International Journal of Enhanced Research in Science Technology & Engineering, ISSN: 2319-7463
Vol. 3 Issue 1, January-2014, pp: (186-193), Impact Factor: 1.252, Available online at: www.erpublications.com

Query Progress Indicator for Open Source Database

Urmila Mane[1], Prof. L. V. Patil[2], Dr. S. D. Joshi[3]
Department of Information Technology, SKN College of Engineering, Pune, India

Abstract: Nowadays mostly database applications are based on long-running and complex queries. So it will be helpful for users to have information about progress of query execution. Recently development of percent done progress indicators has been increased. For this purpose in this paper we propose such progress indicator for multiple concurrently running queries. The main focus is on providing progress indicator for Postgres database.

Keywords: ACID, BI, DW, GUI, PostgreSQL, RDBMSs, SQL, UNIX.

Introduction

Any query processing mechanism consists of range of activities while extracting data from database [1]. These activities include translation of queries in high-level database languages into expressions that can be used at the physical level of any file system and a variety of query-optimizing transformations along with actual evaluation of queries. The steps involved in this processing of any query are parsing and translation, optimization, and evaluation [1]. The cost of query evaluation can be measured in terms of a number of different resources such as disk accesses, CPU time for query execution and in distributed or parallel database system the cost of communication. Out of these the response time required for query evaluation plan that means the clock time required to execute plan would account for all these costs and also could be used as a good measure of cost of plan [1].

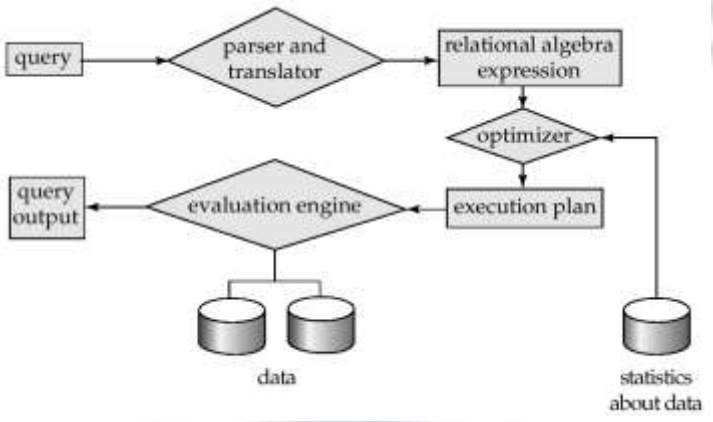

Figure 1. Query Processing Steps [1]

Progress indicators have been studied in various contexts (typical example is file transfer or file download) but there exists very limited work on this topic in case of data management context. In day to day life a typical progress indicator is used to estimate how much of the task has been completed and when the task will finish. In recent years, there has been increasing interest regarding development of progress indicators for SQL queries. A progress indicator in case of database queries is used to estimate precisely the value of a function that is related to the progress towards completion of a running query. For this purpose availability of such indicators can be of great help both to database administrators and end users [5]. Given the complexity of any query in decision support or data warehousing applications, it is common for queries to take hours or days to terminate. During such cases, these indicators can greatly aid a user's understanding of the progress of a query towards completion and allow the user to plan accordingly for example [5], terminate the query and/or change the query parameters. Also from the point of view of administrators, unsatisfactory progress of queries may point to bad plans, poor tuning or inadequate access paths.

Many modern software systems nowadays provide progress indicators for long-running tasks. These progress indicators aim to make systems more user-friendly by helping the user quickly estimate how much of the task has been completed

and when the task will finish. But already existing commercial RDBMSs provide progress indicator for long running queries which were not easy to prove.
Percent-done progress indicators basically used as a technique that graphically shows query execution time that means total and remaining or degree of completion [5]. Also the progress indicator in proposed technique is based on PostgerSQL database engine. Currently PostgreSQL doesn't have SQL query progress indicator for long-running queries. With the help of user-system interaction (interface) the progress indicator show the progress of SQL queries through various phases like parsing, analyzing, rewrite, execution. The graphical user interface show all the queries running on system and their estimated time completion. The execution phase of query is critical phase and also the cost of query varies depending disk read time, type of join used, distribution or broadcast of table, order in which tables are joined, statistics information available.

Why use Postgre SQL?

PostgreSQL database is an object-relational open source database system. It is having strong reputation for data integrity, correctness and reliability. It also has more than 15 years of active development. It is fully ACID compliant that means it assures all database characteristics such as Atomicity, Consistency, Integrity and Durability. It is including most SQL: 2008 data types. It has full support for triggers, joins, views, foreign keys, and also for stored procedures in multiple languages. It also supports storage of sounds, video, pictures, or binary objects. It is having programming interfaces for Java, .Net, C/C++, Ruby, ODBC, Python, Perl, Tcl.

