Study on potential capabilities of a noDB system

L. Kananbala Devi
Department of Computer Science & Engineering and I.T,
Don Bosco College of Engineering and Technology, Assam Don Bosco University, Guwahati, India

ABSTRACT

There is a need of optimal data to query processing technique to handle the increasing database size, complexity, diversity of use. With the introduction of commercial website, social network, expectations are that the high scalability, more flexible database will replace the RDBMS. Complex application and Big Table require highly optimized queries. Users are facing the increasing bottlenecks in their data analysis. A growing part of the database community recognizes the need for significant and fundamental changes to database design. A new philosophy for creating database systems called noDB aims at minimizing the data-to-query time, most prominently by removing the need to load data before launching queries. That will process queries without any data preparation or loading steps. There may not need to store data. User can pipe raw data from websites, DBs, excel sheets into two promise sample inputs without storing anything. This study is based on PostgreSQL systems. A series of the baseline experiment are executed to evaluate the Performance of this system as per -a. Data loading cost, b-Query processing timing, c-Avoidance of Collision and Deadlock, d-Enabling the Big data storage and e-Optimize query processing etc. The study found significant possible capabilities of noDB system over the traditional database management system.

Keywords: NoDB, Big data storage, Raw Data files, Parallel dicing

I. INTRODUCTION

Information sharing is increasing simultaneously with an increase in the speed of networks as well as server storage capacity. Applications are getting more and more complex to match the user friendliness required in current e-world. All of these are running on databases which in turn are becoming larger and larger and maintaining such a huge database is a costly affair. E.g. – The databases of social networking sites like Facebook, online trading market data etc. Users are facing the increasing bottlenecks in their data analysis. Large data may take more time to prepare, to load the data into the database and to execute them. Application like scientific data analysis and social networks has complexity and the increased data-to-query time. Such data deluge will only increase in the near future and much more data are expected.

The amount of available data outgrown the capabilities of query processing technology. Many emerging applications, from social networks to scientific experiments, are representative examples of this deluge, where the rate at which new data is produced exceeds any past experience. Scientific analysis such as astronomy is soon expected to collect multiple Terabytes of data even on a daily basis. Applications like social networks are already confronted with increasing stream of huge data inputs. So, here is an early needs for efficient significant data processing to enable the evolution of businesses. There is a need to design the database, ranging from low-level architectural redesigns to changes in the way users interact with database systems. Complex applications require highly optimized queries so much that major networking organizations such as Facebook and Google wrote custom big data databases like BigTable to manage them. Typically it would devote 20-25 percent of resources to data management, so there's lots of savings. But when creating new applications, we can also reduce the time barrier to building services that may run or access multiple databases. It also needs monitoring and maintaining, scaling of application in distributed environment across multiple service providers.

A noDB system aims at minimizing the data-to-query time by removing the need to load data before launching queries. This present the prototype implementation, built on top of SQL engine, which allows for efficient query execution over raw data files. It performs with zero initialization overhead. So study of noDB philosophy present the techniques and its creating methods related to auto tuning tools, adaptive

indexing, information extraction and external files, caching, adaptive loading and straw-man approaches to situ query processing[1], [2], [3], [4],[5],[6].

The remainder of this study report is systematized as follows. In Section II, we present a literature survey. Section III provides an introduction of the proposed study. Section IV shows the experimental result. The section V provides the summary of the study and concludes the paper.

II. RELATED WORKS

2.1 Data Files to Queries and Results

The study of S. Idreos, I. Alagiannis, A. Ailamaki and R. Johnson proposes a new generation system[7]. It links only user requirement to the raw data files and immediately the queries are fired without preparation steps. Internally the system takes care of selectively, adaptively and incrementally of the queries. Just part of the inquire data is loaded at any specified time, and it is being stored and accessed in the suitable format suitable as per workload. They argue towards a new generation of systems that provide a hybrid experience between using a Unix tool and a DBMS; all they need as a start-up step is a pointer to the raw data files, i.e., the at files containing the data, for example, in CSV format. There is zero initialization over-head as with a scripting tool. It also retained all advanced query processing capabilities and performance of a DBMS. The main idea is that data remains in at files allowing the user to edit or change a file at any time. When a query arrives, the system will take care of bringing the need data, and it will store it and evaluate it accordingly.

