Figure 4(b) shows the results. By adjusting the object size we allow for more detail; as the size increases, the same gesture speed allows the inspection of more data. Similarly to our discussion in the previous paragraph, via adjustments of the object size a user can interactively get a more fine grained or a more high level view of the data on demand.

Summary. Overall, dbTouch provides a promising playground to promote interactive data exploration; data analysts can get a quick feeling of the data on demand, while continuously adjusting the exploration parameters and while using only intuitive touch input as opposed to writing SQL scripts.

4. CHALLENGES AND OPPORTUNITIES

The previous sections described the dbTouch vision and discussed several challenges when designing the individual components of a dbTouch kernel. Here, we discuss an additional set of more broad challenges and opportunities.

Interactive Behavior. Most prominently, developing complete touch-based database kernels requires in depth study of optimization and algorithmic choices. The main challenge is in continuously reacting to user input gestures and properly anticipating the next move such as to avoid big stalls. In all cases, the response times to touch events need to be kept low. The whole design of dbTouch kernels should be geared towards an interactive and adaptive behavior. There should always be a maximum possible wait time for a single touch regardless of the query and the data sizes. Approximate query processing in combination with dbTouch may be an interesting direction, i.e., results appear within the expected response time and then they are continuously refined.

Data Visualization. In addition, studying dbTouch across various application-specific data visualization formats and the implications this brings for designing dbTouch kernels is of high importance. The relational-like visualization we used to describe dbTouch in this paper is a generic format but application-specific formats can enhance the user experience as well as the ability to quickly interpret the data. The data visualization community has tremendous results in this area. For example, maps can be used to cluster data based on geographical information. Data may be colored to quickly determine patterns, outliers, etc. In addition, graphs and mind-maps may be used to depict trends and correlations. Regardless of the visualization format, dbTouch needs to maintain its adaptive character and the goal for interactive response times. The main challenge is whether we can create a generic dbTouch kernel which can be used as a back-end to various visualization formats.

Remote Processing. Another interesting direction is the option of remote processing when the touch device is an interface to a cloud or server facility which does the bulk of the query processing. For example, the server may store the base data and the big samples, while the touch device may store only small samples. Then, during query processing dbTouch may use both local and remote data to process queries; as users request more detail, more requests are shipped to the server. Still, though, dbTouch should guarantee low response times. However, sending a new remote request for every single touch input of a long gesture will lead to extensive administration and communication costs. As a result, dbTouch needs to carefully exploit both local and remote data, i.e., use local data to feed partial answers, while in the mean time more fine-grained answers are produced and delivered by the server.

Alternative Interfaces. Finally, studying other forms of input and its implications in creating interactive database kernels is another exciting opportunity. Motion input, speech recognition as well as combinations of those input methods are some of the options. As in the case of visualization formats, the main challenge here is whether it is possible to create a generic adaptive kernel which can be adjusted easily for any input method.

5. RELATED WORK

This paper takes inspiration from several research areas related to data management.

Data Exploration. Data exploration has seen significant support during the past decade. Several researchers argue towards exploration based database kernels and propose several directions such as sampling based kernels [3, 31, 40], adaptive indexing [26] and adaptive data loading [24, 4, 5]. Overall, this is a quite promising and largely unexplored research area. dbTouch complements ongoing research efforts by providing a promising alternative when it comes to how users interact with an exploration based database kernel. Ideas such as sampling, adaptive indexing or loading can be exploited in the dbTouch context with new challenges on how to adapt to the dynamic touch patterns in gestures.

Online Aggregation. Online aggregation [22, 38] is also related to dbTouch. In online aggregation the system continuously returns results as they are created. A confidence metric is also calculated and reported, allowing the user to terminate query processing when confidence reaches satisfactory levels. Online aggregation techniques can certainly be exploited in dbTouch; dbTouch brings additional challenges as the user drives the speed of requesting more data and determines the data to be processed dynamically.