Though the standard distribution of postgreSQL contains command-line tools for administrating database but it does not contain any graphical tools. In open-source and commercial alternatives there exist graphical tools for the purpose of administration and also tools for database design as well as commercial forms design and report generation tools. But there is no any graphical user interface tool indicating the progress of currently executing query through all its phases of execution.

The rest of the paper is organized as follows: Section II describes related work regarding the topic. Section III discusses proposed optimizer-based query progress indicator. Section IV gives experimental evaluation. Section V concludes the paper. Finally section VI describes future enhancement regarding this topic.

II. Related Work

Following table gives comparative existing work regarding this area of topic.

Table1. Advantages and Disadvantages of Existing Work

Tool	Purpose	Implementation Technique	Advantages	Disadvantages/Limitations
Toward a Progress Indicator For Database Queries (referred as WiscPI)	For implementing simple but useful progress indicator for large subset of RDBMSs queries.	-Collect statistics at selected points of query plan. -Monitor continuously query execution speed. -Unit of Work is one byte processed.	-Gives continuous accurate estimated query execution time. -Monitors progress of rollback operation. -From time to time, it presents latest estimates to user.	-For long-running aggregate queries, online aggregation provides no estimate of the remaining query execution time. - Also, no estimate of the remaining query execution time or the percentage completed is provided in dynamic query optimization, as the refining the query cost is not continues.
Estimating Progress of Execution For SQL queries (referred as MSRPI)	Estimating percentage remaining (or equivalently completed) of query at any point during its execution. -Reporting a "Progress bar" for query execution.	-The GetNext() model of work (MSRPI calculates % of GetNext() calls finished as an estimation of current query progress). -Progress estimation based on GetNext model. -Unit of Work is one GetNext() call.	-Such an estimator is simpler than estimating time remaining since it is independent of other queries (i.e., MSRPI is simpler than WiscPI in case of implementation.).	This estimator does not deal with remaining time while dealing with percentage remaining or completed.
Increasing The Accuracy And	Consider problem of supporting non-trivial progress	-Technique to improve accuracy of estimates. -Technique to provide new functionality.	-Progress indicator can profit from defining segments at a finer granularity.	-It is a non-trivial task to make hybrid method for handling correlated sub-queries work at a reasonable overhead.

Coverage Of SQL Progress Indicators	indicator for a wider class of SQL queries with precise estimates.		-Simple approach of using the optimizer's estimate of whether segment is CPU or I/O bound can substantially increase the accuracy of progress indicator.	-This approach doesn't deal with supporting progress indicator for SQL queries in ORDBMSs. -This approach doesn't investigate how to support progress indicator for SQL queries in parallel DBMSs. -Fails to prove handling of skew on different data server nodes.
Multi-query SQL Progress Indicator.	Consider concurrently running queries and even queries predicted to arrive in future when producing its estimates.	-The RDBMSs processes work units at constant rate C (work units per second) that is independent of number of running queries. -The progress indicator has perfect knowledge about remaining cost C_i of each running query Q_i. -Queries execute at speed proportional to weights associated with their priorities.	-Extends use of progress indicator beyond being a GUI tool. -Shows how to apply multi-query progress indicator to workload management. -Provides more accurate estimates than single-query progress indicator. -Considers impact queries have on each other's progress and eventual termination.	In workload management environment one does not want to sacrifice resource utilization ratio in any RDBMS. Queries may incur substantial I/Os and run for long time.
GSLPI: a Cost-based Query Progress Indicator.	Implement MSRPI and WiscPI both progress indicator in same RDBMS and propose new progress indicator without uniform speed assumption.	-Decomposition of execution plan into set of speed-independent pipelines. -Utilization of wall-clock pipeline cost to represent cost of pipeline. -Estimation of speed of each future pipeline based on its wall-clock pipeline cost.	-They present deeper insight into query's execution. -Due to this it directly affects prediction accuracy. -Lays down foundation for further development of progress estimation.	-GSLPI doesn't deal with parallel database systems regarding some additional challenges like data skew, new operators. -GSLPI doesn't focus on multiple concurrently running queries and also regarding utilization of information provided by progress indicator for better workload and resource management.

III. Proposed System

A. General Features of Progress Indicators

The proposed system is having the following features To provide enhanced feedback to the user/DBA on how much of a SQL query execution has been completed i.e. phase of the query and how long it will take for query execution.