This demonstrates the initialization overhead of DBMS and the flexibility of using a scripting language. Then, this demonstrate the suitability of a DBMS for data exploration. Various policies are available regarding how data can be fetched, cached, reused and how this whole procedure can happen on-the-fly, integrated with query processing in a modern DBMS. This provides a prototype implementation and evaluation over MonetDB. The results clearly show the potential and benefits of the approach as well as the opportunity to further study and explore this topic. A probable system architecture having above facility is given as figure1.

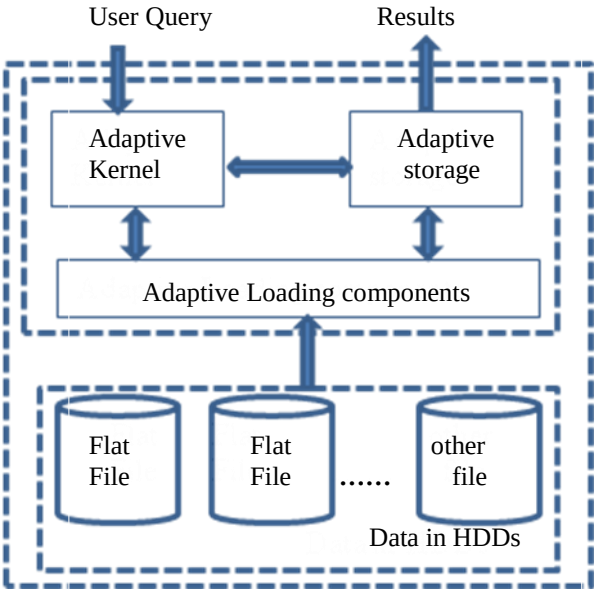

Fig1. A possible Fast excess Database System

2.2 Efficient Query Execution on Raw Data Files

The study of Ioannis Alagiannis, Renata Borovica, Miguel Branco, Stratos Idreos, and Anastasia Ailamaki [8] contributes to the design a new paradigm in database systems which do not require data loading while still maintaining the complete feature set of modern database system. This show how to make raw data files a first-class citizen, fully integrated with the query engine. Through the design and lessons learned by implementing the noDB philosophy over a modern DBMS, this discusses the fundamental limitations as well as the strong opportunities that such a research path brings. The study identifies performance bottlenecks particular for in situ processing having repeated parsing and tokenizing overhead. Such problem could be address by introducing an adaptive indexing mechanisms [9], [10], [11], [12], [13], [14], [15]. This mechanism maintains positional information to provide efficient access to raw data files that have flexible caching structure. The PostgresRaw is able to avoid the loading cost completely, while matching the query performance of plain PostgreSQL. It outperformed in many cases. It is very possible to design and implement such noDB systems over existing database architectures, giving several effect in usability and performance.

The contributions of such study are as follows.

1. This study converts a traditional row-store (PostgreSQL) into a noDB system (PostgresRaw), and discover that the main bottleneck is the repeated access and parsing of raw files. Therefore, this design an innovative adaptive

indexing mechanism that makes the trip back to the raw data files efficient.

2. It demonstrates that the query response time of a noDB system can be competitive with a traditional DBMS. It shows that PostgresRaw provides equivalent or faster access on the TPC-H dataset (scale factor 10) compared to PostgreSQL, even without prior data loading.
3. It shows that the noDB systems provide quick access to the data under a variety of workloads (micro-benchmarks) as well as different file formats. PostgresRaw query performance improves adaptively as it processes additional queries, and it quickly matches or outperforms traditional DBMS, including MySQL and PostgreSQL.
4. It describes opportunities with the noDB philosophy, as well as challenges, identifying primary overheads such as data type conversion.

