

On-line Index Selection for Physical Database Tuning

Karl Schnaitter, UC Santa Cruz

Thesis Proposal, May 12, 2008

Advisor: Neoklis Polyzotis

Abstract

This document presents the main ideas behind my proposed thesis on the topic of physical database tuning. In particular, my thesis will examine methods for on-line index selection. There has been an increasing interest in this problem just in the past year, since on-line index selection has the potential to be very beneficial for database administration. The recent work has provided some very interesting ideas, but the problem has proven to be very difficult and the proposed methods have weaknesses that make them unsuitable for practical use. The primary goal of my thesis is to develop a method for on-line index selection that is robust enough to be deployed in real systems. The approach will incorporate the lessons learned from previous attempts at on-line index selection, while bringing in relevant ideas from the field of on-line computation. It will also require a principled experimental study to verify the performance of the approach. Finally, my thesis will explore an alternative user interface for index selection that works in a semi-automatic fashion. The result will be an end-to-end solution for on-line index selection that is viable in practice. In this proposal, I describe my progress towards these goals and my plans for the coming year.

1 Introduction

From a logical viewpoint, a relational database is defined simply as a collection of tables, each containing records with some specific attributes. On the other hand, the physical data in those tables may be stored in a huge number of different configurations: The data may be arranged in sorted order on disk, tables may be partitioned or replicated on different storage devices, and access structures such as indexes and materialized views may be constructed to speed up query evaluation. The choice of physical design can have a major impact on system performance, and with so many choices it can be nearly impossible to choose the best configuration. Accordingly, physical database design (a.k.a. physical database *tuning*) has been an active research area since the early days of relational database systems [LL71, HC76].

Previous works have looked at many facets of physical tuning, but the main focus of my thesis will be on index selection. More specifically, I will generally consider *B-tree indexes*. A B-tree index allows fast access to the records of a table whose attributes satisfy some equality or range conditions, and also enables sorted scans of the underlying table. The focus on indexes will make the ideas that I explore in my thesis more concrete, and reduce the obstacles to conducting meaningful experiments (not all database systems support other structures). At the same time, it is important to keep other physical design features in mind, and I plan to consider such extensions as part of my dissertation.

During the past few decades, the majority of methods for index selection have taken an *off-line* approach. The general procedure of off-line index selection starts with a description of the query load that will be submitted to the system in the future. It is then a matter of maximizing the

overall performance of the representative workload by manually choosing indexes [LTN07] or using an automated tool [CN97, VZZ⁺00] to make the selection. A myriad of off-line methods have been proposed [FST88, BPS90, CFM95], and some modern techniques (such as the relaxation approach of Bruno and Chaudhuri [BC05]) have reached a considerable level of sophistication. However, there are some drawbacks that are inherent in off-line index selection, described as follows.

- *Workload selection:* Every algorithm for off-line index selection assumes that a representative workload is provided as input, but there are many ways to construct this workload. For example, it can be a window of the most recent queries, a random subset of user queries, or a hand-written collection of queries that are considered important by the administrator. In any case, choosing a good workload is a nontrivial task, and the quality of the recommended index configuration depends heavily on this step.
- *Dynamic environments:* The representative workload can only model the current database access patterns. If the access patterns change over time, it is necessary to repeat the index selection from scratch with a new workload. Moreover, it is not always obvious when this effort becomes appropriate.
- *Index creation cost:* The cost of creating indexes can be substantial. Off-line tuning algorithms, however, do not account for the cost of materializing the indexes they recommend, since the physical design work is assumed to be done separately from the normal operation of the system. This issue is particularly problematic in dynamic environments, where frequent changes to the index configuration may severely disrupt query performance.

