A Tool for Statistical Analysis on Network Big Data

Carlos Ordonez*, Theodore Johnson, Divesh Srivastava, Simon Urbanek
AT&T Labs - Research, USA
*Research work conducted while visiting AT&T Labs, USA. C. Ordonez current affiliation: University of Houston, USA.

Abstract—**Due to advances in parallel file systems for big data (i.e. HDFS) and larger capacity hardware (multicore CPUs, large RAM) it is now feasible to manage and query network data in a parallel DBMS supporting SQL, but performing statistical analysis remains a challenge. On the statistics side, the R language is popular, but it presents important limitations: R is limited by main memory, R works in a different address space from query processing, R cannot analyze large disk-resident data sets efficiently, and R has no data management capabilities. Moreover, some R libraries allow R to work in parallel, but without data management capabilities. Considering the challenges and limitations described above, we present a system that allows combining SQL queries and R functions in a seamless manner. We justify a parallel DBMS and the R runtime are two different systems that benefit from a low-level integration. Our parallel DBMS is built on top of HDFS, programmed in Java and C++, with a flexible scale out architecture, whereas R is programmed purely in C. The user or developer can make calls in both directions: (1) R calling SQL, to evaluate analytic queries or retrieve data from materialized views (transferring result tables in RAM in a streaming fashion and analyzing them in R), and vice-versa (2) SQL calling R, allowing SQL to convert relational tables to matrices or vectors and making complex computations on them. We give a summary of network monitoring tasks at ATT and present specific programming examples, showing language calls in both directions (i.e. R calls SQL, SQL calls R).**

I. Introduction

Big data is characterized by the 3 Vs: volume, variety and velocity of data, where analyzing data is a central goal. It is fair to say that managing and analyzing network data is more difficult than other big data problems due to its streaming velocity, higher volume and format variety. That is, it has three more complicated Vs. Big data analytics is notoriously difficult. This problem becomes orders of magnitude harder with network big data due to its higher volume, streaming behavior and format varying over time. In this paper, we study how to perform statistical processing on a network database [4], integrating diverse data streams (not packet-level data, but network data summaries over time). Computer Science "systems" research has has proposed systems with optimized storage [10] for specialized processing based on rows, columns, and arrays [9]. Most common targets include transactions, queries, detecting patterns and computing mathematical models. In our work we focus on the last one. Streams represent a further challenge, where processing is pushed to main memory, with algorithms working in one pass. On the data mining side there are tons of research proposing algorithms for large data sets, but working mostly on flat files, outside a DBMS. However, integrating statistical systems, like R, with a database system is still a challenge. R is one of the most popular open-source system to perform statistical analysis due to its simple, but powerful, functional language, extensive mathematical library, and interpreted runtime. Unfortunately, as noted in the literature, even though every vendor offers some integration between R and the DBMS, R remains difficult to use and slow to analyze high-velocity streams. From a practical perspective, SQL remains the standard query language for database systems, but it is difficult to predict which language will be the standard for big data analytics: R has a proven track record. With that motivation in mind, we introduce STAR, a system to analyze network data integrating the R runtime with a parallel DBMS for big data supporting standard SQL queries and materialized views. Unlike other R tools and prototypes, STAR can directly process relational tables, truly performing "in-database" analytics. We emphasize that STAR enables analytics in both directions closing the analytic loop: (a) An R program can call SQL queries. (b) An SQL query can call R functions.