2.3 Adaptive Query Processing on Raw Data

The study of Ioannis Alagiannis and group[15], [16], demonstrates a showcase for a new philosophy for designing database systems called noDB. A noDB aims at minimizing the data-to-query time, most prominently by removing the need to load data before launching queries. This work present the archetype implementation, built on top of SQL engine, which allows for capable query execution above raw data files. It gives zero initialization overhead. It illustrates that PostgresRaw adaptively touches, parses, caches and indexes raw data files. The study purposed PostgresRaw, a full noDB system based on PostgreSQL. The study demonstrates the performance of its core components in a range of scenarios. This also presents a comparison between PostgresRaw and other broadly used DBMS in an interactive way. The study can be discussed on following points. Some of the advantages of such study are given below as per the subsection of their importance.

A. Innovation

PostgresRaw immediately starts processing queries without any data preparation. As more queries are processed, response times increase due to the adaptive properties of PostgresRaw. The study demonstrates the effects by observing internal components-the indexes and caches on raw data files, which allow PostgresRaw to adaptively and continuously helps to improve its performance..

B. Visual Experience

The user of such system has the capability to interact with the system through a GUI that allows them to change the input characteristics of the workload. Properties such as the number of attributes and the width of the attributes may significantly change the performance of a noDB system. The graphical interface provides access to PostgresRaw specific execution configuration parameters. For example, the user can allow or can halt such noDB components and identify the amount of storage room which is devoted to internal indexes and caches. Users will be able to change such parameters and monitor the impact on performance.

C. Postgresraw Architecture:

The prototype of noDB system known as PostgresRaw, is designed and implemented by modifying PostgreSQL version 9.0. The main bottleneck of in situ query processing is the access to raw data. The aim of PostgresRaw of the study is geared towards improving access on raw data in two ways by

(a) Speeding up the steps required via raw data indexing
(b) Eliminating the need to access hot raw data via caching.

Assuming that raw data is stored in comma-separated value (CSV) files and are challenging for an in situ engine, by considering the high conversion cost.

D. Query plans in PostgresRaw:

When a query submitted to PostgresRaw references relational tables that are not yet loaded. PostgresRaw needs to access the respective raw file(s). PostgresRaw overrides the scan operator with the ability to access raw data files directly, while the rest of the query plan, generated by the optimizer, works without any changes of baseline theory as compared to a conventional DBMS.

E. Parsing and Tokenizing Raw Data:

Every time a query needs to access raw data, PostgresRaw has to perform parsing and tokenization of the raw data. Having the binary values at hand, PostgresRaw feeds those principles in a classic DBMS query plan.

F. Selective Tokenizing:

PostgresRaw reduces the tokenizing costs by opportunistically aborting tokenizing tuples as soon as the required attributes for a query have been found. It occurs at a per tuple basis. Supposing that CSV files are prepared in a row-by-row basis, particular tokenizing does not bring any I/O benefits; nonetheless, it significantly reduces the CPU processing costs.

G. Selective Parsing

In addition to particular tokenizing, PostgresRaw also employs selective parsing to further reduce raw file access costs. It needs only to convert to binary the value required for the remaining query plan.

H. Selective Tuple Formation

To fully capitalize on particular parsing and tokenizing, PostgresRaw also applies selective tuple formation. Tuples are not fully composed but only contain the attributes needed for a

given query. In PostgresRaw, tuples are only created after knowing which tuples qualify.

2.4. Columnar storage

The concept of adaptive query is incorporated in our study. Google uses a strongly-typed data model for the serialization of structured data. A record wise representation and columnar representation is shown in given figures. By opting for a columnar striped representation and storing values of a nested field continuously -for example all values of nested field A.B.C. then one can read all values of a nested field without reading unneeded neighboring values. In addition, data columns are easier to compress[17].