These issues with off-line index selection have naturally led the database community to study *on-line* index selection. The main idea is to add an on-line tuning module to the database architecture that monitors the query load to identify dominant access patterns, and automatically adjusts the index configuration to maximize query performance. The move to on-line tuning, however, leads to challenges that are not found in the off-line setting. Specifically, an on-line tuner faces the same challenging task of selecting an effective index design, but it must do so with limited overhead in order to avoid disrupting the concurrent execution of queries. Moreover, an on-line tuner does not have the convenience of a representative workload, and instead must identify promising configurations based on the current trends in the queries submitted by clients. Finally, and more importantly, the on-line tuner must determine whether a recent change in the workload provides enough evidence of a shift that justifies the cost of materializing new index structures. These observations hint at the challenges behind the on-line problem, and indicate that a solution is not likely to come from straightforward extensions of existing off-line techniques.

Some past studies have developed rudimentary on-line tools for index selection in relational databases [HC76, FON92], but the idea has received little attention until recently. In the past year, on-line tuning has come into the spotlight and more refined solutions have been proposed [BC07, SLSS07, SAMP07]. Although these techniques provide interesting insights into the problem of selecting indexes on-line, they are not robust enough to be deployed in a real system. This opinion was corroborated by Surajit Chaudhuri, a leader in the field and co-author of one of the aforementioned works, during a talk that he gave at the 2007 VLDB Conference. The fundamental goal of my thesis, then, is to expose the specific weaknesses of these techniques, and develop a technique that overcomes those weaknesses in order to be useful in practice.

The following section discusses relevant background material on the subject of on-line computation and provides a brief overview of recently proposed algorithms for on-line index selection. The main components of my thesis are described in Section 3, and Section 4 contains a summary of my proposal.

2 Background

2.1 On-line Computation

Outside the realm of databases, there is a rich literature in the field of on-line algorithms. An on-line algorithm solves a problem that is exposed in pieces, often referred to as *requests*, and must make decisions as each new request is encountered. (In the case of on-line index selection, we consider each query to be a request, and the decisions of the on-line algorithm determine which indexes to create or delete.) Often the best decisions depend on requests that are not yet observed, but the algorithm must try to make good decisions nonetheless. The algorithms that have been studied are quite varied, but there are some general paradigms that we can identify.

Many on-line algorithms make predictions about future requests, and make decisions that have the most benefit with respect to those predictions. We say these algorithms belong to the *predictive* paradigm. A simple example of this concept can be illustrated by cache replacement policies. It is well known that an optimal policy is to replace the cache element whose next request is latest. Based on the notion of temporal locality, it is natural to predict that the least recently used element will be the last to be accessed next. This prediction immediately leads to the commonly used LRU policy for cache replacement. There are other on-line algorithms that involve more explicit predictions, and we will see an example shortly.

We also study a second paradigm of on-line algorithms that we refer to as *retrospective*. These algorithms analyze the history of requests in hindsight to determine what the best decisions would have been if all the requests were known ahead of time. The algorithm then uses this information to guide the next decision. In some cases, retrospective algorithms also take into account the actual decisions they have made so far. A frequently used example of the retrospective strategy is an algorithm for the ski rental problem: If we assume it costs \$10 to rent skis and \$100 to buy them, how many times should a skier rent before buying? A good algorithm will rent skis ten times before buying, because it is not until the eleventh ski trip that the purchase is more attractive in hindsight. A simple analysis of this algorithm shows that the skier pays no more than twice as much as they would have with full knowledge of the future. We state this property by saying the algorithm has a *competitive ratio* of 2.

These paradigms are not exhaustive, as there are many on-line algorithms that do not fall into either category. They are, however, an excellent fit for the on-line tuning algorithms that have been studied, as explained below.

2.2 Recent Proposals for On-line Tuning

This section finishes with a description of two on-line tuning algorithms that have been proposed recently. For each algorithm, I explain how it relates to the overall field of on-line computation and critique the main strengths and weaknesses of the technique.

2.2.1 QUIET: A Predictive Algorithm

One interesting approach to on-line database tuning is a system we refer to as QUIET [SLSS07]. (The name “QUIET” actually comes from an older version of the system [SSG04], but we use the name for convenience.) The algorithm falls into the predictive class of on-line algorithms. Throughout the operation of the system, QUIET maintains a pool of candidate indexes that are either materialized already or appear promising to be created in the future. The algorithm periodically predicts the benefit of each candidate index, and uses the prediction to select a set of indexes with the most benefit overall, subject to a limit on the total index storage. These indexes replace the current index configuration if their predicted benefit is larger by a factor of at least T , where T is a user-defined threshold.