Visual Analytics. The idea that simple text mode can hurt usability is not new. Polaris is the pioneering system from Stanford University for visual analytics [41, 42]. In Polaris, there are two distinct features to ease usability and exploration: (a) users can synthesize SQL queries by drag and drop actions and (b) results appear directly in the proper visual format. For example, results can appear directly in a bar graph or in other graph formats depending on the kind of data. The ideas pioneered in Polaris and later commercialized in the Tableau system are directly in line with the dbTouch vision. In addition to Tableau, there are more commercial systems exploiting similar ideas; in particular, Data Clarity and VisuaLinks from Visual Analytics Inc. and the Visual Analytics platform by SAS. All these systems have the same high level goal; they try to provide an easy way to construct queries graphically and to visualize results using the proper format according to the result data.

In all cases above, the underlying database system is a traditional system; what changes is the input method and the visualization of the results. What dbTouch brings is the idea of building database kernels to inherently support touch interfaces and interactive exploration at their core, taking visual analytics one major step further by allowing systems to increase their interactive and exploratory character. Users in dbTouch do not pose a traditional query by pointing in an object and then waiting for an answer. Instead, they drive the query processing actions, working on top of data samples and by continuously defining (touching) the data to be processed next; they have full control of the exploration process.

Data3. Data3 is a recent system which exploits motion sensing [23]. Data3 builds on top of an existing stream engine and allows users to interact with the stream engine using gestures as opposed to using a declarative language.

Users can define streams and pose continuous queries via gestures. Essentially, Data3 implements an interface to an existing and traditional system. This is still an exciting and crucial step. However, when it comes to database architectures, nothing changes regarding how data is processed or the ability of the system to adapt to user behavior.

On the contrary, the dbTouch vision is about redesigning systems to exploit the new opportunities that appear with the emerging input methods such as touch and motion sensing. In dbTouch, we do not pose traditional queries using a new input method; instead, we envision new ways to process data and a new query processing paradigm all together; users drive the low level query processing actions by pointing to the data that should be processed in every query processing step. This changes drastically the way we should design database kernels and brings several opportunities and challenges for data exploration.

6. CONCLUSIONS

Data exploration has emerged as a critical need in our effort to handle big data and the new stream of applications that rely on data management. Data exploration tools are meant to assist when we are in need to quickly analyze data, not necessarily knowing exactly what we are looking for; the results of one query inspire the formulation of the next query. At the same time, touch interfaces have proven to be extremely successful mainly by promoting usability.

In this paper, we introduce dbTouch, a new research direction towards touch-based database kernels, tailored for interactive data exploration in the big data era. We propose to rethink database architectures towards designs which can cope with the different needs posed by a full-blown touch-based database system. This paper discusses the dbTouch vision and challenges along with several gestures and accompanying exploration-based database operators driven by touch patterns. In addition, we present a dbTouch prototype over IOs for IPad, showcasing the simplicity and intuitive nature of dbTouch. Overall, dbTouch opens a new and completely unexplored territory for database researchers in the path towards big data analytics and insights.

7. REFERENCES

[1] A. Abouzeid, K. Bajda-Pawlikowski, D. J. Abadi, A. Rasin, and A. Silberschatz. Hadoopdb: An architectural hybrid of mapreduce and dbms technologies for analytical workloads. *PVLDB*, 2(1):922–933, 2009.

[2] F. N. Afrati, A. D. Sarma, D. Menestrina, A. G. Parameswaran, and J. D. Ullman. Fuzzy joins using mapreduce. In *ICDE*, pages 498–509, 2012.

[3] S. Agarwal, A. Panda, B. Mozafari, S. Madden, and I. Stoica. Blink and it's done: Interactive queries on very large data. In *PVLDB*, 2012.

[4] I. Alagiannis, R. Borovica, M. Branco, S. Idreos, and A. Ailamaki. Nodb: efficient query execution on raw data files. In *SIGMOD*, 2012.

[5] I. Alagiannis, R. Borovica, M. Branco, S. Idreos, and A. Ailamaki. Nodb in action: Adaptive query processing on raw data. *PVLDB*, 5(12):1942–1945, 2012.