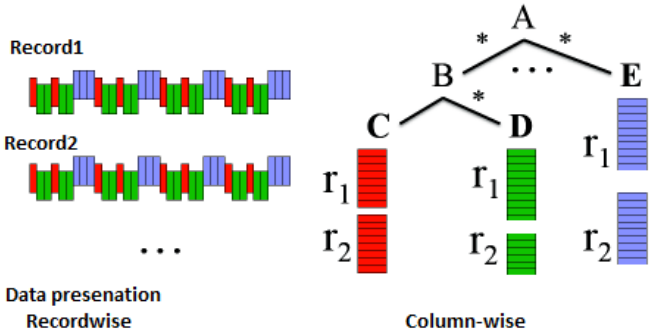

Fig2. Data presentation: a. Record wise, b. Column-wise

2.5 Adaptive Query Service for In Situ Querying

Manolis Karpathiotakis[17] examine the noDB model, which proposes querying data in situ. They describe the structures it introduces to overcome the limitations of having to access raw files repeatedly, its adaptive nature, and see how it is manifested in a system extending a traditional DBMS. To address shortcomings of query execution in noDB, this examine a data-centric query execution model that deviates from the traditional Volcano iterator model. To materialize this design, query compilation techniques are employed, leading to the generation of machine code. Besides bringing performance benefits, these code generation mechanisms can be used in an alternative manner to create a query engine producing data-specific code at query time, thus supporting various data formats. In addition, affected by the sheer amounts of data that need to be managed, this present a cloud-based system named Dremel. Dremel natively supports queries over nested data; its entire design is aiming to optimize aggregation queries and has been shown to scale to petabytes of data. Finally, inspired by the previous approaches, this briefly present the vision for a lightweight database service which allows users to seamlessly pose interactive queries over raw files distributed over the cloud, regardless their format. As the noDB model suggests, raw data need to become a first-class citizen in a DBMS and also by Dremel, multiple types of data need to be managed, not being restricted to relational formats. Cloudscale deployments are required for processing, both due to the excessive amounts of data present, but also so that data can be queried in their original locations. Still, single-node performance cannot be overlooked.

2.6 Other studies on fast data processing

The study of [18] proposes an architecture that uses a column store as a basis for a lazy index. In this process, fields are stored, unprocessed, embedded in the data until the first query is issued. Then a simple index is built and facilitate faster querying, using a parallel technique and then and stored on disk[19].

A new study known as dbTouch is a new research direction towards the next generation of data management systems. It essentially supports data exploration by allowing user touch-based interaction. Data is represented in a visual format, while users can feel those shapes and interrelate with their gestures. In adbTouch scheme, the whole database kernel is geared towards fast responses. The user drives query processing via touch gestures of such system[20].

A study on a Light-Weight Data Management Layer over HDF5 has been proposed[21]. It allows server-side sub setting and aggregation on scientific datasets stored in HDF5. Their tool generates code for hyper slab selector and content-based filtering, and parallelizes selection and aggregation queries efficiently using novel algorithms.

II.

III. PROPOSED METHODOLOGY

A noDB system is an API that takes inputs from internal and external data. It compares all data sources with a trailing confidence score on the item that attached to a unique ID. The noDB has facilities to store, search, update and delete objects in memory. Moreover, index can be created over those objects for faster response. A **Pure noDB** application will have all the objects model stored in memory. It requires enough memory to fulfill the application requirements and the noDB spaces to store its associated objects. The noDB can be broken into Promise, Interval and Query parts. It sits on the top and funnels data through the Promise, Interval and Query.

The study included the purpose of a full noDB system based on PostgreSQL. Performance evaluation on this system as per

a) Data loading cost
b) Query processing timing
c) Avoidance of Collision and Deadlock

d) Enabling the Big data storage.
e) Optimize query processing etc

This can be achieved by developing APIs for all kind of data transactions happens between application and the source data. The input data can be in any format which is easy and fast to access but not the traditional DBMS. In this way, there is no need to store data. The raw data can be piped from websites, DBs, excel sheets into promise without storing anything. We can also start adding more data as we promises isolate and ultimately provide structured data with helps of machine learning. Some actions to be followed are - choose proper file system/extension and a flexible memory storage system to which data can be written. Finally, we can also use these flexible memory storage devices as the data backup for future purposes and avoid using traditional database systems. Database cracking [12] also helps to improve case selections and tuple reconstruction. This improved the performance using an optimized physical design, where storage is adapted to the queries via indexes.

