Automatic Indexing in PostgreSQL

Ashish Wankhade, Ameya Mohape, Ashish Jangra

Department of Computer Science and Engineering, IITB

{183050078, 183050055, 183050043}@iitb.ac.in

Abstract: Indexing helps in increasing performance by reducing the number of disk accesses required when queries are executed. PostgreSQL creates an index on key attributes by default. However, it does not create index on non-key attributes even if there are considerable number of queries with such attributes. This reduces the performance drastically. We have developed an application to automatically create index on non-key attributes by taking into account the frequency of scans performed by PostgreSQL during normal query execution. This is done by making changes to the source code of PostgreSQL to output relation scan data into a separate log file. The python application periodically extracts data from the log file to check if an index should be created and triggers index creation when a certain desired threshold is crossed.

1. Introduction

Query processing in PostgreSQL takes place it four steps:

a. Parse (*Bison*)
b. Analyse and Rewrite (function *pg_analyse_and_rewrite*)
c. Plan (function *pg_plan_queries*)
d. Execute (function *exec_simple_query*)

PostgreSQL converts a query into a parse tree. This is done through the use of *Bison* which is a parser generator. This tree is then passed to the analyse and rewrite step which simplifies and optimizes the query. The third step uses this information to create a plan for executing the query. The last step uses the now created plan to execute the query and generate the results.

2. Logging query plans and extracting relation attributes

We have edited the function *ExecutorEnd* in file execMain.c in *src/backend/executor/*. We have used code from *explain.c* file in *src/backend/commands/* and appropriate logic to get output of execution plan as JSON data. We parse the plan and for every sequential scan in the plan, we extract from the plan attributes like relation name, alias, total cost and filter attributes and put the values of these attributes in a log file.

3. Automatic Indexing

We run a separate python daemon(*daemon.py*) in the background which monitors the logfile. If sequential query data is encountered on some combination of attributes of a relation, cost of the sequential scan is added to accumulated cost for queries on same combination of filter attributes. If the accumulated cost crosses some predefined threshold, an index with same combination of attributes is created by sending a command to *psql* application with appropriate parameters. Both files *daemon.py* and *execMain.c* have absolute paths for log file. To run the code, edit log file path in *daemon.py* at line 33 and *execMain.c* at lines 705 and 835 appropriately.

4. Conclusion

We have successfully implemented automatic indexing by keeping track of frequency of sequential scans on the relations. Such automatic indexing for non-key attributes reduces the query cost considerably as expected from indexing. Automatic deletion of unused indices can be easily implemented in the future using timestamps. Indices not used for a long period could be deleted safely. Cost of older recorded queries could be made to decompose logarithmically to give more realistic results. Another important enhancement that can be made is to put the extracted data directly into the database instead of a log file.

5. Contributions

Ashish Wankhade (183050078) – coding and implementation
Ameya Mohape (183050055) – study PostgreSQL structure and logic
Ashish Jangra (183050043) – background study, testing, bug fixing