II. Related Work

We present an overview of database systems built at AT&T. GigaScope Tool [2] was a pioneer system that could evaluate a constrained form of SQL queries on packet-level data streams (i.e. very high velocity) as network packets were flowing in a network interface card (NIC). The main analytic goal was to analyze the probablistic distribution of the data stream based on histograms [1]. GigaScope had important limitations: it could not store streaming data, it did not take advantage of a parallel file system in a cloud infrastructure and it could not correlate streaming data with stored historical data. Therefore, as requirements changed it became necessary to store summarized historical stream data (orders of magnitude smaller than packet-level data, but orders of magnitude larger than transactional data) and supporting standard SQL became a requirement. Specifically, queries could have arbitrary joins (natural, outer, time band) and diverse aggregations (distributive, algebraic, holistic). Storing, managing and querying stream data was significantly more difficult than analyzing packet-level data, but it enabled advanced analytics to monitor the network. Such needs pushed the creation of the DataDepot Warehouse system [3], which featured a POSIX-compliant parallel file system, standard SQL and extensibility via UDFs [7], [5] (which enabled mathematical analytics). The DarkStar data warehouse at ATT, with DataDepot as the backbone system, can manage hundreds of data streams and maintains more than two thousand tables with real-time data loading and long-term histories. This network big data warehouse

2378-3915/17 $31.00 © 2017 IEEE
DOI 10.1109/DEXA.2017.23

supports networking analytics as well as real-time alerting and troubleshooting applications for ATT network day-to-day operations. In short, it is necessary to have access to real-time, recent and historical data. The big data trend brought more requirements and new technology: higher stream volume (with more data), HDFS (instead of a POSIX file system), many more database sources (more streams from more network devices) intermittent streams (with traffic spikes and transfer interruptions), more efficient C++ code for queries (because critical SQL queries were compiled), eventual consistency and advanced analytics beyond SQL queries. Given the common wisdom that one-size-does-not-fit-all [10] and the difficulty of changing the source code of a large existing system, it was decided to develop a next generation DBMS, TidalRace [4].

III. System Description

A. Parallel DBMS for Big Data: TidalRace

This section explains the main features of the parallel DBMS TidalRace [4], with a scalable architecture to process network big data. TidalRace [4] is a next-generation data warehousing system specifically built for data management of high volume network data, building on long-term experience from previous systems built at AT&T. The following paragraphs summarize the main features of TidalRace and its limitations for statistical analysis.

Storage: TidalRace is built on top of HDFS, to support scale out as data volume grows. Time partitions (a small time interval) are the main storage I/O unit for data streams, being stored as large HDFS blocks across nodes in the parallel cluster. The storage layout is hybrid: a row store for recent data (to insert stream records and maintain small materialized views), and a column store to query large historical tables with recent and old data (to evaluate complex queries). The system provides a Data Definition Language (DDL) with time-oriented extensions. TidalRace's SQL supports both atomic (i.e. standard) and structured data types (to connect to R). Atomic data types include integers, floats, date/time, POSIX timestamps and strings. POSIX timestamps are fundamental to create time partitions. Vectors and matrices are supported internally within UDFs in C++ and special SQL access functions. A major departure from traditional DBMSs is that the TidalRace DBMS supports time-varying schemas, where columns are added or deleted from an existing table over time. This unconventional "varying structure" feature is fundamental to keep the system running without interruption concurrently processing insertions from new file formats, critical queries and propagating updates to materialized views.

Language: The DBMS provides standard SQL enhanced with time-oriented extensions to query streaming tables. As mentioned above. TidalRace's SQL supports both atomic (i.e. standard) and structured data types (to connect to R). Its SQL offers both distributive aggregations (e.g., sum() and count()) and holistic aggregations (harder to compute, like rank, median, quantiles). User-defined functions (UDFs) are available as well: scalar and user-defined aggregates (especially useful to compute multidimensional statistical models [6]), programmable in the C language. Query processing is based on compiling SQL queries to efficient C code, instead of producing a traditional query plan in an internal representation, which allows optimizations only at compile time. Materialized views, based on SQL queries combine WHERE filters, joins and GROUP BY aggregations.

