VQA: Vertica Query Analyzer

Alkis Simitsis
HP Labs
Palo Alto, CA, USA
alkis@hp.com

Kevin Wilkinson
HP Labs
Palo Alto, CA, USA
kevin.wilkinson@hp.com

Jason Blais
HP Vertica
Campbridge, MA
jason.j.blais@hp.com

Joe Walsh
HP Vertica
Campbridge, MA
joe.e.walsh-jr@hp.com

ABSTRACT

Database query monitoring tools collect performance metrics, such as memory and cpu usage, while a query is executing and make them available through log files or system tables. The metrics can be used to understand and diagnose query performance issues. However, analytic queries over big data presents new challenges for query monitoring tools. A long-running query may generate tens of thousands of values so simply reporting the metrics may overwhelm the user. Second, analytic queries may be written by database novices who have trouble interpreting the metrics. Third, analytic queries may access data or processing outside the database through user-defined functions and connectors. The impact of these on query performance must be understood. Vertica Query Analyzer (*VQA*) is a query monitoring tool to address these challenges. *VQA* is both a useful tool and a research platform for query analytics. It presents query performance metrics through a variety of views and granularities. In addition, it analyzes the metrics for typical performance problems and suggests corrective actions. We demonstrate *VQA* using TPC-DS queries which have a wide range of query duration and complexity.

Categories and Subject Descriptors

H.2.m [**Database Management**]: Miscellaneous

Keywords

SQL; SQL query monitoring; SQL query analysis; parallel database

1. INTRODUCTION

This demonstration addresses the question: '*Why is my analytic query so slow?*' Most database systems include monitoring tools that enable administrators and users to view performance metrics for individual queries. Typically, the performance metrics are stored into log files or system tables that can then be read by an application and displayed on a dashboard or summarized in a report. However, for analytic queries over big data, we believe this approach is lacking in several respects. First, the user may be a data scientist or analyst for whom the low-level metrics provided by the monitoring tools are incomprehensible. Second, a long-running query on a parallel database engine may generate tens or hundreds of thousands of metrics. Simply reporting this data is not helpful. A user needs to find the useful information. Third, complex analytic queries may invoke user-defined functions (UDFs) or use connectors to other processing engines. Their impact on query performance must be understood.

As a first step in addressing these challenges, HP Labs, in collaboration with Vertica, have developed a research platform, the Vertica Query Analyzer (*VQA*). The Vertica database engine is an excellent testbed for this project. Vertica is designed for analytic workloads and it employs a high degree of intra-query parallelism. This combination of high query complexity and large scale parallelism presents many opportunities for studying problem queries. This paper starts with an overview of the *VQA* architecture, then describes the features to be demonstrated, and then provides an outline of our planned presentation to SIGMOD attendees.

2. SYSTEM OVERVIEW

At a high-level, *VQA* executes a query, collects performance metrics and presents those metrics in a variety of representations and perspectives. It also provides a number of monitoring and analysis actions. *VQA* features are described in the next section.

VQA comprises a user interface module and a server module (Figure 1). The interface module accepts a SQL query, sends it to the server module, collects performance metrics from the server module and renders charts and graphics of those metrics. An *external* query, i.e., one executed outside *VQA*, may also be analyzed by entering its Vertica query identifier. The interface is Web-browser based and uses HTML and various graphics libraries for display, and Ajax and JSON to communicate with the server module.

Figure 1: *VQA* architecture

The server module submits queries for execution to Vertica and retrieves query execution performance metrics. Per user request, metrics may be collected periodically during execution or after the query completes. The server also retrieves and parses the query explain plan and checks for query execution events (e.g., hash table overflow). The server is implemented in Jetty and Java and uses JDBC to access Vertica. It may run on a user machine, a Vertica node or an application server.

SIGMOD'14, June 22–27, 2014, Snowbird, Utah, USA.

