Constrained Dynamic Physical Database Design

Hannes Voigt, Wolfgang Lehner
Dresden University of Technology
Database Technology Group
01062 Dresden, Germany
{hannes.voigt,wolfgang.lehner}@tu-dresden.de

Kenneth Salem
David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, Ontario N2l 3G1, Canada
kmsalem@uwaterloo.ca

Abstract

Physical design has always been an important part of database administration. Today's commercial database management systems offer physical design tools, which recommend a physical design for a given workload. However, these tools work only with static workloads and ignore the fact that workloads, and physical designs, may change over time. Research has now begun to focus on dynamic physical design, which can account for time-varying workloads. In this paper, we consider a dynamic but constrained approach to physical design. The goal is to recommend dynamic physical designs that reflect major workload trends but that are not tailored too closely to the details of the input workloads. To achieve this, we constrain the number of changes that are permitted in the recommended design. In this paper we present our definition of the constrained dynamic physical design problem and discuss several techniques for solving it.

1 Introduction

There has been a substantial amount of work on the problem of automating physical database design [11, 16, 2, 4, 5], and many database management systems now come with database design advisors [22, 23, 1]. Such advisors generally view database physical design as a static problem. Given a set of queries and updates describing the database workload, plus a storage capacity constraint, a design advisor recommends a set of physical database structures, such as indexes and materialized views, that will minimize the cost of executing the workload. However, such formulations of the physical design problem do not account for the fact that the database system's workload may change over time, and that it may be desirable to change the database physical design accordingly.

To address this shortcoming, some researchers have proposed *dynamic, on-line* approaches to the physical design problem [18, 20, 7, 8, 19, 21]. For example, Bruno and Chaudhuri [8] model the database workload as a sequence of queries and updates, and they propose a mechanism that monitors the workload and continuously adjusts the database physical design based on the queries and updates that it has observed so far. This is appealing because it attempts to automatically adjust the database physical design to account for changes in the workload over time. However, because it is an on-line mechanism, it can only consider that portion of the workload that it has already observed. It must predict the future behaviour of the workload based on the past, and does not exploit any *a priori* workload information that may be available.

In this paper, we consider a *dynamic, off-line* version of the physical design problem. We are given, in advance, a description of the database system workload consisting of a sequence of queries and updates, as well as a storage capacity constraint. Essentially, our goal is to recommend a series of physical designs which will result in efficient execution of the workload.

The dynamic, off-line physical design problem was first posed by Agrawal, Chu and Narasayya [3]. In their formulation of the problem, the input workload is a sequence of n queries and updates, and the output is a sequence of n physical designs, one for each statement in the workload. This is an ideal formulation for situations in which the given query sequence represents an exact characterization of the expected database system workload. For example, if a particular application program generates a specific sequence of potentially complex queries each time it runs, the recommended physical design sequence can be tailored to the needs of that application. The recommended design sequence might indicate that a particular index should be created prior to the execution of a specific query in the sequence, and then dropped in favor of a different index for a subsequent query.

In other common scenarios, however, the given input sequence may not reflect an exact characterization of the database system's workload. For example, we may cap-

ture a workload trace from a database system on a particular day, and expect a similar but not identical workload on other days. In that case, we can view the workload sequence not as an exact characterization of the workload, but rather as a representative of the type of workload that is anticipated. In such situations, we want to recommend a dynamic physical design that reflects trends in the input workload, but that is not fit too tightly to the exact input query sequence. This is the problem that we consider in this paper.

The remainder of this paper is structured as follows. We formulate a constrained variant of the off-line dynamic physical design problem in Section 2. The optimal solution is described in Section 3, followed by two heuristic solutions in Section 4 and an alternative optimal approach in section 5. Section 6 presents the results of some simple experiments with constrained dynamic designs. Section 7 provides a brief summary of related work, and Section 8) concludes.

2 Change-Constrained Physical Design

Following Agrawal, Chu, and Narasayya [3], we assume that we are given as input a *sequence* $[S_1, S_2, \ldots, S_n]$ of SQL statements. Our goal is to choose a sequence of physical designs $[C_1, C_2, \ldots, C_n]$, where C_i is the physical design that will be used for the execution of S_i. A physical design consists of a set of structures (e.g., indexes or materialized views) chosen from a set of candidate structures. There are several techniques that can be used to generate such candidates [9, 22]. Like Agrawal et al., we will not be concerned with the means by which they are determined.