One of the interesting aspects of QUIET is the function for future benefit. For each candidate index, the algorithm tracks the most recent k queries that would be affected by the index. The benefit of the index for each query is scaled based on the age of the measurement, and the scaled benefits are added together. The final prediction $benefit(I)$ is written as

$$benefit(I) = \sum_{j=1}^k \frac{profit(I, ts_j)}{ts_E - ts_j}$$

where ts_1, ts_2, \dots, ts_k are the timestamps of queries in the history of I , the current timestamp is ts_E , and $profit(I, ts_j)$ is the reduction in cost that I provides for the query at timestamp ts_j .

Another interesting contribution of this work is the study of a technique the authors call “on-the-fly” index generation. The general idea is to efficiently create the indexes selected by the on-line tuner by integrating index creation with query execution. This is an important component of on-line tuning because the overhead of index creation can be enormous. Some of the ideas are novel, and the authors provide a substantial performance study.

In addition to these strong points, there are some drawbacks of the QUIET system. One of the main issues is that several parameters need to be chosen off-line. Two of these parameters were described above: the number of queries k in the history of an index, and the threshold T that the predicted benefit of an index configuration needs to reach in order to get materialized. The value of T seems to be the most difficult to determine. The threshold must account for the overhead of index creation, since the predicted benefit of an index configuration does not include this cost. It is not clear how the index creation overhead relates to the metric used for future benefit, and the authors do not offer guidance to help make this determination.

It is often instructive to see simple examples where an on-line algorithm behaves poorly. Consider a scenario with two candidate indexes I_1 and I_2 , let Q_1 be a query that has a profit of 10 from I_1 , and let Q_2 be another query with a profit of 20 from I_2 . Suppose the query load is very periodic, repeating the pattern Q_1, Q_1, Q_2 . Both indexes consistently yield a profit of 20 units for every three queries, but the predicted benefit of I_2 will be much larger, because the k queries in its history carry more weight. Basically, the algorithm gives an unfair advantage to indexes with large, infrequent profits, and penalizes indexes that yield small, frequent profits. This favoritism seems arbitrary, and makes the algorithm less appealing. Moreover, the example could be extended to show the potential for major performance issues, even when the queries follow a simple repeating pattern.

2.2.2 The Retrospective Approach of Bruno and Chaudhuri

Another on-line algorithm was published last year by members of the AutoAdmin Project at Microsoft Research [BC07]. We refer to the algorithm as BC after the authors Bruno and Chaudhuri. In contrast to the QUIET system, BC does not use any predictions of the future, but instead takes a retrospective approach. The first step in the development of the algorithm looks at the scenario where there is only one candidate index that may be created or dropped. In this simple scenario, there is a very efficient algorithm that can determine points in the past where a configuration change would have been beneficial in hindsight. The simplified algorithm makes these changes as soon as these points are identified, and the authors show that this leads to a competitive ratio of 3. In other words, the total cost of query execution and index creation is within a constant factor of the optimal cost that would be possible with complete knowledge of the future.

The BC algorithm is an extension of the main ideas behind the single-index algorithm to the case of multiple indexes. The intuition is that the single-index algorithm can be run in parallel for each candidate index that is being considered. In reality, the algorithm is not that straightforward, due to two complicating factors: index interactions and storage limits. An index interaction occurs whenever the utility of an index depends on the existence of one or more other indexes. Interactions are addressed by a collection of special rules that scale the recorded benefits of indexes up or down whenever indexes they depend on are created or deleted. Storage limits are especially tricky for the BC algorithm because the logic of the single-index algorithm does not provide a metric that can be used to compare two indexes that are both attractive candidates for materialization. The authors invent a benefit metric in order to make comparisons, but the metric is inevitably detached from the logic of the competitive algorithm for one index.

