COLT: Continuous On-Line Database Tuning

Karl Schnaitter[†] Serge Abiteboul[‡] Tova Milo[±] Neoklis Polyzotis[†]

[†]Univ. of California Santa Cruz
{karlsch,alkis}@cs.ucsc.edu

[‡]INRIA and Univ. Paris 11
fname.lname@inria.fr

[±]University of Tel Aviv
milo@cs.tau.ac.il

ABSTRACT

Self-tuning is a cost-effective and elegant solution to the important problem of configuring a database to the characteristics of the query load. Existing techniques operate in an off-line fashion, by choosing a fixed configuration that is tailored to a subset of the query load. The generated configurations therefore ignore any temporal patterns that may exist in the actual load submitted to the system.

This demonstration introduces COLT (Continuous On-Line Tuning), a novel self-tuning framework that continuously monitors the incoming queries and adjusts the system configuration in order to maximize query performance. The key idea behind COLT is to gather performance statistics at different levels of detail and to carefully allocate profiling resources to the most promising candidate configurations. Moreover, COLT uses effective heuristics to regulate its own performance, lowering its overhead when the system is well-tuned, and being more aggressive when the workload shifts and it becomes necessary to re-tune the system. We present a specialization of COLT to the important problem of selecting an effective set of relational indices for the current query load. Our demonstration will use an implementation of our proposed framework in the PostgreSQL database system, showing the internal operation of COLT and the adaptive selection of indices as we vary the query load of the server.

1. INTRODUCTION

Consider a data server (e.g., a database or an XML repository) confronted with a heavy workload of queries. Typically, the physical database schema is enriched with specific access structures, such as indices or materialized views, that can improve the performance of query evaluation. The selection of such structures is performed by a system administrator, based on her knowledge of the query load and perhaps with the help of tuning tools [4, 9], and it is in itself an optimization problem: given a limited storage space for physical structures, select the ones that are most effective for the expected query workload. When the workload is not stable, however, this off-line approach has serious shortcomings as the selection may soon become inappropriate; furthermore, the Internet has allowed many data sets to be made available to large audiences, and there is often no expert administrator who may configure the system.

These observations have motivated our work on an on-line tuning framework, termed COLT (Continuous On-Line Tuning), that supports the automatic selection of physical access structures. COLT addresses the following problem: given a stream of queries and a storage budget, it dynamically selects access structures that fit in the budget and improve the performance of query processing for the current characteristics of the query load. In order to provide the most effective tuning, COLT continuously monitors the query load in order to judge the benefit of existing and hypothetical access structures, and periodically adjusts the set of materialized structures to provide the best gain in performance. This is the key difference between our work and the majority of recent studies on self-tuning [3, 5, 1, 2], which determine the configuration of the database off-line and are thus susceptible to the shortcomings outlined earlier. There has been recent work on the problem of on-line index selection [8] that is similar in spirit to the proposed COLT framework. The novelty of COLT lies in its ability to regulate the overhead of on-line tuning depending on the performance of the system, and to allocate its resources judiciously based on a principled methodology.

We illustrate the application of COLT on the practical problem of self-tuning a relational database system [7, 6, 3]. To focus the presentation further, the access structures we consider are indices. We want to stress, however, that the technique goes beyond relational queries and indices; indeed, the present work has been originally motivated by the on-line tuning for an XML system taking region and keyword indices into account.

The demonstration will highlight the benefits of the approach, using our implementation of COLT inside the PostgreSQL database system. We will show how the system self-tunes to a given workload and to shifts in workloads. We will interact with the server by executing queries both individually and in batches. Our interaction will simulate various operating conditions, including (i) a query load with stable characteristics, (ii) a transition in the query load, and (iii) a query load with "noise" in the form of random queries. The demonstration will visually display COLT's adaptations to the environment, such as the evolution of the index configuration and the resources devoted to self-tuning.

2. COLT OVERVIEW

Figure 1 depicts the high-level architecture of COLT. Its main components are the Extended Query Optimizer that optimizes the incoming flow of queries as well as profiles indices, and the Self-Tuning Module that selects the materialized indices. In this section, we describe how these new modules operate and define how they interact with the rest of the system.