We use $\text{EXEC}(S_i, C_i)$ to denote the cost of executing statement S_i under physical design C_i, and we use $\text{TRANS}(C_i, C_j)$ to denote the cost of changing the physical design from C_i to C_j. Finally, we assume that each physical design has a size, denoted by $\text{SIZE}(C_i)$. The off-line, dynamic optimization problem defined by Agrawal et al. is to choose the designs C_i such that the sequence execution cost

$$\sum_{i=1}^{n} \text{EXEC}(S_i, C_i) + \text{TRANS}(C_{i-1}, C_i)$$

is minimized, subject to the space constraint $\forall i : \text{SIZE}(C_i) \leq b$, where b is a given space bound.

As noted in Section 1, we would like to treat input sequence as a representative example of some unknown workload process and avoid fitting it too closely. There are various ways that one might consider doing this. For example, instead of starting with a single given workload sequence, one could require that a set of representative sequences be given. Another possibility would be to provide as input a more explicit representation (e.g., a state machine) of a workload generation model.

In this paper, we consider a simpler strategy. Like Agrawal et al., we assume that the input is a single sequence of SQL statements. However, in our problem definition we restrict the number of design changes that are allowed in the resulting sequence of physical designs.

Definition 1 (Constrained Dynamic Physical Design) *Given a database, a query sequence $[S_1, S_2, \ldots, S_n]$, an initial physical design C_0, a space bound b, and a change constraint k, find a sequence of physical designs $[C_1, C_2, \ldots, C_n]$ such that $\forall i : \text{SIZE}(C_i) \leq b$ and $C_{i-1} \neq C_i$ for at most k values of i, and the sequence execution cost is minimized.*

Note that the input does not specify the points at which the design is permitted to change. It specifies only the maximum number of changes allowed (k). We expect the optimization to determine the correct number of changes as well as when those changes should occur. By choosing $k < n$, we can ensure that the resulting design sequence cannot be tailored to fit every individual statement in the given workload.

How should k be chosen? It is important to note that it is *not* necessary to choose a small k to limit the cost of design changes. The impact of these costs is already accounted for by the $\text{TRANS}(C_i, C_j)$ terms in the sequence execution cost. Instead, the change constraint k can be viewed as a parameter that controls the tightness of fit of the resulting design sequence. Smaller values of k ensure a looser fit. In some situations, it may be possible to select a k based on domain knowledge of applications that generated the representative trace. For example, if we are aware of time-of-day phenomena that cause the workload to change at lunchtime and in the evening, we can choose a value of k equal to or a bit larger than the number of anticipated fluctuations. A more general strategy for choosing k would require some means of characterizing representativeness of the given workload trace. Although the tuning of k is an interesting issue, we will not pursue it further in this paper

3 Optimal Change-Constrained Designs

Agrawal et al. observed that the set of possible dynamic physical designs for a given workload and set of candidate structures is isomorphic to the set of paths in a *sequence graph*, and that the problem of finding an optimal dynamic physical design is equivalent to finding a shortest path in that graph [3]. Figure 1 illustrates the sequence graph for a workload sequence consisting of $n = 3$ statements, under the assumption that there is only a single candidate index, denoted by IX. The leftmost node in this graph represents the initial physical design, C_0, which we have assumed to be empty. Each of the following n columns, or stages, of

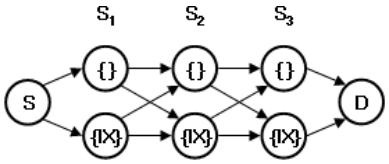

Figure 1: Sequence graph for three statements and one candidate index

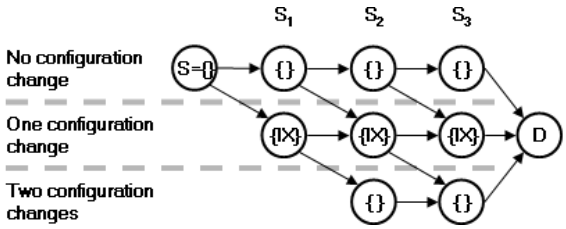

Figure 2: $(k = 2)$-aware sequence graph for three statements and one candidate index

nodes represents the possible physical designs for one of the n statements in the workload. Since there is only one candidate design structure, there are only two nodes in each stage, one representing the empty physical design ($\{\}$) and the other representing the physical design consisting of the index IX. In general, if there are m candidate physical design structures, there will be 2^m possible physical designs and hence 2^m nodes in each stage of the graph. The rightmost node is the destination node. It can serve to constrain the final configuration. We assume it to be unconstrained.