There are very valuable ideas in the work of Bruno and Chaudhuri. I believe the most valuable idea is the algorithm for the single-index scenario. Their algorithm shows that it is very easy to achieve a competitive ratio in this case, and opens the door to new ways of thinking about on-line index selection. Another important idea in their work involves the evaluation of index benefits. They choose to use a fast and approximate method to determine the benefit of an index, whereas QUIET uses extra invocations of the query optimizer. Although BC does not use the complete functionality of the optimizer, it is still strongly integrated with the optimizer, using techniques originally developed for the physical design alerter [BC06]. As a side note, the idea of fast benefit evaluation is a hot research topic [PDA07], and we should expect more alternatives in the near future.

The main shortcomings of the BC algorithm are in the components previously mentioned for handling index interactions and storage limits. These aspects of BC make the algorithm unattractive for the simple reason that they are difficult to understand, and thus difficult to build upon and improve. There is also a technical weakness in the benefit metric that is used to compare two appealing indexes. Due to the complexity of the benefit metric, I will not describe all of its details here. The key aspect of the computation comes into play when a materialized index candidate is consistently beneficial for a long period of time. When the benefit metric of a materialized index gets too high, it does not keep increasing; it will decrease the benefit of all unmaterialized indexes instead. Under some situations, this can be reasonable, but it can also lead to poor decisions in other cases. We'll look at an example with three candidate indexes I_1, I_2, I_3 and three queries Q_1, Q_2, Q_3 such that I_i is beneficial for Q_i and irrelevant to the other queries. We also assume that there is enough storage for two of the three indexes. The example goes through three phases as follows.

- **Phase 1:** The query workload starts by alternating between Q_1 and Q_2 for a long time, causing I_1 and I_2 to be materialized.
- **Phase 2:** The workload starts repeatedly cycling through Q_1, Q_2, Q_3 . Rather than increasing the benefit metric of I_1 and I_2 without bound, the benefit of I_3 will be reduced each time these indexes are useful. Since there is not enough storage space for I_3 and the materialized indexes are still useful, this is reasonable.
- **Phase 3:** We stop seeing Q_1 and the workload alternates between Q_2 and Q_3 . Now each execution of Q_2 will still reduce the benefit metric of I_3 . If this reduction is enough to outweigh the contributions of Q_3 to I_3 , then the system will never materialize I_3 , despite the fact the I_1 is no longer used and should obviously be replaced.

This example shows that, similar to QUIET, there are very simple scenarios with obvious solutions where BC makes very bad decisions. It is surprising that the most up-to-date algorithms for on-line tuning have such poor performance in extremely simple cases. These examples are the strongest evidence that on-line index selection is an unsolved problem.

3 Dissertation Content

The aim of my dissertation is to develop on-line index selection into a practical tool that is useful for database practitioners. It will be necessary to leverage existing work and also develop substantial new ideas to meet this goal. For the core of the work, I will explore the predictive and retrospective approaches to tuning, beyond the progress made by prior studies. Another major step will be to develop a meaningful benchmark that can evaluate the performance of these systems. Finally, my dissertation will consider the user interface of the tuner, to make the system more friendly to administrators who would like to maintain some control over the index configuration. This will allow for an end-to-end system that is useful in practice. In the remainder of this section, I discuss some details of these ideas, my progress so far, and current plans going forward.

3.1 A Predictive Algorithm

In my earliest work as a graduate student, I helped to develop a system for on-line index selection named COLT, which stands for Continuous On-Line Tuning. We presented a demonstration of COLT [SAMP06] at the 2006 SIGMOD Conference, and published a paper [SAMP07] describing the details of the system at the 2nd International Workshop on Self-Managing Databases, which was co-located with the 2007 ICDE Conference. One of my contributions to this work was an implementation of COLT within the PostgreSQL Database System, which was used in our experimental study. I will give a brief overview of the system in this section; for the full details, I have attached our technical report with this proposal.