SIGMOD 2006, June 27–29, 2006, Chicago, Illinois, USA.

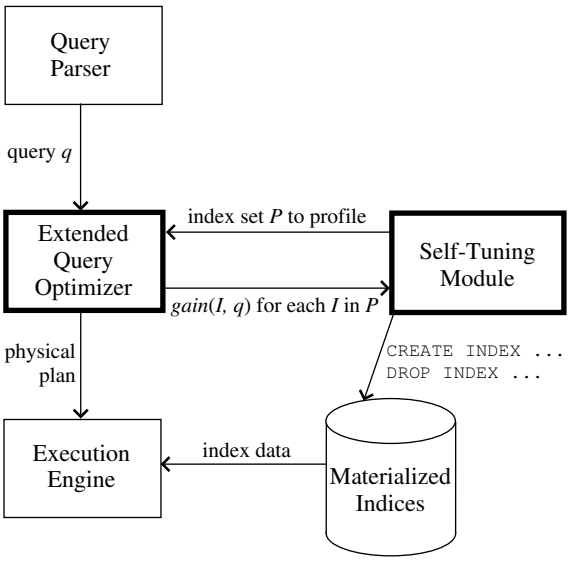

Figure 1: COLT Architecture.

2.1 The Extended Query Optimizer

We have replaced the query optimizer in the traditional query pipeline with an *Extended Query Optimizer* (EQO). The primary responsibility of the EQO is the same as the standard optimizer: to choose the optimal physical execution plan for a query. The EQO extends the functionality of the optimizer with the ability to *profile* hypothetical and materialized indices. More concretely, given a query q and an index I to be profiled, the EQO will return the reduction in time to execute q that would result from using I. We term this reduction the *gain* of I *with respect to* q and denote it as $gain(I, q)$. We will often abstractly refer to the *gain* of an index as the performance improvement that the index can provide for the current query load. The gain of an index, therefore, is always non-negative, and a zero gain ($gain(I, q) = 0$) implies that I cannot improve the running time of q. In addition, the gain of an index depends on the current set of materialized indices and hence $gain(I, q)$ can change depending on when it is evaluated.

The EQO computes index gains in a batch fashion as part of the optimization of the current query q. More precisely, the EQO first chooses the optimal execution plan for q using the current set of materialized indices $\mathcal{M}$. Let c_M be the cost of this initial plan. Subsequently, the EQO receives an optional set P of indices to profile. For each index $I \in P$, the EQO recomputes the optimal execution plan for q as follows: (a) if I is materialized, the EQO generates the optimal plan that does not use I, or (b) if I is a *hypothetical* index, the optimal plan is chosen assuming that I is available. We refer to these plans as *what-if plans* [4]. Let c_I be the cost of a what-if plan. With these measurements, the EQO calculates $gain(I, q) = |c_M - c_I|$.

This batch-style profiling is important for reducing the cost of what-if planning, as the initial planning only needs to be performed once for the whole set of profiled indices. To reduce the overhead further, our implementation of the what-if planner reuses intermediate solutions from the initial planning stage. Of course, not all intermediate solutions can be reused, so the optimizer must be careful to only reuse subsolutions that do not depend on the profiled index.

2.2 The Self-Tuning Module

The purpose of the *Self-Tuning Module* (STM) is to tune the system by adjusting the set of materialized indices. In order to evaluate the potential benefit of candidate indices, the STM sends profiling requests to the EQO. The STM also interacts with the physical data storage by sending commands to create or delete indices. Based on this interface, the STM has two difficult tasks: (1) choosing candidate indices to profile, and (2) maintaining a set of materialized indices that fit within the storage budget. In order to coordinate these tasks, the STM partitions the incoming queries into regular intervals, called *epochs*. In our implementation, an epoch consists of 10 queries. During an epoch, the STM gathers statistics for the query load and profiles the gains of candidate indices. At the end of each epoch, the candidates are reexamined and changes may be made to the materialized indices.