Processing: The database tables are refreshed by time partition, being capable of managing out-of-order arrival of record batches, intermittent streams and streams with varying speed (e.g. traffic spikes). That is, the system is robust to ingest many diverse streams traveling in a large network. The system uses MVCC (lock-free), which provides read isolation for queries when they are processed concurrently with insertions. The system provides ACID guarantees for base tables (historical tables) and database metadata (schema info, time partition tracking), and eventual consistency for views (derived tables). Therefore, queries, including those used in views, read the most up-to-date version, which is sufficient to compute queries with joins and aggregations on a time window. Query processing is multi-threaded, where threads are spawned at evaluation time by the query executable program. A key feature are materialized views, which are periodically updated when inserting records. Materialized views are computed with SQL queries combining WHERE selection filters on time partitions, time band joins (θ joins) and GROUP BY aggregations. In general, new records from base tables are propagated to materialized views with incremental computation. We emphasize that every query and view should have a time range, where such time range generally selects the most recent data. A major goal is to operate on a sliding time window with low latency. The DBMS operates with a minimal time lag between data stream loading (less than 1 minute) and querying (less than 2 minutes after loading) and efficiently propagates insertion of new records and removes old records to update materialized views (less than 5 minutes).

B. System Architecture and Streaming Processing

From the parallel DBMS TidalRace we get a parallel file system (currently HDFS, formerly a POSIX file system), stream data management and query processing. On the other hand, from the R side we get single threaded processing in main memory, text file I/O and rich set of mathematical operators and libraries.

C. Chunk-based Processing in RAM

We make two reasonable assumptions to process data streams based on a sliding time window: (1) the result table from an SQL query or materialized view having a time range filter generally fits in RAM. (2) the result data frame or output matrix from an R program can be divided into chunks (data blocks in RAM) and processed in streaming fashion. Our system incorporates optimizations to efficiently transform a relational table into an R data frame and vice-versa under the assumption that the data set fits in RAM. This is a valid assumption because the data set is computed by an SQL query

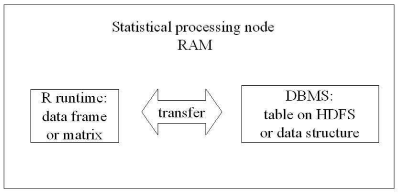

Fig. 1. Bi-directional analytic architecture: R ⇔ DBMS.

in a materialized view. That is, our system assumes data is pre-processed, transformed and summarized in SQL, not in R.

Based on these assumptions, our system provides a fully bidirectional programming API: an R program can call (evaluate) any SQL query and its results are seamlessly and efficiently transferred block-wise into R. Alternatively, SQL can call any R function via UDFs (embedding R calls into C code) and stored procedures (mixing queries with procedure calls). This bidirectional communication is achieved by a direct binding between R and DBMS runtimes in main memory, bypassing network communication protocols. Our integrated system architecture is shown in Figure 1.

The key issues to integrate R with a parallel Database System are understanding main memory management, layout of vectors and matrices in RAM, building data frames as a set of columns, setting up R function calls, access serialization and properly configuring the operating system environment. Main memory management is significantly different in both systems. R has a garbage collector and the runtime is single threaded. R can address main memory with 64 bits, but integers for subscripts to access data structures are internally 32 bits. On the other hand, the C++ in the DBMS uses a flat 64 bit memory space also with a single thread per compiled query, but no garbage collector. Therefore, each system works as a separate process with its own memory space. In addition, since both systems internally have different data structure formats it is necessary to transfer and cast atomic values between them. A fundamental difference with other systems, integrating R and a parallel data system is that building data structures and transferring them is done only in main memory, copying atomic values as byte sequences in most cases, moving memory blocks from one system to the other and avoiding creating files.

D. Mapping Data Types and Data Transfer

R and SQL exchange data with a careful mapping between atomic values. Data structures like vectors, matrices, data frames and tables are built from atomic values. Data structures include vectors, matrices and data frames on the R side and only tables (including materialized views) in SQL. To achieve maximum efficiency, transferring is always done as byte sequences: string parsing is avoided. We make sure a data frame only contains atomic values, thereby enabling converting data into an SQL table. Notice lists in R violate a database first normal form. Therefore, they cannot be transferred into the DBMS, but they can be pre-processed converting them into a set of data frames. Transferring in the opposite direction, an SQL table is straightforward to convert into an R data frame since the latter is a more general data structure. Converting an SQL table into an R matrix requires considering a sparse or dense matrix storage and how subscripts are represented in SQL. Finally, vectors and matrices in C++ are a mechanism to efficiently transfer and serialize data from the UDF to vectors and matrices in R (which have different storage and require memory protection), but not to perform statistical analysis. That is, they are transient data structures.