COLT is designed using the predictive paradigm described in Section 2. In our system, predictions are made relatively frequently. The incoming workload is divided into non-overlapping *epochs*, which consist of a small number of consecutive queries (10 queries in our experiments), and the future benefit of index candidates is computed after each epoch. The reason for these epochs is that predictions are not likely to change from one query to the next, so it is wise to space out the computational effort required to make predictions.

The predictions made by COLT are conceptually similar to the predictions used by QUIET¹, in the sense that we predict the future benefit of individual candidate indexes. However, our metric for predicted benefit is computed using a significantly different method. The basic idea behind our forecasting model is to apply a smoothing procedure to the past benefit measurements, then conservatively predict how the recent trends will continue. The most important difference in our prediction model is that our metric is meant to be directly comparable to the cost of materializing the index. This is important because the predicted benefit and materialization cost may be subtracted to predict the net effect of creating the index. This removes the need for a parameter similar to the threshold T used by QUIET.

The main contribution of our work on COLT is a policy for managing the overhead of on-line index selection. A major source of overhead for an on-line tuning algorithm comes from extra calls to the query optimizer to determine the benefit of hypothetical changes to the index configuration. This activity is often referred to as *what-if optimization*. The BC algorithm controls the overhead of what-if optimization by using a technique that approximates the behavior of the optimizer with lower overhead. In contrast, the approach of COLT carefully limits the number of invocations of the what-if optimizer. This is accomplished using several different techniques. First, COLT enforces a budget on the number of what-if optimizer invocations that may be performed per epoch. This budget may be increased or decreased based on the level of stability in the current workload. Second, we identify similarities between queries, and allow similar queries to inherit the results of the what-if optimizer from each other. The idea is that a good approximation of index benefits can be derived by using the what-if budget judiciously on dissimilar queries, since similar queries can share the results of the optimizer. The final way that overhead is controlled is by partitioning the index candidates into a *hot set* and a *cold set*. The hot set contains the indexes that seem the most promising, and only hot indexes are evaluated using the what-if optimizer. Members of the cold set may be promoted to the hot set if they perform well according to a crude (but efficient) metric.

The techniques used by COLT are interesting, and we conducted experiments to verify the benefits that it can provide to a dynamic workload. But like BC and QUIET, our system has considerable drawbacks. One drawback is that we only allow indexes on a single attribute to be created, but indexes with multiple attributes can have much higher benefits for some workloads. Also, in contrast to the BC algorithm, COLT does not reason directly about index interactions. Index interactions are clearly important in practice, so this aspect of COLT is a definite weakness.

One of the common criticisms of COLT is that there are too many parameters to set off-line. This is also an issue for the QUIET system. The most influential parameter of COLT is used by the benefit prediction function. This parameter is called the *averaging window*, and its value should be set to an integer A such that A queries are sufficient to get an accurate picture of the current query distribution. We used $A = 40$ as the default value in our experiments. This parameter is crucial because our model for benefit prediction looks at how the query distribution has changed in the recent past. Any periodic behavior with a period greater than A will essentially be ignored by COLT. Thus, it is possible to “break” COLT by submitting a workload where candidate indexes are useful every $A + 1$ queries. On the other hand, a large value of A will give too much importance to queries that are far in the past, and cause COLT to adapt very slowly to changes in the workload.

¹It is worth mentioning that our development of COLT was completely independent of QUIET, and there are coincidental similarities between the systems that are interesting.

3.2 A Retrospective Algorithm

Our work on COLT has introduced some valuable ideas, and we have learned a lot in the process. However, the flaws of COLT make it unsatisfactory for practical use. We observe that COLT and QUIET both require parameters to be configured off-line. It seems that this issue is difficult to avoid when we use the predictive approach. Since BC does not require any parameter tuning, it appears that the retrospective paradigm may help avoid this issue.

To get started in this direction, we have investigated some of the existing work in on-line computation. There is a general model which is called a *metrical task system* (MTS) in the literature [BLS92], and on-line index selection is closely related to this model. An MTS is allowed to be in any of N different states at any time, and the cost to transition between two states x and y is given by a distance metric $d(x, y)$. The job of the MTS is to complete a sequence of input tasks, and the cost to complete a task depends on the current state. Overall, the goal is to complete the tasks while minimizing the combined cost of tasks and state transitions.