The node corresponding to physical design C_i in stage x of the sequence graph is labelled with $\text{EXEC}(S_x, C_i)$. The edge from configuration C_i in stage x to configuration C_j in stage $x+1$ is labelled with $\text{TRANS}(C_i, C_j)$. The optimal dynamic physical design then corresponds to the weighted shortest path through the sequence graph from the source node to the destination node, with the weights determined by the node and edge labels.

The shortest path through a directed acyclic graph can be found in $O(|V| + |E|)$ time, where $|V|$ is the number of nodes and $|E|$ is the number of edges [12]. A sequence graph has $|V| = n2^m + 2$ nodes and $|E| = (n-1)2^{2m} + 2^{m+1}$ edges, so a shortest path can be found in $O(n2^{2m})$ time.

We generalized sequence graphs to solve our change-constrained dynamic physical design problem. These generalized graphs called k-aware sequence graphs in the following. Figure 2 illustrates the k-aware sequence graph for the same scenario used in Figure 1, for $k = 2$. Like the original sequence graph, the k-aware graph contains one stage of nodes for each statement in the workload sequence. However, the k-aware graph is also layered, as illustrated by the horizontal divisions in Figure 2. The layers are used to encode the number of physical design changes that have occurred along the paths through the graph. For example, consider the node representing physical design C_i at stage x in layer l. This node represents the use of design C_i to execute statement S_x, and that a total of l design changes have been made up to and including stage x. This node will have an outgoing edge to the node representing C_i in stage $x+1$ at layer l, which represents the possibility that design C_i will also be used to execute S_{x+1}, and that the total number of design changes will remain unchanged at l should that occur. This node will have $2^m - 1$ outgoing edges to designs C_j $(j \neq i)$ at stage $x+1$ in layer $l+1$. These represent the possibility that S_{x+1} is executed using a different design than S_x, and that the number of design changes will have increased from l to $l+1$ as a result.

The set of paths from source to destination through the k-aware sequence graph with $k+1$ layers is isomorphic to the set of dynamic physical designs with at most k design changes, i.e., to the set of possible solutions to the constrained dynamic physical design problem. A k-aware sequence graph with $k+1$ layers contains $O(kn2^m)$ nodes and $O(kn2^{2m})$ edges. Thus, an optimal solution to the constrained dynamic physical design problem with a change bound of k can be found in $O(kn2^{2m})$ time.

4 Heuristic Solutions

Unless m is very small, the shortest-path-based algorithms for both the unconstrained and constrained versions of the dynamic physical design problem are probably impractical because they are exponential in m. In this section, we discuss two more efficient but heuristic approaches to the constrained design problem.

4.1 Greedy-Seq

Agrawal et al. proposed a greedy heuristic called GREEDY-SEQ that produces an optimized dynamic physical design in polynomial time, but does not guarantee the optimality of the recommended design. The shortest-path-based algorithm for finding optimal dynamic physical designs is exponential because it considers an exponential (in m) number of candidate physical designs. The idea behind GREEDY-SEQ is to identify a much smaller set of candidate physical designs, and then run the shortest-path-based algorithm on that smaller set.

This same approach can be used to obtain a good solution to the constrained dynamic physical design problem. In the constrained case, the set of candidate designs can be generated exactly as in the GREEDY-SEQ approach, in $O(nm^2)$ time. Once the candidates have been determined, we can generate a k-aware sequence graph using those candidates

and obtain a constrained dynamic physical design. Since GREEDY-SEQ generates $O(mn)$ candidate physical designs, the resulting k-aware sequence graph will have $O(kn^2m)$ nodes and $O(kn^3m^2)$ edges, and the constrained shortest path can be found from these candidates in $O(kn^3m^2)$ time.

4.2 Sequential Design Merging

Another way to solve the constrained physical design problem is to start with a solution to the unconstrained problem and then refine that solution until it satisfies the constraints. In this section we describe one such approach. Suppose that we have a solution to the unconstrained dynamic physical design problem, and that the solution includes l design changes. We would like a solution to the constrained design problem for some $k < l$. We will transform the given solution into a solution to the constrained problem in at most $l - k$ steps, where each step reduces the number of design changes by at least one. This approach is heuristic, so it is not guaranteed to produce an optimal solution to the constrained problem, even if the initial design is optimal for the unconstrained problem.

