Autonomic Index Management

Sérgio Lifschitz, Marcos Antonio Vaz Salles
Departamento de Informática
Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio)
{sergio,mvsalles}@inf.puc-rio.br

Abstract

This work presents the core ideas for an approach that enables index tuning with no human intervention at all. SQL statements submitted to the database are monitored and indices are created or dropped according to the benefits or overheads regarding the actual workload. We have implemented this strategy using a software agent architecture embedded in PostgreSQL. We discuss here some of the architecture core ideas and present initial experimental results.

1 Introduction

The database tuning task consists of fine manipulations aiming at obtaining better performance of DBMS-based applications by means of an efficient use of the available computational resources. Tuning can as well be done in hardware configuration, physical design and query specifications. Due to the high complexity of DBMSs' current implementations, self-tuning or autonomic solutions are still very restricted.

We are mainly concerned in this paper with index autonomic management for database systems. We briefly discuss an approach that matches software agents systems and DBMSs in a feasible architecture. In our work we study strategies to completely automate the index self-tuning process. This means no human intervention at all. We integrate a software agent with the DBMS optimizer to choose good indices and create them when needed. We have implemented our architecture in PostgreSQL and obtained promising improvements to system performance. It is important to note that, unlike our proposal, all previous works in the literature require DBA intervention in order to complete the whole index tuning process.

The rest of the paper is organized as follows: in the next Section, we discuss our agent architecture for autonomic index management. Then, in Section 3, we present initial experimental results. Finally, we conclude in Section 4 listing our contributions and future work.

2 Index Management Architecture

Finding an adequate index design depends fundamentally on the workload submitted to the DBMS. Workloads mix query and update statements (insert/delete/update) with varied frequencies. Performance conflicts can be created when an index benefits queries and hurts updates. To resolve such a conflict one should choose the alternative index design that minimizes the total impact on computational resources and, therefore, maximizes throughput.

In our approach, we use a component embedded in the DBMS to manage indexes automatically. Our architecture uses a layered software agent, here called *Benefits Agent*, that monitors the system and creates or destroys indexes. Software engineering techniques needed to build such an agent are discussed in [1]. Figure 1 shows the self-tuning model for the *Benefits Agent*.

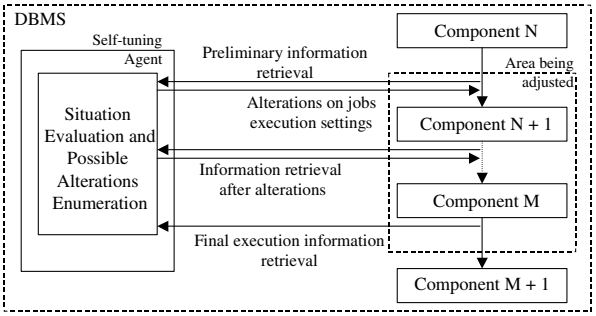

Figure 1. Index self-tuning model

The self-tuning agent interacts with DBMS components guided by a self-tuning process with the following stages: Information Retrieval, Situation Evaluation, Possible Alterations Enumeration and Alterations Accomplishment. During Information Retrieval, the *Benefits Agent* interacts with the DBMS to obtain every SQL statement that is optimized,

Proceedings of the Second International Conference on Autonomic Computing (ICAC'05)
0-7695-2276-9/05 $ 20.00 **IEEE**

along with cost and index usage information. For each statement, Situation Evaluation takes place, causing the agent to update its beliefs about the statement currently being processed by the system. The agent then applies heuristics to enumerate hypothetical index designs and to evaluate the fitness of candidate indices.

The benefits brought by candidate indexes are estimated using the system's optimizer. These calculations comprise the Possible Alterations Enumeration stage. More details on the heuristics we have used can be found at [3]. Finally, the Alterations Accomplishment stage is executed, with the consequent creation or destruction of actual indices, if necessary.

3 Implementation

We have implemented the *Benefits Agent* on PostgreSQL 7.4, beta 3, running on Red Hat Linux 8. Our prototype deals, so far, with index creation but not removal. To evaluate its effectiveness, we have used OSDL's DBT-2 toolkit [2] to submit PostgreSQL to an OLTP workload. This toolkit uses TPC-C transactions and, therefore, simulates a wholesale parts supplier operating out of a number of warehouses and their associated sales districts.

The DBT-2 toolkit comes with a set of suggested indices that improve the evaluation of the given workload. In order to evaluate the contribution brought by our agent, we have observed system throughput in three test configurations:

1. A database with no indices at all and our agent turned off, which we call here I0A0 (both indices and agent zero).

2. No indices at the database and agent turned on (I0A1 - indices zero, agent one).

3. Agent turned off and the database with the indices suggested by the DBT-2 toolkit (I1A0 - indices one, agent zero).

Two variables may affect the experiments' throughput. The first one, the number of warehouses in the application, may be viewed as a scale factor and, therefore, the greater this number is, also greater will be database tables' sizes and the user load on the system.

The second variable to be considered is how long the experiment takes. When the agent is active, it must first learn which are the best indices for the submitted commands and materialize them. Then the agent enters a steady state phase, in which it only verifies whether or not the existing indices are still adequate. The throughput in the steady state phase is considerably greater than during learning periods, consequently leading to a greater average throughput in longer experiments.

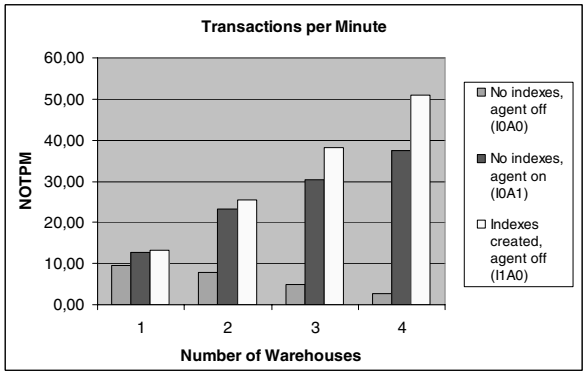

Figure 2. Throughput in a 90 minute test

Figure 2 shows the results obtained for a 90 minute time test. It is worth noting that the throughput for the I0A0 configuration decreases when the number of warehouses increases. This is due to full scan query processing, leading to worse performances when we deal with larger tables. As expected, an intermediate throughput appears for the I0A1 configuration.

The agent eventually reaches the end of the learning step and the workload remains active for a while with adequate indices. Therefore, the throughput gets closer to the I1A0 configuration throughput as the steady state phase duration increases. Additional experiments indicate that the longer are tests' running times, the more interesting it gets to have our agent present in the environment [3].

4 Conclusion

In this paper we have integrated a software agent with PostgreSQL and have conducted experiments that indicate that our agent may tune up the database's index design dynamically. As future work, we would like to extend the implementation to deal with index destruction and to investigate how to make our agent able to deal with workloads for decision support applications and involving *ad hoc* queries.

References

[1] R. Costa, S. Lifschitz, and M. Salles. Index self-tuning with agent-based databases. *CLEI Electronic Journal*, 6(1):22 pages, 2003. http://www.clei.cl/cleiej/paper.php?id=88.

[2] Open source development labs database test 2 (osdl-dbt-2). http://www.osdl.org/lab_activities/kernel_testing/osdl_database_test_suite/osdl_dbt-2/.

[3] M. Salles. Autonomic index creation in databases (in portuguese). Master's thesis, Dep. de Informática, Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio), 2004.