- **Multiple Query Progress Graph Display**: The system is designed to handle and display multiple queries progress in form of graphs. The graphs can be disguised by the distinct transaction-id and XY-Line color. The transaction-id is unique local transaction-id given by postgresql for every query.
- **Estimated Time for Query Completion**: The system gives the estimated time for query completion. The estimated time is dynamic i.e. it varies depending on the system load, resource etc.
- **History of Committed Queries**: The system is also featured with query history. It shows both last committed query and the list of committed queries.
- **Dynamic Variation of Y-Axis**: The Y axis is the time axis and is dynamic in nature as query completion time for different queries is different i.e. one query may commit early and other may long time to complete.
- **Client-Server Implementation**: The system is implemented in 2 tier architecture i.e. client-server .From client side user can fire the query and GUI of query progress will be at client side. At server side query execution is done by the database.

B. Model and Implementation

The architecture shown below, describes how the different components of the system interact and there working collaboratively to achieve the desired functionality of the system. The system mainly consists of user/dba, postgresql database, and the GUI which shows the progress of the query and all these components interact with each other.

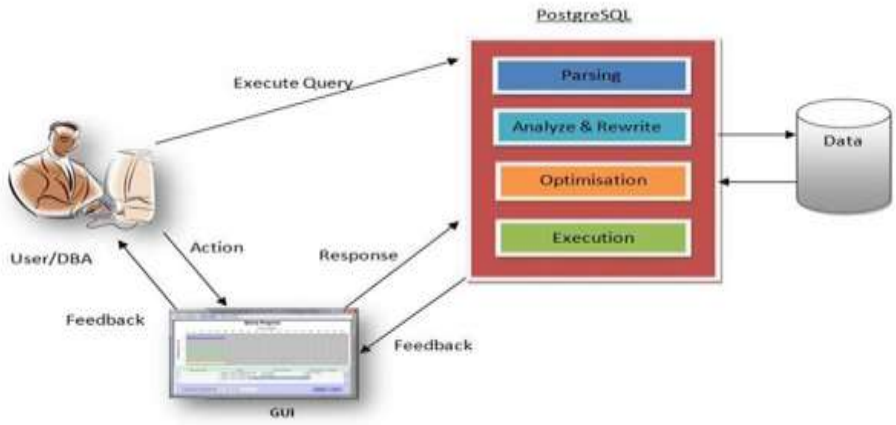

Figure 2. General Architechture of Proposed system.

When user/DBA fires a query then it passes through different phases i.e. parsing, analyze, rewrite, planning, execution of postgresql and at every phase it gives the feedback to the user/dba through the GUI .The feedback is about how much percent of query is completed , how long it will take for query to run to its execution. Also the user/DBA can interact with the GUI during execution by aborting the query in between and the DBA can see at what percentage of the query it is aborted. Aborting the query in between will not harm the data as the kill signal is sent which cause the shutdown of query execution i.e. data integrity is maintained. Effect is only reflected into the database when the execution of the query is complete. GUI also handles the history of committed queries.

1) SQL Query Execution Plan-background

SQL divides a given query plan for each query it receives [1]. The right plan is chosen so as to match the query structure and the properties of the data. It is absolutely critical for good performance of any system. For this purpose the system includes a complex planner. This complex planner tries to choose good plans.
The query plan consists of tree of plan nodes. In this tree of plan nodes, the bottom level nodes are table scan nodes. They return raw rows from a table. There exist different types of scan nodes required for different table access methods such as sequential scans, index scans, and bitmap index scans. If the given query is requiring aggregation, joining, or sorting or other operations on the raw rows, then there will be some additional nodes "atop" the scan nodes which are used to perform these operations. Also there will be usually more than one possible way to do these types of operations, so different node types can also appear here.

The obtained costs will be measured in arbitrary units. These arbitrary units will be determined by the planner's cost parameters. Traditionally, cost is measured in units of disk page fetches that means sequential page cost is always set to 1.0 and then the other cost parameters are set relative to that.
It is necessary to take one thing into account that the cost of an upper-level node always includes the cost of all its child nodes. It is important to note that the cost only reflects things that the planner is caring about. The cost does not consider the time which is spent transmitting result rows to the client because this could be an important factor in the true elapsed time; but the planner ignores it because it cannot change it by altering the query plan. It is assumed that every correct plan will output the same row set.

Rows output is usually less because it sometimes reflects the estimated selectivity of any WHERE-clause conditions that are being applied at the node. But rows output is somewhat tricky because it is not the number of rows processed or scanned by the plan node. Ideally the top-level rows estimate will approximate the number of rows actually updated, deleted, or returned by the query.