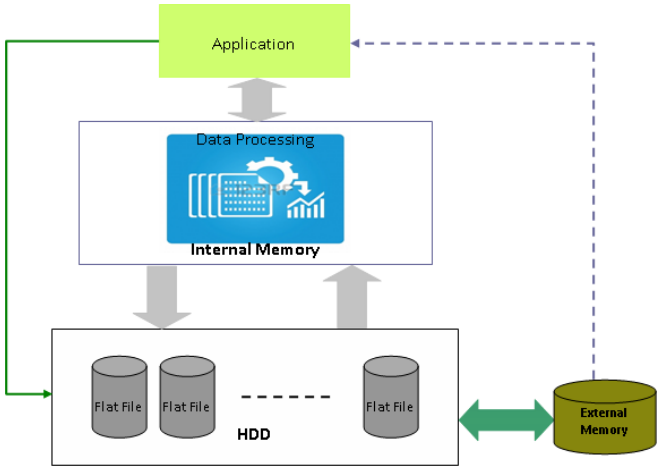

Fig3. A pure noDB proposed structure.

IV. PROPOSED PERFORMANCE EVALUATION

This study is one of our ongoing study. The results provided are as per the parameters used in the baseline of the study. The performance is compared with that of traditional Database Management systems. Several enhancement can be incorporated. The outcome of the study on possible capability of a noDB system is discussed in following subtopics.

4.1 Data loading cost

The traditional Database Management system(DBMS) requires several steps to load data. The NoDB loading allows to copied the data block from the flat files into the DBMS. The flat files are in CSV format. For processing a query in a DBMS, the firstly it has to incur the complete loading charge. Thus, the first query response time can be seen as the cost to load the data plus the cost to run the actual query. There may be the table having several tuples (in billions) with hundreds of columns; a query might be interested in only a few of those. This pays the significant cost of the loading procedure hitting some of the unnecessary columns.

4.2 Query processing timing

A traditional query undergoes the pre-processing process before its actual execution. The process is complicated if there presents any embedded SQL. For example accessing DB2 data from COBOL programs. The process of Binding is also required. A DBMS has to first load all columns before it can process any queries. It has to replicate the complete data set.

The noDB explicitly identify the data parts that are interested in with zero preparation or quick pre-processing. They do not need to replicate the data, and this can actually edit the data with a text editor directly at any time and fire a query again. In such way, the noDB system has much more flexible and optimistic than a traditional DBMS for inspecting and quickly initiating explorations for handing the large datasets.

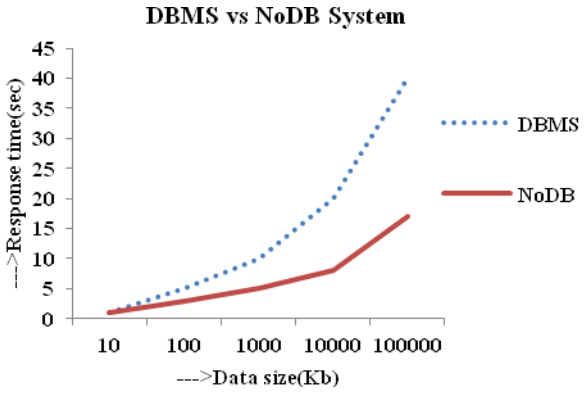

Fig4. Performance evaluation NoDB System(response time)

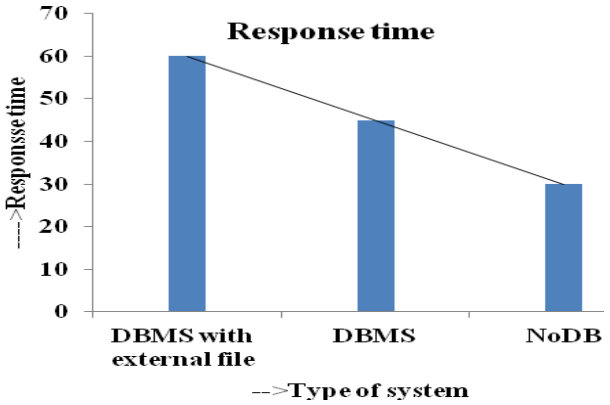

Fig5. Performance of Different system (response time) with respect to processing of raw data