ACM 978-1-4503-2376-5/14/06 ...$15.00.
http://dx.doi.org/10.1145/2588555.2594531.

Vertica can collect detailed (thread-level) performance metrics for the operators of a running query. Some metrics are common to all operators (e.g., CPU, memory usage, rows produced) and others are operator-specific (network bytes sent). A large analytic query may comprise a few hundred logical operators, each of which may comprise a number of multi-threaded, physical operators, each running on multiple nodes. So, a long-running, parallel query for which metrics are collected every few seconds may easily generate tens to hundreds of thousands of metric values.

Vertica stores query performance metrics in a system table. The *VQA* server uses a SQL query to read the metrics for a query into its main memory. To reduce memory usage on the server, by default, only a subset of all possible metrics are collected. But, the interface allows users to request more or fewer metrics as needed. Performance metrics for a query are not retained indefinitely by Vertica or *VQA*. However, a user may request that *VQA* log query metrics to disk for later review or off-line analysis.

3. DEMONSTRABLE FEATURES OF VQA

VQA provides four primary functions to help users understand query behavior: *Entry*, *Monitor*, *Analyze*, and *Action*. The first step is to enter the query to be studied. Monitor refers to collecting and viewing performance metrics for a query, either during its execution or after. Analyze runs a set of data mining algorithms over the metrics looking for patterns of performance problems. And, *VQA* provides users a number of actions on the metrics or the query. This section describes these functions in more detail.

3.1 Entry

There are three use cases for *VQA*. The first is when a user enters the SQL text for a query. Here, *VQA* initiates query execution and so *VQA* may collect performance metrics for the entire query execution. The second is when a user wants to monitor an external query, e.g., to study a query that is blocked or running too long. Here, the user enters its Vertica query identifier and *VQA* can collect performance metrics only from that point until the end of execution. Third, a user may wish to retrieve metrics from the *VQA* log for a previously monitored query for review and analysis. Figure 2(top-left) shows a fragment of the query entry interface.

3.2 Monitor

VQA provides several options for when to collect metrics for a query. First, a user may collect an immediate snapshot of a query's performance metrics to study the current query state. Subsequent snapshots may be collected at the user's discretion. Second, a user may let a query run to completion and then collect a final snapshot of the performance metrics. Third, a user may have *VQA* automatically collect periodic snapshots in order to view the dynamic behavior of the query. In addition, a user may specify which metrics to collect. *VQA* collects a default set of metrics for each operator, but the user may add or delete from that set. For each snapshot of metrics, *VQA* provides chart views and tree views.

3.2.1 Chart Views

VQA provides a number of two-dimensional charts to view a metric snapshot. These charts are refreshed for each new snapshot collected during query execution. Typically, there is one chart for each type of metric, but some charts show multiple metrics for comparison; e.g., estimates vs. actual measures. The Vertica engine collects metrics for each thread of each physical operator for a query. But the user view of query execution is the tree of logical operators (termed *paths* in Vertica) shown by the SQL explain plan command. Each logical operator (e.g., GroupBy) comprises a number of physical operators (e.g., ExpressionEval, HashGroupBy). And a physical operator may run as multiple threads on a node (e.g., a parallel table scan). Additionally, Vertica is a parallel database, so a physical operator may execute on multiple nodes. Consequently, each logical operator in the tree may correspond to many metric values at the physical operator level.

The snapshots collected for a query form a multi-dimensional, hierarchical dataset where the dimensions are path/operator, node, time, and the cells contain metric values. The metrics may then be aggregated (rolled-up) at the physical operator level, the logical operator (path) level, the node level, or the query level. To reduce the level of detail, the charts initially display metrics aggregated at the node level. The user may selectively drill down to see more detail. Clicking on a node displays all paths on the node and clicking on a path displays all operators for that path. Different colors are used to distinguish operators/paths that are currently executing from those that have completed. Pending paths and operators that have not started are not shown to simplify the charts.