We can naturally think of SQL queries as tasks, and a particular set of indexes as representing a state of the MTS. Then the cost to transition between two index configurations $d(C_1, C_2)$ is given by the cost to materialize the indexes in $C_2 - C_1$, if we consider index deletion to be free. This model almost works, except that the function $d(C_1, C_2)$ is not symmetric, meaning that d is not a distance metric. When the symmetry property fails, the system is generally referred to as a nonsymmetrical task system. The published algorithms for task systems almost exclusively consider the symmetric case, since it allows for some interesting algorithmic ideas. Many algorithms are difficult to adapt to the nonsymmetric case, making them unsuitable for index selection. With some work, we were able to adapt one MTS algorithm to nonsymmetrical task systems, which we describe below.

The general idea of a retrospective algorithm was described in Section 2. In the context of an MTS, a retrospective algorithm will determine what the optimal states would have been to complete the tasks seen so far (ignoring future tasks) and use this information to choose the next state. One of the applications of this idea is the *work function algorithm* [BEY98], or WFA. The application is not immediate, however: in addition to analyzing the optimal states in hindsight, WFA also balances the cost to go back to the current state when selecting the next state. As a result, WFA achieves a competitive ratio that is linear in the number of states.²

It is not immediately obvious from the original description of WFA whether it can be applied to the nonsymmetrical case. Indeed, a careful proof is needed to show that WFA is well defined, and this proof requires state transition costs to be symmetric. After some thought, we discovered that a small change in the algorithm allows the proof to go through, but the competitive ratio appears to be larger. Fortunately, due to the specific nature of transition costs in the index selection problem, we are able to prove that WFA achieves the same competitive ratio as the symmetric case when it is used for index selection. The proof of this fact requires the common assumption that index deletion is free.

We still have a major hurdle before we can apply WFA to our problem. The basic issue is that there are too many states if we consider each set of indexes to be its own state. A large number of states can cause WFA to have too much overhead cost to make decisions. Although we do not have a fully developed solution, we have made progress toward overcoming this obstacle. At a high level, our idea is to partition the candidate indexes into subsets, and treat each subset as a small task

²The proof of this competitive ratio does not consider the possibility that there are any patterns in the tasks submitted to the system. In many practical scenarios, the tasks will exhibit temporal locality, so the competitive ratio may be overly pessimistic.

system that can be handled separately. This will capture index interactions within each subset, but ignore interactions across subsets. If the partitioning is done wisely, the reduction in accuracy can be acceptable, and the overhead of index selection will be much lower. Once the missing pieces of this approach are filled in, we hope that it will be far more robust than the existing solutions for on-line index selection.

3.3 On-line Tuning Benchmark

There are several ways to illustrate the viability (or infeasibility) of an on-line tuning algorithm. It is certainly valuable to prove theoretical properties of an algorithm (such as a competitive ratio) and hand-worked examples can provide a lot of insight, but it is also crucial to verify the performance of any algorithm with an empirical study. The experiments that appear in published works have been quite varied, making them difficult to compare to one another. Indeed, two of the recent works [SLSS07, SAMP07] measure actual query execution times, while the experiments used by Bruno and Chaudhuri [BC07] are simulations that measure performance using cost estimates of the query optimizer, and do not reveal the true overhead of concurrent index selection. The algorithms are generally compared to a system with indexes that are chosen off-line; there has not yet been a study to compare different tuning algorithms.

These observations motivate an *on-line tuning benchmark* that can be used to compare existing and future algorithms for on-line index selection. My dissertation will include a description of such a benchmark as one of its contributions. The workload simulates a database hosting environment where several databases share the same server. There are some interesting aspects of this scenario. First, the schemata of the different databases may be designed in very different ways, e.g., one database might use a star schema and another might have a hierarchical schema. This heterogeneity creates a variety of challenges for the on-line tuning module. Second, the separation of the schemata allows for a natural way to create a dynamic workload, by shifting the popularity of the separate databases over time. If there is only one database, more manual work is required to create a dynamic workload that is realistic.