4.3 Avoidance of Collision and Deadlock

The NoBD does not undergoes the lengthy process of loading data. The NoDB loading allows to copied the data block from the flat files into the DBMS. For query execution, it has the facility to fires the query directly on selected group of data. The processes of Dicing and Slicing are used for such process [23]. The essential point of the transaction is that it bundles multiple steps into a single operation. The number of such steps will be less of NoDB system as compared to traditional database systems. As the steps are very minimal, the changes of the collision and deadlock are also very less. The normal transaction processing operation may not have any types of deadlocks. However, the deadlock can be detected for the heavy loads such us transaction processing in heavily loaded social networking website. The baseline experiments executed using the algorithm of Dicing minimized the deadlock.

4.4 Enabling the Big data storage.

Bigtable is a storage system on distributed environment that managed structured data that is designed to scale to a very large size- petabytes of data across thousands of commodity servers. In bigtable, it uses the MapReduce process where the input data and output data is partitioned into rows and columns. Typically, the information to a MapReduce computation is a file stored as blocks (contiguous byte extents) in a distributed file system, and MapReduce processes each block of the input file in parallel. This approach works well because typical MapRe-duce computations are able to independently process arbitrary byte extents by sequentially reading each information block from the file system. However, requests for data expressed at the abstraction level of the logical, scientific data model do not always correspond to contiguous, low-level byte extents at the physical level. So, the difficulty of processing scientific data with MapReduce is manifested as a scalability limitation and arises disconnection between the logical views of data, and the physical layout of that data within a byte stream. So it is very difficult to maintain the table and may take extra cost for partitioning the information. Instead noDB can accept the raw data file as information and store in a flexible storage memory. It also provides the indices for raw data files for future query processing. Thus noDB has more reliable performance than bigtable for complex application.

4.5 Optimize query processing

Since,, without loading the data in noDB, it fires the query directly, will help to increase the processing compared to traditional database system. noDB has dynamic arrangement in dataset and does more suitable for distributed database system which has the significant issue in reliability and data replication. In Distributed DBMS, if a query involves data from many nodes, it may be possible to split the query into one or more subqueries that can be executed in parallel by many notes. This computation allows faster processing of queries. By replicating only the indices to all sites and can execute the query to the selected raw data will improve the query processing speed in DDBMS.

V. CONCLUSION

The philosophy proposed by the many authors remained depend on the traditional DBMS, just avoiding the data loading by using some technique like situ query, adaptive indexing on raw data files. But still it is can't give the optimum level in flexibility and performance when it comes to query processing, scalability and accuracy. There is no need to store data only will pipe raw data from websites, DBs, excel sheets etc, as input. This also means we can start adding more data as our promises isolate and ultimately provide structured data with this machine learning. The only worry is needed to do is to decide the best file system/extension and a flexible memory storage system to which data can be written. Finally, we can also use these flexible memory storage devices as the data backup for future purposes and avoid using traditional database systems. It will overcome the problem of bottlenecks of data loading, data analysis and complexity of data-to-query time. Our baseline experiment shows that there is the significant improvement in response time under environment of a noDB system. Some important reasons are-working on raw data, used of indexes, usage of data and memory as per the requirements. A series of the baseline experiment are executed to evaluate the Performance of this system: a. Data loading cost, b.Query processing timing, c. Avoidance of Collision and Deadlock

d.Enabling the Big data storage and e. Optimize query processing etc. The study found significant possible capabilities of noDB system over the traditional database management system.