E. R calling SQL

Since R has a flexible script-based runtime it is not necessary to develop specialized C code to call an SQL query: the SQL query is simply called with a system command call. Transferring data from the evaluated SQL query to R is achieved via a packed binary record format that is converted into data frame format and then incrementally transferred to a data frame in RAM (via Unix pipes). This format resembles a big network packet, with a header specifying fields and their sizes, followed by a payload with the sequence of packed records. We note that since in a DBMS strings generally have variable length then records also have variable length. Therefore, conversion and transfer row by row is mandatory (instead of block by block), but it can be efficiently done in RAM, moving byte sequences. In the unusual case (because a time range is assumed) that the output SQL table does not fit in RAM the data set can be processed in a block-by-block fashion in R; the drawback is that many existing R functions assume the entire data set is used as input and therefore they must be reprogrammed. Finally, a data frame containing only real numbers can be easily converted to matrix. Therefore, it becomes feasible to call most R functions with a data frame or a matrix as input. Further math processing happens in R and in general R mathematical results (models, a set of matrices, diagnostic statistics) remain in R. However, when the R output is a data frame, preferably with a timestamp attribute, it can be converted to our packed binary format and then loaded back into the DBMS, possibly into a materialized view.

F. SQL calling R

SQL is neither a flexible nor an efficient language to manipulate data structures in main memory, but it offers UDFs programmable in C/C++ (called in a SELECT statement) and stored procedures (calling external routines). On the other hand, the most flexible mechanism to call R to perform low-level manipulation of data is to embed R function calls inside C (or C++) code. Since UDFs are C++/C code fragments plugged into the DBMS that isolate the programmer from the internals of physical database operators and memory management we use them as the main programming mechanism to

TABLE I
TABLES STORED ON THE DBMS.

Name	Type	Size
Device Measurement	base	large
Link utilization	base	large
Phone call	base	large
Connection	base	large
Feeds	base	large
Interrupted phone call	derived	large
Device measurement summary by minute	derived	medium
Link utilization by hour	derived	medium
Connection summarization	derived	medium
Feed summary by source/dest	derived	medium
High traffic site	derived	medium
Minutes used per phone	derived	medium
Quantile traffic approximate histogram	derived	small
Abnormal connections	derived	small

call R, bypassing files and network communication protocols. Specifically, calling R from the UDF C++ code is achieved by building temporary C++ vectors and then converting the set of C++ vectors into an R matrix. Notice we do not convert SQL records to data frame format in R because we assume the R function to call takes a matrix as input, the most useful case in practice. R results can be further processed in C++ inside the DBMS and potentially be imported back into a table. Only R results that are a data frame can be transferred back into an SQL table. In general, there exist materialized views which have a dependence on this temporary table. From a query processing perspective when the R result is a data frame the DBMS can treat R functions as table user-defined operators, where the size of the result is known or bounded in advance.

IV. STATISTICAL ANALYSIS ON NETWORK BIG DATA

A. Network Big Data Tables

As mentioned before, TidalRace stores a mix of tables to ingest stream data on base tables and periodically propagate changes to derived tables (materialized views). Table I shows tables going from stream ingestion to sophisticated data analytics.

B. Network Monitoring

As mentioned above, our system assumes every SQL query has a time range, which results in a sliding time window. Such time range represents the last x minutes in real time, where $1 \leq x \leq 60$ (with 1 minute being near real-time or so-called active data warehousing) or the last y hours where $1 \leq 24$ (with 24 hours being a worst case scenario with systemic patterns too hard to detect quickly). That is, our system enables monitoring the network for the most recent events with up to one hour delay, but not sooner than one minute. In general, it is not possible to guarantee an event is detected in less than one minute because streams must be transferred from diverse sources on the network all over the world, ingested into a single-point feed management system (Bistro [8]), transformed into quasi-relational files (because schema varies over time) and then transferred, distributed and stored on HDFS. Recall that these two last steps are performed by the TidalRace DBMS.