It is also interesting to incorporate “adversarial” test cases, similar to the examples described in Section 2. These microbenchmarks will be an excellent supplement to the more realistic database hosting environment already described, as they can expose some of the common pitfalls of on-line algorithms, while being simple enough to analyze the internal logic of the tuning algorithm.

This benchmark has been the focus of my most recent dissertation work. I worked closely with my advisor on the design and implementation of the database hosting benchmark, and we have set up the basic facilities necessary to run a wide variety of experiments. In order to use this benchmark to its full potential, it is also necessary to implement different algorithms on the same database platform—a nontrivial task due to the close integration that is required with the query optimizer. The BC algorithm is especially difficult to re-implement, since it was originally designed to use the features of Microsoft SQL Server, which has closely guarded source code. We have made significant progress to implement this technique in the PostgreSQL Database System, while staying consistent with the spirit of the original algorithm. The more salient details of this implementation will be included as part of my dissertation. Regarding the QUIET algorithm, the original implementation used an old version of Postgres, and I have very recently acquired source code from the authors. It is now a matter of porting the implementation to the most recent Postgres version, which is used for BC and COLT.

3.4 Semi-Automatic Index Selection

One of the fundamental aspects of on-line index selection is that the tuning module of the database works autonomously, creating and deleting indexes without any option for human intervention. This is very convenient for someone who does not have the expertise to be involved in index selection. On the other hand, an autonomous tuning module may be unacceptable for administrators who would like to have some degree of control over the indexes. An example would be a database where some of the queries are extremely critical, while others are not terribly important. This information cannot be gleaned from the query load, so human intervention may be necessary to avoid disastrous decisions. Since off-line tuning can be tedious, we would like to create a compromise between the on-line and off-line approaches. We refer to this approach as *semi-automatic index selection*.

In our vision of semi-automatic index selection, the system continuously monitors queries and gathers statistics that are very similar to the on-line tuning module. The administrator can invoke the semi-automatic tuner at any time to be presented with the current index recommendation, using the same logic as the retrospective on-line algorithm. The system also presents the user with evidence that the index is useful, such as recent queries that would benefit from the index. Next, the user chooses to reject and accept indexes. Due to index interactions, the user's decisions might affect other indexes in the configuration, so the recommendation would need to be regenerated, taking the user's constraints into account. This cycle repeats until the user is satisfied.

The semi-automatic system is by far the least developed portion of my dissertation, but we have a clear idea of the important steps needed to implement the system. Some of the main challenges are to present useful evidence in favor of the recommended indexes, and to incorporate user constraints in index selection. We will be able to make our ideas more concrete once the details of the retrospective algorithm are finalized.

4 Summary

Indexes can be crucial for a relational database to process queries with reasonable efficiency, but the selection of the best indexes is difficult. The problem is even more complex if the characteristics of the query load change over time, and this is a common case in practice. The goal of an on-line index selection algorithm is to choose indexes automatically in this type of dynamic environment, which can make database administration much more manageable.

In this proposal, I described two recent proposals for on-line index selection: the predictive algorithm QUIET, and the retrospective algorithm BC. These algorithms have their strengths, but do not suffice for a system that is useful in practice. My dissertation will provide a solution for on-line tuning that overcomes the weaknesses in existing algorithms, and thus provides a viable alternative to manual index selection. Towards this goal, I worked on a project to develop a predictive algorithm named COLT, which involved some worthwhile ideas but had limited success overall. My dissertation will use the lessons learned from COLT and prior research in on-line computation to develop a retrospective algorithm for index selection. This algorithm will be founded on stronger principles than existing solutions, and we expect it to be far more robust. In order to compare the alternative approaches to on-line index selection, my dissertation will include a benchmark for on-line tuning algorithms. The results that we gather will be an important contribution to the field, along with the methodology of the benchmark itself. Finally, my dissertation will explore an alternative user interface for on-line tuning that selects indexes in a semi-automatic fashion.