In order to be effective, the STM has to address several issues that pertain to the selection of candidate indices, their efficient profiling, and the choice of which structures are materialized. We now outline the challenges involved and discuss the highlights of our proposed approach.

Index Organization. As queries are submitted to the system, the STM analyzes the workload and decides which candidate indices are relevant. The (potentially many) candidate indices are partitioned into three sets as follows. The *materialized set* $\mathcal{M}$ contains the materialized indices that are managed by the STM. The *hot set* $\mathcal{H}$ contains candidate indices that are relevant to the recent query load and have shown strong evidence that they will significantly improve performance. The *cold set* $\mathcal{C}$ contains candidate indices that are relevant to the recent query load, but have only shown mild evidence that they will improve performance.

Profiling Strategy. The distribution of indices into $\mathcal{M}$, $\mathcal{H}$, and $\mathcal{C}$ is primarily guided by the estimated gains of candidate indices. Since what-if plans are relatively expensive to compute, the STM is allocated a profiling budget, which limits the number of what-if plans that may be evaluated in each epoch. This budget can change over time based on the stability in the query load. Profiling is also constrained by the interface of the EQO, because what-if plans may only be evaluated with respect to the current query that the system is processing.

Given these restrictions, the STM has to select carefully which indices to profile after each query is optimized. To address this issue, we adopt the following profiling methodology. For cold indices, the STM evaluates their performance only with a crude metric based on the selectivity of predicates in the query. For hot and materialized indices, the STM uses what-if calls to the EQO in a randomized fashion, with greater probability given to the indices for which the gain is the most uncertain. More concretely, the STM may be able to confidently guess the gain of an index with respect to the current query if it has observed consistent gains with "similar" queries in the past. In our implementation of COLT, two queries are similar if they involve the same relations, and their selection predicates only differ slightly in selectivity. The STM is more likely to allocate what-if calls if there have not been similar queries in the past with consistent gains, because a confident guess cannot be made in this case.

Index Selection. As queries are submitted, the STM identifies candidate indices that are relevant to selective predicates, and new candidates are placed in the cold set. At the end of each epoch, the STM uses the estimated index gains to partition candidate indices into $\mathcal{M}$, $\mathcal{H}$, and $\mathcal{C}$. This redistribution is performed as follows. A cold index that has shown significant potential for the current query

load (based on the crude metric mentioned above) may be *promoted* to become a hot index. If profiling shows that a hot index has high gain, the STM may select it for materialization by promoting the index to $\mathcal{M}$ and creating the index in physical storage. On the other hand, the index may lose its relevance after being promoted. In these cases, the STM may choose to *demote* indices from $\mathcal{M}$ to $\mathcal{H}$, or from $\mathcal{H}$ to $\mathcal{C}$. For each demotion from $\mathcal{M}$, the corresponding index is deleted from storage. Since a change in the materialized set can be costly, the STM chooses to promote an index to $\mathcal{M}$ based on its projected future gain as well as the cost of materialization.

3. DEMONSTRATION

The purpose of our demonstration is to illustrate how the techniques of COLT are used to tune the system. The demonstration is built on our implementation of COLT that we have integrated into the PostgreSQL database system. In order to make the demonstration clear, we use a simple, synthetic data set. We have two query loads, named W_1 and W_2, that contain queries with predictable characteristics. Many of the queries are chosen from W_1 or W_2, but we also introduce "noise" queries that have characteristics that differ significantly from W_1 and W_2.

We consider two main scenarios in our demonstration:

1. The first scenario demonstrates the behavior of the system with a changing workload. The system is first given queries from W_1. We monitor the operation of the STM (see details below) and measure the improvement and convergence of performance. We then switch to the workload W_2 and monitor the automatic adjustment of the system to the new workload. When we switch to W_2, we will see a drop in performance, which will climb back up again as the system self-organizes.

2. The second scenario demonstrates the effect of noise in the workload. We give the system a steady workload, and introduce brief periods of "noise," where we have queries that are not characteristic of the workload. We observe the loss in performance during the noisy periods, and the ability of the system to return to the optimal configuration.