Figure 2(bottom-left) shows an Elapsed Time chart drilled-down to the operator level for path 6 on node 3. It shows the time when each operator starts and ends execution. Charts for other metrics are available, e.g., Time (cpu usage, clock time), Memory (reserved and used), Rows (estimated and produced), and so on. Query execution events (e.g., group_by_spilled, no_histogram), if any, are listed on the affected path.

To view different perspectives, a user may aggregate along different hierarchies, e.g., path first, then physical operator and then node. Note that for some metrics, the range of values may be several orders of magnitude, e.g., memory usage for a filter operator compared to a sort operator. Consequently, by default, *VQA* will not display operators or paths with insignificant metric values. In this way, less useful information is not exposed to the user. Alternatively, a user may choose a logarithmic scale for a chart.

When a monitored query completes execution, a History chart is available that shows metrics aggregated at the query level at each timestamp where a snapshot was collected. This helps in seeing trends and correlations in the metrics. Figure 2(right) shows an example History with snapshots taken approximately every four seconds. This timeline chart is designed to give a coarse overview of the query execution to check for obvious anomalies.

3.2.2 Tree Views

In a tree view, *VQA* first displays a query's explain plan as a graphical tree of operators. Then, as metric snapshots arrive, a query's progress is shown by annotating the operator tree. Each node in the operator tree represents a path (logical operator) in the query and arcs between nodes represent dataflow. Within a tree node, bar charts indicate relative values for metrics: e.g., execution time, memory allocated, and rows produced. Each bar chart has its own scale and shows the fraction of total query metric value consumed or produced by this operator. Additionally, the color of the tree node indicates the execution status of the path, i.e., waiting (not started), running or completed. Tool tips over nodes and arcs provide additional details about the operator or data.

Figure 3(left) shows the tree view for a query which has a hash join as its root operator (#2). As can be seen, its right child (#15) has finished, indicating the hash table has been built. The second input (#3) has just started executing. We can also see that, currently, operators #6 and #17 use most of the memory in this query and operators #6, #15-#17 use most of the CPU. It appears anomalous that some operators have finished while their child operators are still running, e.g., operator #16 and operator #17. This is an artifact of collecting the metrics in real-time across multiple Vertica nodes.

Figure 2: *VQA* entry (top left), chart view (bottom left), and history (right) for an execution of the TPC-DS Query-2

Metrics for some operators may be delayed but eventually things will converge. If an operator is shown completed it is safe to infer its input operators have also completed.

One challenge with a graphical display of the query tree is that the screen space needed to render a large tree may exceed what is available. *VQA* addresses this in two ways. First, nodes in the tree may be hidden when they have low information value or are not active, much as the chart views hide operators and paths with little or no data. Second, a synopsis of the entire query tree is displayed in a fixed-sized box below below the legend on the left. The synopsis is active in that clicking on a portion of the tree in the synopsis shifts the screen to display that portion of the query tree.

Note that a user may reorient the tree display (e.g., left-to-right rather than top-down) to better use the available space depending on the shape of the tree (e.g., especially broad or deep). Finally, on the left of the display above the tree synopsis, is shown a query progress indicator. It estimates the completion percentage of the running query using techniques like those in [4].

3.3 Analyze

The Analyze function may be invoked after metrics are collected for a query. It invokes a series of heuristic algorithms that mine the query metrics for patterns of performance problems.

As a simple example, consider a scan of a hash partitioned table. Ideally, the number of result rows will be approximately equal across all nodes. Otherwise, there is skew in the data distribution and the amount of work done per node will vary at this level and possibly higher levels in the query. Similarly, a parallel hash join may repartition its input which could result in skew. So, differences in execution or elapsed time may be accounted for by skew.