References

- [BC05] Nicolas Bruno and Surajit Chaudhuri. Automatic physical database tuning: a relaxation-based approach. In *ACM SIGMOD International Conference on Management of Data*, pages 227–238, 2005.
- [BC06] Nicolas Bruno and Surajit Chaudhuri. To tune or not to tune?: A lightweight physical design alerter. In *International Conference on Very Large Databases*, pages 499–510, 2006.
- [BC07] Nicolas Bruno and Surajit Chaudhuri. An online approach to physical design tuning. In *International Conference on Data Engineering*, pages 826–835, 2007.
- [BEY98] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.
- [BLS92] Allan Borodin, Nathan Linial, and Michael E. Saks. An optimal on-line algorithm for metrical task system. *Journal of the ACM*, 39(4):745–763, 1992.
- [BPS90] Elena Barcucci, R. Pinzani, and Renzo Sprugnoli. Optimal selection of secondary indexes. *IEEE Transactions on Software Engineering*, 16(1):32–38, 1990.
- [CFM95] Alberto Caprara, Matteo Fischetti, and Dario Maio. Exact and approximate algorithms for the index selection problem in physical database design. *IEEE Transactions on Knowledge and Data Engineering*, 7(6):955–967, 1995.
- [CN97] Surajit Chaudhuri and Vivek R. Narasayya. An efficient cost-driven index selection tool for microsoft sql server. In *International Conference on Very Large Databases*, pages 146–155, 1997.
- [FON92] Martin R. Frank, Edward Omiecinski, and Shamkant B. Navathe. Adaptive and automated index selection in RDBMS. In *International Conference on Extending Database Technology*, pages 277–292, 1992.
- [FST88] S. Finkelstein, M. Schkolnick, and P. Tiberio. Physical database design for relational databases. *ACM Transaction on Database Systems*, 13(1):91–128, 1988.
- [HC76] Michael Hammer and Arvola Chan. Index selection in a self-adaptive data base management system. In *ACM SIGMOD International Conference on Management of Data*, pages 1–8, 1976.
- [LL71] Vincent Y. Lum and Huei Ling. An optimization problem on the selection of secondary keys. In *ACM Annual Conference*, pages 349–356, 1971.
- [LTN07] Sam Lightstone, Toby Teorey, and Tom Nadeau. *Physical Database Design: the database professional's guide to exploiting indexes, views, storage, and more*. Morgan Kaufmann, 2007.
- [PDA07] Stratos Papadomanolakis, Debabrata Dash, and Anastasia Ailamaki. Efficient use of the query optimizer for automated physical design. In *International Conference on Very Large Databases*, pages 1093–1104, 2007.

- [SAMP06] Karl Schnaitter, Serge Abiteboul, Tova Milo, and Neoklis Polyzotis. Colt: continuous on-line tuning. In *ACM SIGMOD International Conference on Management of Data*, pages 793–795, 2006.
- [SAMP07] Karl Schnaitter, Serge Abiteboul, Tova Milo, and Neoklis Polyzotis. On-line index selection for shifting workloads. In *International Workshop on Self-Managing Database Systems*, pages 459–468, 2007.
- [SLSS07] Kai-Uwe Sattler, Martin Lühring, Karsten Schmidt, and Eike Schallehn. Autonomous management of soft indexes. In *International Workshop on Self-Managing Database Systems*, pages 450–458, 2007.
- [SSG04] Kai-Uwe Sattler, Eike Schallehn, and Ingolf Geist. Autonomous query-driven index tuning. In *International Database Engineering and Applications Symposium*, pages 439–448, 2004.
- [VZZ⁺00] Gary Valentin, Michael Zuliani, Daniel C. Zilio, Guy M. Lohman, and Alan Skelley. DB2 advisor: An optimizer smart enough to recommend its own indexes. In *International Conference on Data Engineering*, pages 101–110, 2000.