Data loading cost is reduced because the NoDB loading allows to copied the data block from the flat files into the DBMS. Query processing timing is reduced. The noDB explicitly identify the data parts that are interested in with zero preparation or quick pre-processing. They do not need to replicate the data. It has much more flexible and optimistic than a traditional DBMS for inspecting and quickly initiating explorations for handing the large datasets. The enabling processes of Dicing and Slicing makes NoDb system to minimize the Collision and Deadlock scenarios. The number of such scenarios will be less of NoDB system as compared to traditional database systems. As the steps are very minimal the chances of the collision and deadlock are also very less. The noDB can accept the raw data file as input and store in a flexible storage memory giving comparison performance then bigtable for complex application.

III. REFERENCES

[1] S. Agrawal et al. Database Tuning Advisor for Microsoft SQL Server. In VLDB, 2004.

[2] N. Bruno and S. Chaudhuri. Automatic Physical Database Tuning: A Relaxation-based Approach. In SIGMOD, 2005.

[3] N. Bruno and S. Chaudhuri. To Tune or not to Tune? A Lightweight Physical Design Alerter. In VLDB,2006.

[4] K. Schnaitter et al. COLT: Continuous On-Line Database Tuning. In SIGMOD, 2006.

[5] G. Valentin et al. DB2 Advisor: An Optimizer Smart Enough to Recommend Its Own Indexes. In ICDE,2000.

[6] D. C. Zilio et al. DB2 Design Advisor: Integrated Automatic Physical Database Design. In VLDB, 2004.

[7] 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

[8] I. Alagiannis, R. Borovica, M. Branco, S. Idreos, and A.Ailamaki. NoDB: Efficient Query Execution on Raw Data Files. SIGMOD, 2012

[9] G. Graefe, S. Idreos, H. Kuno, and S. Manegold.Benchmarking adaptive indexing. In TPCTC, 2010.

[10] G. Graefe and H. Kuno. Adaptive indexing for relational keys. In SMDB, 2010.

[11] G. Graefe and H. Kuno. Self-selecting, self-tuning, incrementally optimized indexes. In EDBT, 2010.

[12] S. Idreos, M. Kersten, and S. Manegold. Database Cracking. CIDR, 2007.

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

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

[15] I.Alagiannis, R. Borovica, M. Branco, S. Idreos and A. Ailamaki.NoDB in Action: Adaptive Query Processing on Raw Data. In the Proceedings of the VLDB Endowment, 2012

[16] S.Richter, J.Quian´eRuiz, S.Schuh, J. Dittrich, Towards zero-overhead adaptive indexing in Hadoop,Technical report. Saarland University,2012

[17] M. Karpathiotakis.Adaptive Query Service for In Situ Querying. DIAS, I&C, EPFL,2013

[18] Parker-Wood, Aleatha, et al. Examining extended and scientific metadata for scalable index designs. Tech. Rep. UCSC-SSRC-12-07, University of California, 2012.

[19] Joe B. Buck, Noah Watkins, Jeff LeFevre, Kleoni Ioannidou, Carlos Maltzahn, Neoklis Polyzotis, and Scott Brandt. Sci-Hadoop: Array-based Query Processing in Hadoop. In Proceedings of SC, 2011.

[20] Liarou, Erietta, and S. Idreos. "dbTouch in Action: Database Kernels for Touch-based Data Exploration." Proceedings of the 30th IEEE International Conference on Data Engineering (ICDE), 2014

[21] Y. Wang, Y.Su, G.Agrawal, "Supporting a Light-Weight Data Management Layer over HDF5," ccgrid, pp.335-342, 2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, 2013

[22] Joe B. Buck Noah Watkins Jeff LeFevre Kleoni Ioannidou Carlos Maltzahn Neoklis Polyzotis Scott Brandt UC Santa Cruz ," SciHadoop: Array-based Query Processing in Hadoop." UCSC,2011.

[23] Buck, Joe B., et al. "Scihadoop: Array-based query processing in hadoop." Proceedings of 2011 Intl. Conference for High Performance Computing, Networking, Storage and Analysis. ACM, 2011.