The visual aspects of the demonstration are divided into a *control panel* and a *surveillance panel*. The contents of these components are detailed below.

The Control Panel. In our demonstration, we have the ability to set various parameters that control the flow of queries and the settings of COLT. We set these parameters in the control panel. We select the executed queries by a *workload distribution* control and a *noise rate* control. The workload distribution controls the portion of workload queries (i.e. queries that are not noise) that are in W_1 and W_2. For example, if the workload distribution is 60/40, then 60% of workload queries come from W_1 and 40% of workload queries come from W_2. The noise rate is a number between 0 and 1 that indicates the probability of choosing a noise query. Once these parameters are chosen, we may execute random queries selected from the desired distribution. The actual queries that are submitted will be displayed in order to view the true contents of the workload. We execute individual queries to show the detailed behavior of COLT, or at times we may execute multiple queries to "fast-forward" the demonstration.

The Surveillance Panel. The central visual component of our demonstration is the surveillance panel. The internal operation of the STM is shown in this component. Most importantly, the status of candidate indices is displayed. The surveillance panel shows the division of candidate indices into the cold, hot, and materialized sets. We also display the gain that the STM has evaluated for each index. This will help visualize how the transitions of candidate indices are made between $\mathcal{C}$, $\mathcal{H}$, and $\mathcal{M}$.

In the interactive demonstration, we will use a simplistic setting to clearly illustrate the underlying techniques, the functionalities of the system, and its benefits. To conclude the demonstration, we will briefly present experimental results that demonstrate benefits provided by COLT in more realistic and complex settings. In particular, we will show that COLT provides a significant improvement in average query time when compared to a system that has been configured with an optimal fixed set of indices using off-line techniques.

Acknowledgments

This work was supported in part by EC project Edos, by the Israel Science Foundation, and by a research grant from Microsoft Corporation.

4. REFERENCES

[1] S. Agrawal, S. Chaudhuri, and V.Narasayya. Automated Selection of Materialized Views and Indexes for SQL Databases. In *Proceedings of the 26th Intl. Conf. on Very Large Data Bases*, pages 496–505, 2000.

[2] Elena Baralis, Stefano Paraboschi, and Ernest Teniente. Materialized views selection in a multidimensional database. In *Proceedings of the 23rd Intl. Conf. on Very Large Data Bases*, pages 156–165, 1997.

[3] Surajit Chaudhuri and Vivek R. Narasayya. An efficient cost-driven index selection tool for microsoft sql server. In *vldb97*, pages 146–155, 1997.

[4] Surajit Chaudhuri and Vivek R. Narasayya. Autoadmin 'what-if' index analysis utility. In *Proceedings of the 1998 ACM SIGMOD Intl. Conf. on Management of Data*, pages 367–378, 1998.

[5] Himanshu Gupta, Venky Harinarayan, Anand Rajaraman, and Jeffrey D. Ullman. Index selection for olap. In *Proceedings of the 13th Intl. Conf. on Data Engineering*, pages 208–219, 1997.

[6] Michael Hammer and Arvola Chan. Index selection in a self-adaptive data base management system. In *Proceedings of the 1976 ACM SIGMOD Intl. Conf. on Management of Data*, pages 1–8, 1976.

[7] N.Bruno and S. Chaudhuri. Automatic Physical Database Tuning: A Relaxation-based Approach. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, 2005.

[8] Kai-Uwe Sattler, Eike Schallehn, and Ingolf Geist. Autonomous Query-driven Index Tuning. In *Proceedings of the International Database Engineering and Applications Symposium*, 2004.

[9] Daniel C. Zilio, Jun Rao, Sam Lightstone, Guy M. Lohman, Adam Storm, Christian Garcia-Arellano, and Scott Fadden. DB2 Design Advisor: Integrated Automatic Physical Database Design. In *Proceedings of the 30th Intl. Conf. on Very Large Data Bases*, pages 1087–1097, 2004.