The planner cost and rows output will be used to estimate the query completion time. The cost estimates are expressed in arbitrary units, but thing to pay attention to is ratios of actual time taken by query and estimated planner cost is somewhat consistent.

1.1 Feedback Mechanism

As explained in the previous section, the query plan is divided into number of nodes (for large query) and each node has cost/"rows output". We will extrapolate planner cost and rows output (and some heuristic, which will be based on testing of large TPCH queries) of all the nodes in plan to come with rough query completion estimates, when query start execution.

As query progresses we will go on refining estimates based on actual time taken by each node (sometimes also called as snippet). The current execution node estimates will be then taken based on above feedback and planner cost/"rows output". Please note we are considering that query is going to take maximum (around 90%) time in execution phase and very less time in parsing, analyze, rewrite, planning, optimization phase.

1.2 Plan Tree Walker

The structure of a query plan is a tree of plan nodes. We will walk the entire plan tree to come up with rough query estimates at start and then go on refining the estimates based on above mathematical model. The planner has different types of nodes based on kind of operation node is going to perform. For example, there are different types of scan nodes for different table access methods: sequential scans, index scans, and bitmap index scans. If the query requires joining, aggregation, sorting, or other operations on the raw rows, then there will be additional nodes "atop" the scan nodes to perform these operations. We will walk the entire query plan tree to get rough total query completion estimate at start. During query execution, execution engine walks through all the plan nodes sequentially. At each plan node we will use our feedback model to refine the particular node's estimate and also total query completion estimates.

2) Algorithm

1. **Input**: SQL Query
2. **Calculate query estimate**

2.1 Calculate cost based on plan tree.

```
// calculating the total cost of the plan tree by recursively traversing the plan tree
// p = plan tree structure of query
// x = plans rows of node
// y = deciding factor, where to take plan rows input or not
// result is updated in global "final_cost" variable
 if (y==1)
            p->conti = p->plan_rows;
             else
            p->conti = 2;
 final_cost = final_cost + (p->total_cost * p->conti);
 estimate_cost(p->lefttree,x,1);
estimate_cost(p->righttree,x,1);
```

2.2 **Calculate percentage with respect to cost of tree**.

```
// Calculate the percentage or contribution with respect to total cost of tree.
// percentage_so_far: stores the accumulated percentage
// final_cost: total cost of plan tree. Calculated prior at the end of planning phase.
// value: cost of the current executing node.
// Used "90" based on heuristic – considering the fact that execution phase going to eat most of // the time
percentage_so_far = percentage_so_far + (value * 90) /  final_cost;
return percentage_so_far;
```

2.3 **Estimate updation based on feedback**

Below calculation will be done by execution engine during each plan node execution.

```
// Take feedback into account.
// final_cost is global variable and its updation will reflect in all the alogorithms
final_cost =   final_cost * actual_time_so_far / total_cost_so_far
current_cost = current_node_cost * actual_time_so_far / total_cost_so_far
total_cost_so_far = total_cost_so_far + current_cost
// call percentage completion function
percentage_completion (total_cost_so_far);
```

3. **Output**: GUI indicating progress of query execution.

IV. Experimental Evaluation

This section presents experimental results which in turn show the effectiveness of the proposed techniques. We first describe the experimental setup and then evaluate the performance of progress indicator.

Experimental Setup

We implemented progress indicator in postgreSQL database server 9.0.4. The experiments were run on the machine with Core 2 Duo(64 bit processor) with memory requirement as minimum 2GB RAM 150 GB of Hard-Disk. The operating system used in this experimental setup is open Solaris 10 operating system. Also Dtrace tracing tool is used for dynamically tracing database server. Also J2EE and Java language is used for GUI of system and C language is used for database coding.