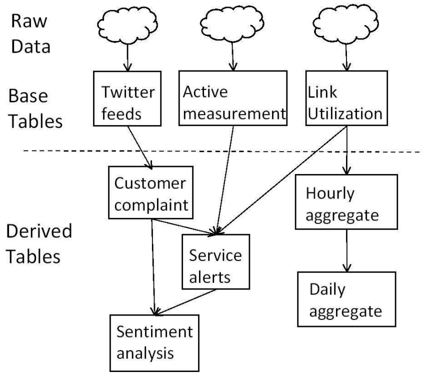

Fig. 2. Network big data analytics.

Figure 2 shows important end-user applications at AT&T, where base tables are periodically appended by time partition as streams are ingested and derived tables represent materialized views, periodically refreshed during low traffic periods.

C. Analytic Examples in Both Directions

R calling SQL: Assume there exists a long script with many SQL queries to derive a data set for statistical analysis. In a network data warehouse environment, such data set is periodically recomputed from a materialized view based on a sliding time window (e.g. every 5 minutes, every 30 minutes, every day). The resulting data set is produced by aggregating columns to create variables for statistical analysis in the R language. We contrast analytic calls on streams with three transfer mechanisms going from slowest (but most portable) to fastest (but ad-hoc): (1) JDBC connection, the standard DBMS protocol; (2) plain files exported from the DBMS and loaded into R; (3) binary files directly transferred in RAM to R via Unix pipes. By leveraging sufficient statistics maintained on the time window the analyst can call R functions to compute a predictive model such as linear regression (to predict a numeric variable) or classification (to predict a discrete variable). These analytic tasks boil down to developing an SQL script, starting the R language runtime, sending the SQL queries to the DBMS for evaluation, transferring the final SQL table for the data set into a data frame and then analyzing the R data frame with R operators and mathematical functions. We emphasize that in general the output of these calls cannot be easily and intuitively transferred back to the DBMS because they are a complicated collection of diverse vectors, matrices, arrays and associated diagnostic metrics (e.g. error, fit, and so on). That is, it is preferable they are managed by the R language.

SQL calling R: In this scenario we assume there is an experienced SQL user, with basic statistics knowledge, who needs to call R to exploit some mathematical function in a materialized view. In contrast to the previous case all processing takes place in main memory. A representative analysis is getting the covariance or correlation matrix of all variables. Moreover, these matrices are used as input to multidimensional models like PCA. Assume the user builds the data set with SQL queries as explained above, but the user wants to compute some model with data residing on the DBMS. To accomplish this goal, the user just needs to develop a "wrapper" function (aggregate UDF) that incrementally builds a matrix, row by row. Every tuple is dynamically converted to vector format in RAM. When the matrix is ready the desired R function is called in embedded C code. A second representative analysis is building a time series and smoothing the time series. In contrast to the previous example, the analyst stores the smoothed time series back into the DBMS. A common scenario is that the user wants to call R to solve the Fast Fourier Transform (FFT), a sophisticated mathematical computation definitely out of reach for SQL, to find the harmonic decomposition of the time series (i.e. from primitive time series) and identify its oscillation period (the one with strongest correlation to some specific primitive time series). When the period has been determined time series values are averaged (smoothed) with a sliding time window. The result is a transformed time series that is much easier to interpret and further analyze because it has less noise and a periodic pattern has been identified. We assume the input table has a timestamp and some numeric measure. Then a DBMS user-defined function (UDF) dynamically builds a data frame in RAM and then it calls R to get an output data frame. Finally, this transformed time series is efficiently transferred back into the DBMS as a stream, but only in RAM. That is, the stream data records do not touch disk on the R side.