Another important analysis function is identification of critical paths in query execution. By critical path, we mean those paths with the highest impact on query execution or those responsible for deviations between estimated and actual performance. Knowing the critical paths may reveal query design or execution problems and can help users focus on the most important areas to improve performance. *VQA* can inform the user in a variety of ways: through the analysis report, annotations on the explain plan tree or highlighting the SQL code fragment responsible for the critical path. Alternatively, for users not comfortable with SQL, a natural language description of the SQL code fragment may be generated using techniques as in [3].

Figure 3(right) illustrates example sections of an Analyze report for a query, listing for example, the critical paths, scan skew, accuracy of estimated rows, and the execution time of a user-defined function. The analysis function has a plug-in architecture so it is easy to add algorithms for different performance patterns.

3.4 Actions

VQA provides a number of actions on both the collected metrics for a query and the operator tree. Recall the History chart shows the timestamps when metrics were collected for a query (see Figure 2(right)). A user may click on any timestamp and *time-travel* to that point in the query's execution, i.e., the chart and tree views will reflect the state of the query at that point in time. From that point, a user may *replay* a query execution either forward or backward at a user-specified speed. A user may *save* the metrics snapshots for a query execution to a log file. In addition, a user may *share* those metrics with another user (e.g., customer support) or later *get* the metrics for replay or for comparison with another execution of that query.

Note that a long-running query may require a large memory footprint, so a user may *compress* the metric snapshots for a query execution. An upper bound may be placed on the *VQA* server memory used to store metric values for a single query. If that is exceeded while monitoring a query, a deletion algorithm is used to selectively

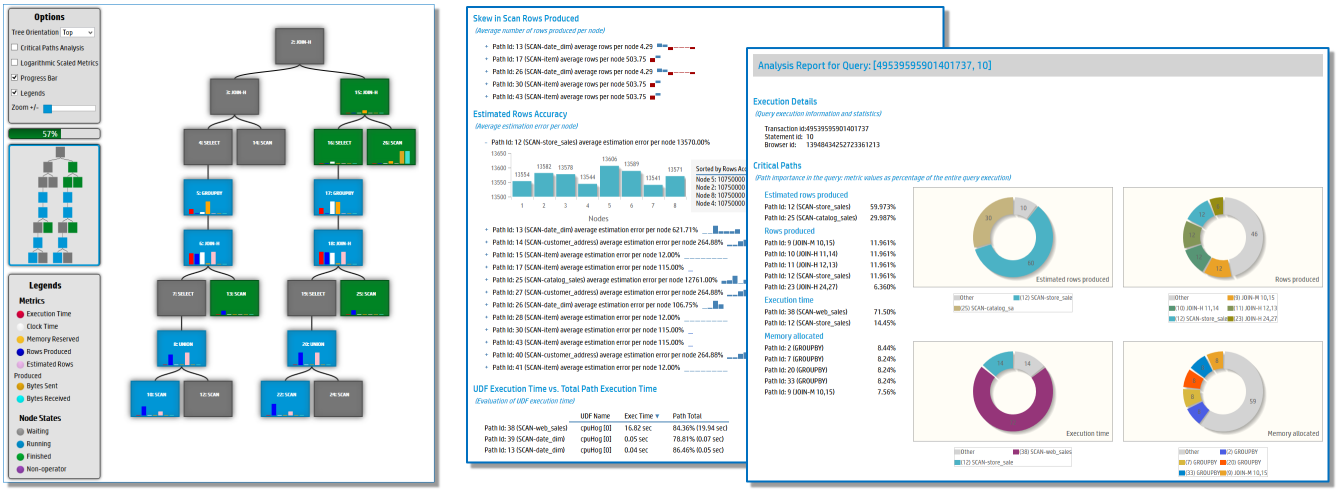

Figure 3: *VQA* tree view (left) and analyze (right)

remove metrics with little information content. Essentially, it *samples a multivariate time series in real-time* but does so in a way that retains approximately equal time intervals between retained snapshots and also attempts to retain high-value data; e.g., snapshots that include minimum or maximum performance metrics or that represent an inflection point.