A. Performance of Progress Indicator

1) GUI Design PostgreSQL Query Progress Indicator

Figure 3. GUI Design For Query Progress Indicator

2) XY chart

Figure 4. XY chart

3) Bar chart

Figure 5. Bar chart

4) Aborting query

Figure 6. Aborting Query

5) Query history

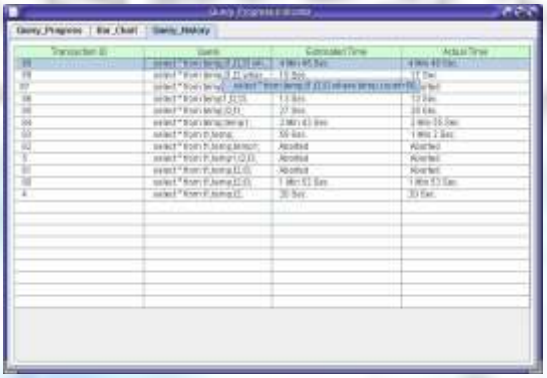

Figure 7. Query History

V. Conclusion

The SQL progress indicators for long-running queries are nowadays becoming a desirable user-interface tool to monitor progress of executing query in RDBMSs. But all the previously proposed techniques for supporting the construction of progress indicators for SQL queries are having very limited functionality and accuracy. In this paper, we have implemented a technique based on query optimizer's cost which can be used for the development of query progress indicator.

VI. Future Enhancement

As we know that today's world is completely dependent on the internet and online tools. We can enhance our idea and can make our tool as web portal, so that anyone can use it at any time. We can also send the progress status of the query through email or the sms to the DBA. So that he can know the progress of the query without running the GUI and sitting in front of the machine. So like this possibilities are endless.

References

[1]. Avi Silberschatz, Henry F. Korth, S. Sudarshan. (2005). Database System Concept. (5th Edition). [online] Available: http://www.cse.iitb.ac.in/~sudarsha/db-book/slide-dir/ch1.pdf

[2]. Basit Raza, Abdul Mateen, M M Awais and Muhammad Sher; "Survey on Autonomic Workload Management: Algorithms,Techniques and Models"; Journal of computing, volume 3,Issue 7,July 2011, ISSN 2151-9617.

[3]. Kristi Morton, Abram Friesen, Magdalena Balazinska, Dan Grossman; "Estimating the Progress of Map Reduce Pipelines";ICDE Conference 2010.

[4]. Elnaz Zafarani, Mohammad_Reza Feizi_Derakhshi, Hasan Asil, Amir Asil; "Presenting a New Method for Optimizing Join Queries Processing in Heterogeneous Distributed Databases"; 2010 Third International Conference on Knowledge Discovery and Data Mining.
[5]. Mario Milicevic, Krunoslav Zubrinic, Ivona Zakarija; "Dynamic Approach to the Construction of Progress Indicator for a Long Running SQL Queries"; international journal of computers issue 4, volume 2, 2008.
[6]. Mario Milicevic, Krunosla V Zubrinic, Ivona Zakarija; "Adaptive Progress Indicator for Long Running SQL Queries"; Proceedings of the 8th WSEAS International Conference on Applied Computer Science(ACS'08).
[7]. Chaitanya Mishra, Nick Koudas; "A Lightweight Online Framework For Query Progress Indicators"; 2007 IEEE.
[8]. Gang Luo , Jeffrey F. Naughton , and Philip S. Yu;" Multi-query SQL Progress Indicators"; Y. Ioannidis et al. (Eds.): EDBT 2006, LNCS 3896, pp. 921 – 941, 2006, Springer-Verlag Berlin Heidelberg 2006.
[9]. Christian M. Garcia-Arellano, Sam S. Lightstone, Guy M. Lohman, Volker Markl, and Adam J. Storm; "Autonomic Features of the IBM DB2 Universal Database for Linux, Unix, and Windows"; IEEE Transactions on systems, MAN, And Cybernetics Part C:Applications And Reviews, Vol.36,No.3, May 2006.
[10]. Gang Luo, Jeffrey F, Naughton, Curt J. Ellmann, Michael W. Watzke; "Increasing the Accuracy and Coverage of SQL Progress Indicators"; Proceedings of the 21st International Conference on Data Engineering (ICDE 2005).
[11]. S. Chaudhuri, R. Kaushik, and R. Ramamurthy, "When can we trust progress estimators for SQL queries?" in SIGMOD, 2005.
[12]. Suraji Chaudhuri, Vivek Narasayya, Ravishankar Ramamurthy; "Estimating Progress of Execution for SQL Queries"; SIGMOD 2004, June 13–18, 2004, Paris, France, 2004 ACM.
[13]. Gang Luo , Jeffrey F. Naughton , Curt J. Ellmann , Michael W. Watzke; "Toward a Progress Indicator for Database Queries"; ACM SIGMOD 2004, June 13–18, 2004, Paris, France,2004 ACM.
[14]. Chaitanya Mishra, Nick Koudas; "A Lightweight Online Framework For Query Progress Indicators"; 2002 ACM.
[15]. Jiexing Li, Rimma V. Nehme , Jeffrey Naughton; "GSLPI: a Cost-based Query Progress Indicator"; 2012 IEEE 28th International Conference on Data Engineering.