D. Throughput and Performance

Two major goals are to monitor devices and monitor connections. To monitor devices, in general, each record contains at a minimum 3 attributes: a timestamp, a device id or network address, and some measurement (i.e. bytes per second). To monitor connections records are used to track data transmission, which requires more attributes: source and destination (IP addresses), protocol, transmission time, received time, throughput metrics and status. Summarizing, network data sets have between 3 and 10 attributes. In other words, they are relatively narrow, but extremely large. It is assumed SQL queries reduce such size orders of magnitude because they compute summarizations. Therefore, in general the data sets for statistical analysis in R fit in RAM.

Given our system efficiency, high-end hardware and simplicity we did not conduct a detailed performance study. Our system is capable of transferring between n =1M and n =10M records per second from the DBMS to R and vice-versa on average hardware (e.g. a Quadcore CPU, 8 GB of RAM). This time excludes the actual time to compute an SQL query or update a materialized view. In an analog manner, this time excludes the time to call R on a data frame or matrix and transform, return results as an SQL table and loading them back into the DBMS. Analyzing overall time to compute statistical models or transformations on diverse network monitoring problems is an issue for future research.

V. Conclusions

We presented a system that enables fast bi-directional data transfer between a parallel DBMS and the R runtime. In one direction our system converts SQL relational tables into R data frames or matrices. On the opposite direction an R data frame or matrix is converted into a relational table, with a transformed data frame being the most common case. Our system is built on top of a careful mapping between atomic data types. The system efficiently constructs data structures (i.e. non-atomic data types) in RAM in one pass over a data set. The net gain is that an R script can call an SQL query or materialized view to analyze the result set. On the other hand, an SQL query (not a script or longer embedded SQL program) can call an R function to perform some mathematical computation in an intermediate step.

Our initial prototype opens several research directions. We want to define functional constructs in the R programming language to transform relational tables into data frames. In a similar manner, we want to study alternatives to transform a matrix into an SQL object (flat table, subscript/value triples, or binary object). Propagating insertions to materialized views and then to a mathematical model computed by R is a challenging problem. Finally, we need to conduct a detailed performance study on the ATT network data warehouse.

References

[1] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava. Space- and time-efficient deterministic algorithms for biased quantiles over data streams. In *Proc. ACM PODS*, pages 263–272, 2006.

[2] C. Cranor, T. Johnson, O. Spataschek, and V. Shkapenyuk. Gigascope: a stream database for network applications. In *Proc. ACM SIGMOD*, 2003.

[3] L. Golab, T. Johnson, J. Spencer Seidel, and V. Shkapenyuk. Stream warehousing with DataDepot. In *Proc. ACM SIGMOD*, pages 847–854, 2009.

[4] T. Johnson and V. Shkapenyuk. Data stream warehousing in Tidalrace. In *CIDR*, 2015.

[5] C. Ordonez. Building statistical models and scoring with UDFs. In *Proc. ACM SIGMOD Conference*, pages 1005–1016, NY, USA, 2007. ACM Press.

[6] C. Ordonez. Statistical model computation with UDFs. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 22(12):1752–1765, 2010.

[7] C. Ordonez and J. García-García. Vector and matrix operations programmed with UDFs in a relational DBMS. In *Proc. ACM CIKM Conference*, pages 503–512, 2006.

[8] Vladislav Shkapenyuk, Theodore Johnson, and Divesh Srivastava. Bistro data feed management system. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 1059–1070. ACM, 2011.

[9] M. Stonebraker, P. Brown, D. Zhang, and J. Becla. SciDB: A Database Management System for Applications with Complex Analytics. *Computing in Science and Engineering*, 15(3):54–62, 2013.

[10] M. Stonebraker, S. Madden, D. J. Abadi, S. Harizopoulos, N. Hachem, and P. Helland. The end of an architectural era: (it's time for a complete rewrite). In *VLDB*, pages 1150–1160, 2007.