4. OUR PRESENTATION

Our presentation will demonstrate *VQA* features using TPC-DS [5] benchmark queries running on a multi-node Vertica cluster. These queries have a wide range of execution times and query complexity so they provide many options for illustrating features. As a background demonstration, we will have *VQA* loop through the TPC-DS queries, monitoring the execution of each while showing the operator tree. This will show a variety of query tree shapes and animate their execution.

Our demonstration script begins by monitoring a relatively short TPC-DS query that is easy to understand. Various metric charts will be shown, e.g., different levels of aggregation, different chart types, and so on, along with drill-down to different levels of detail. Since the ultimate goal is to understand performance problems, an artificial load will be placed on one Vertica node and the query monitored again. The charts will then show significant CPU latency on the loaded node.

The next demonstration will monitor a longer-running query that undergoes several phase changes (alternating periods of high CPU, memory and disk usage). During execution, the charts will be refreshed automatically showing how different operators and paths use resources and produce rows. When the query completes, the History chart will be displayed to show the phase changes over time as inflection points in the timeline of metric values. We will then click on one inflection point to go back in time and view the metrics at the time of the inflection point. We then repeat the monitor action with an artificial, memory-intensive load on one node.

The tree view will be demonstrated with a small query tree that easily fits on the screen, but that runs for about 10 seconds. This will show the basic animation features and the progress indicator. Next, we will show a variety of very large query trees and show how the synopsis tree can be used to navigate to different fragments of the query tree.

The analyze function will be demonstrated by monitoring the TPC-DS queries. As an example, we will use a TPC-DS query that is slightly modified to invoke a user-defined function that is a cpu hog. The operator tree view will show the large fraction of execution time devoted to the operator that invokes the user-defined function. Next, the analyze function will be invoked to produce a report showing, for example, critical paths, accuracy of estimated values, scan skew, and join skew.

Finally, for off-script presentation and discussion, we will provide interactivity, where the participants can browse example queries and experiment with *VQA*.

5. RELATED WORK

The Vertica Management Console (MC) provides a comprehensive, real-time view of the state of the database and includes database and node management functions [2]. MC also includes some functions adapted from *VQA*, e.g., users can view query explain plans, profile query execution, and view aggregate query performance metrics for each logical operator.

We also note that several other database systems provide some of the features of *VQA*. Probably the most closely related to *VQA* is Stethoscope [1]. But we know of no system that works with a parallel database engine and offers a real-time, interactive analytics of query performance metrics at a fine granularity like that in *VQA*.

6. ACKNOWLEDGMENTS

The authors thanks Jorge Saldivar Galli (Univ. of Trento), Chris Sulawko (HP Vertica), Craig Sayers (HP Labs), Fernando Martinez (HP GUAPO), Samuel Heaney (ITESM), and Andres Trevino (ITESM) for their help in various stages of *VQA*.

7. REFERENCES

[1] M. Gawade and M. L. Kersten. Stethoscope: A platform for interactive visual analysis of query execution plans. *PVLDB*, 5(12):1926–1929, 2012.

[2] HP Vertica. Vertica Management Console, v. 7.0.x. Available at: https://my.vertica.com/docs/7.0.x/HTML/index.htm#Authoring/ConceptsGuide/Components/ManagementConsole/ManagementConsole.htm, 2014.

[3] G. Koutrika, A. Simitsis, and Y. E. Ioannidis. Explaining structured queries in natural language. In *ICDE*, pages 333–344, 2010.

[4] J. Li, R. V. Nehme, and J. F. Naughton. Gslpi: A cost-based query progress indicator. In *ICDE*, pages 678–689, 2012.

[5] TPC. TPC Benchmark DS, v. 1.1.0. available at: http://www.tpc.org/tpcds/spec/tpcds_1.1.0.pdf, 2012.