[6] R. Barber, P. Bendel, M. Czech, O. Draese, F. Ho, N. Hrle, S. Idreos, M.-S. Kim, O. Koeth, J.-G. Lee, T. T. Li, G. M. Lohman, K. Morfonios, R. Müller, K. Murthy, I. Pandis, L. Qiao, V. Raman, R. Sidle, K. Stolze, and S. Szabo. Business analytics in (a) blink. *IEEE Data Eng. Bull.*, 35(1):9–14, 2012.

[7] P. A. Bernstein, C. W. Reid, and S. Das. Hyder - a transactional record manager for shared flash. In *CIDR*, pages 9–20, 2011.

[8] P. Boncz, M. Zukowski, and N. Nes. MonetDB/X100: Hyper-Pipelining Query Execution. In *CIDR*, 2005.

[9] S. Chaudhuri. What next? a half-dozen data management research goals for big data and cloud. In *PODS*, 2012.

[10] S. Chen, P. B. Gibbons, and S. Nath. Rethinking database algorithms for phase change memory. In *CIDR*, pages 21–31, 2011.

[11] P. Cudré-Mauroux, E. Wu, and S. Madden. The Case for RodentStore: An Adaptive, Declarative Storage System. In *CIDR*, 2009.

[12] C. Curino, E. P. C. Jones, R. A. Popa, N. Malviya, E. Wu, S. Madden, H. Balakrishnan, and N. Zeldovich. Relational cloud: a database service for the cloud. In *CIDR*, pages 235–240, 2011.

[13] J. Dittrich and A. Jindal. Towards a one size fits all database architecture. In *CIDR*, pages 195–198, 2011.

[14] A. Doan, R. Ramakrishnan, and A. Y. Halevy. Crowdsourcing systems on the world-wide web. *Commun. ACM*, 54(4):86–96, 2011.

[15] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin. Crowddb: answering queries with crowdsourcing. In *SIGMOD*, 2011.

[16] G. Graefe, F. Halim, S. Idreos, H. Kuno, and S. Manegold. Concurrency control for adaptive indexing. *PVLDB*, 5(7):656–667, 2012.

[17] M. Grund, P. Cudré-Mauroux, J. Krüger, S. Madden, and H. Plattner. An overview of hyrise - a main memory hybrid storage engine. *IEEE Data Eng. Bull.*, 35(1):52–57, 2012.

[18] 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.

[19] P. Hanrahan. Analytic database technologies for a new kind of user - the data enthusiast. In *SIGMOD*, 2012.

[20] N. Hardavellas, I. Pandis, R. Johnson, N. Mancheril, A. Ailamaki, and B. Falsafi. Database servers on chip multiprocessors: Limitations and opportunities. In *CIDR*, pages 79–87, 2007.

[21] S. Harizopoulos, M. A. Shah, J. Meza, and P. Ranganathan. Energy efficiency: The new holy grail of data management systems research. In *CIDR*, 2009.

[22] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online aggregation. In *SIGMOD Conference*, pages 171–182, 1997.

[23] S. Hirte, A. Seifert, S. Baumann, D. Klan, and K.-U. Sattler. Data3 - a kinect interface for olap using complex event processing. In *ICDE*, pages 1297–1300, 2012.

[24] S. Idreos, I. Alagiannis, R. Johnson, and A. Ailamaki. Here are my data files. here are my queries. where are my results? In *CIDR*, pages 57–68, 2011.

[25] S. Idreos, F. Groffen, N. Nes, S. Manegold, K. S. Mullender, and M. L. Kersten. Monetdb: Two decades of research in column-oriented database architectures. *IEEE Data Eng. Bull.*, 35(1):40–45, 2012.

[26] S. Idreos, M. Kersten, and S. Manegold. Database Cracking. In *CIDR*, 2007.

[27] S. Idreos, M. Kersten, and S. Manegold. Updating a Cracked Database. In *SIGMOD*, 2007.

[28] S. Idreos, M. Kersten, and S. Manegold. Self-organizing Tuple-reconstruction in Column-stores. In *SIGMOD*, 2009.

[29] 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):585–597, 2011.

[30] A. Kemper and T. Neumann. Hyper: A hybrid oltp&olap main memory database system based on virtual memory snapshots. In *ICDE*, 2011.

[31] M. L. Kersten, S. Idreos, S. Manegold, and E. Liarou. The researcher's guide to the data deluge: Querying a scientific database in just a few seconds. *PVLDB*, 4(12):1474–1477, 2011.

[32] N. Khoussainova, M. Balazinska, and D. Suciu. Perfxplain: Debugging mapreduce job performance. *PVLDB*, 5(7):598–609, 2012.

[33] W. Lang and J. M. Patel. Towards eco-friendly database management systems. In *CIDR*, 2009.

[34] D. B. Lomet, A. Fekete, G. Weikum, and M. J. Zwilling. Unbundling transaction services in the cloud. In *CIDR*, 2009.

[35] A. Marcus, E. Wu, S. Madden, and R. C. Miller. Crowdsourced databases: Query processing with people. In *CIDR*, pages 211–214, 2011.

[36] R. Müller, J. Teubner, and G. Alonso. Data processing on fpgas. *PVLDB*, 2(1):910–921, 2009.

[37] A. Nandi and H. V. Jagadish. Guided interaction: Rethinking the query-result paradigm. *PVLDB*, 4(12):1466–1469, 2011.

[38] N. Pansare, V. R. Borkar, C. Jermaine, and T. Condie. Online aggregation for large mapreduce jobs. *PVLDB*, 4(11):1135–1145, 2011.

[39] A. G. Parameswaran, H. Garcia-Molina, H. Park, N. Polyzotis, A. Ramesh, and J. Widom. Crowdscreen: algorithms for filtering data with humans. In *SIGMOD Conference*, pages 361–372, 2012.

[40] L. Sidirourgos, M. L. Kersten, and P. A. Boncz. Sciborq: Scientific data management with bounds on runtime and quality. In *CIDR*, 2011.

[41] C. Stolte, D. Tang, and P. Hanrahan. Polaris: A system for query, analysis, and visualization of multidimensional relational databases. *IEEE Trans. Vis. Comput. Graph.*, 8(1):52–65, 2002.

[42] C. Stolte, D. Tang, and P. Hanrahan. Query, analysis, and visualization of hierarchically structured data using polaris. In *KDD*, pages 112–122, 2002.

[43] M. Stonebraker, D. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E.O'Neil, P. O'Neil, A. Rasin, N. Tran, and S. Zdonik. C-Store: A column oriented DBMS. In *VLDB*, 2005.

[44] P. Triantafillou. Anthropocentric data systems. *PVLDB*, 4(12), 2011.

APPENDIX

A. DBTOUCH DEMO

Here, we describe a demo proposal for dbTouch. The main goal of dbTouch is to enable data exploration through interactive touch-based database kernels. In this way, the demo focuses on exposing the exploration properties and opportunities of dbTouch to the audience.

Set-up and Scenarios. The main idea for the demo is that the audience will get direct access to the dbTouch prototype. We will provide our development IOs device for this purpose (IPad 1). Data will be loaded in a tablet device and the audience will get the chance to use the system and interact via gestures to query and explore the data. We will provide alternative data sets with a varying set of properties and patterns. The audience attending the demo will have the task of discovering these properties by interacting with dbTouch via gestures.

Exploration Contest: dbTouch Vs. DBMS. In addition, the audience attending the demo can participate in an exploration contest. We will provide a laptop installed with the open-source column store DBMS, loaded with the same data sets as dbTouch. Two audience members will simultaneously start exploring the data sets; one member will be using the tablet dbTouch prototype, while the other member will be using the SQL interface of the DBMS on the laptop. Both audience members will be free to perform any kind of query processing actions, i.e., to apply any kind of supported gestures in dbTouch and to fire any kind of SQL queries in the DBMS. The winner is the one who can first figure out the data properties and patterns.

Summary. Overall, the demo will allow users to play with dbTouch on an IPad and to directly compare with a DBMS experience for the same tasks.