In each step we choose a sequence of two consecutive distinct configurations $\langle C_i, C_{i+1}\rangle$ from the current dynamic physical design and replace those two configurations with a new configuration C'_i chosen from the same set of candidate configurations that was used to generate the original, unconstrained design sequence. We choose the replacement configuration C'_i so that

$$\text{TRANS}(C_{i-1}, C'_i)+\text{EXEC}(S_i \cup S_{i+1}, C'_i)+\text{TRANS}(C'_i, C_{i+2})$$

is minimal. The appropriate C'_i can be found by enumerating all candidate configurations. Note that if $C'_i = C_{i-1}$ or $C'_i = C_{i-2}$, then replacing $\langle C_i, C_{i+1}\rangle$ with C'_i would reduce the number of design changes by two. Otherwise, the number is reduced by one.

The penalty p associated with replacing $\langle C_i, C_{i+1}\rangle$ with the configuration C'_i is

$$\begin{aligned} p \;=\; & (\text{TRANS}(C_{i-1}, C'_i) + \text{EXEC}(S_i \cup S_{i+1}, C'_i) \\ & +\text{TRANS}(C'_i, C_{i+2})) \\ - \; & (\text{TRANS}((C_{i-1}, C_i) + \text{EXEC}(S_i, C_i) \\ & +\text{TRANS}(C_i, C_{i+1}) \\ & +\text{EXEC}(S_{i+1}, C_{i+1}) + \text{TRANS}(C_{i+1}, C_{i+2})) \end{aligned}$$

From among all pairs $\langle C_i, C_{i+1}\rangle$ in the given sequence, we choose the pair for which the penalty is smallest and replace the pair with the appropriate C'_i to produce a dynamic design with fewer changes.

Consider the following simple example. We are given a sequence of $n = 3$ statements, the set of candidate configurations is $\{\emptyset, \{IX\}\}$, the initial configuration is $\emptyset$, and the change bound is $k = 1$. Suppose that the best unconstrained design is $[\emptyset, \{IX\}, \emptyset]$, which includes $l = 2$ configuration changes. With one merging step we can produce design with at most $k = 1$ configuration changes. The merging step identifies the best single configuration to replace the sequence $\langle\emptyset, \{IX\}\rangle$ and the best single configuration to replace the sequence $\langle\{IX\}, \emptyset\rangle$. Call these configurations C'_1 and C'_2, respectively. Then we can easily determine the replacement penalty of each of these choices. If C'_1 has the smaller penalty, we will replace $\langle\emptyset, \{IX\}\rangle$ with $\langle C'_1, C'_1\rangle$. Otherwise, we will replace $\langle\{IX\}, \emptyset\rangle$ with $\langle C'_2, C'_2\rangle$.

Assuming that the current design sequence includes x design configuration changes, and that there are 2^m candidate design configurations, a single step of the merging algorithm will required $O(x2^m)$ time, since each candidate must be checked as a potential replacement for a pair of consecutive designs. If the initial, unconstrained design includes l design changes and a design with at most $k < l$ changes is desired, then $(l - k)$ merging steps will be required, resulting in a total time complexity of

$$\sum_{x=k}^{l} O(x2^m) = O(2^m(l^2 - k^2))$$

Note that if the initial unconstrained design is generated using fewer than 2^m candidate designs, then the complexity of sequential merging is reduced accordingly.

5 Ranking shortest paths

Our constrained dynamic physical design problem can also be seen as an instance of the constrained shortest path problem. A very simple and general solution to the constrained shortest path problem is to turn it into the shortest path ranking problem (mentioned inter alia in [15]). Shortest path ranking algorithms generate paths in ascending order of length until a given stopping condition is reached. For our purposes, we rank paths until we found one having k or fewer designs. In other words, we generate and check possible design sequences in order of their total estimated costs. The first sequence containing k or fewer designs is the optimal solution to the constrained problem, since we have seen only sequences with more than k designs and any sequence we would see if we went on is by definition longer.

Shortest path ranking has been well studied. One of the fastest algorithms is based on the idea of adding to the graph nodes that represent possible alternatives to the previous shortest path and deleting that path, so that it is no longer a possible solution. This is known as path deletion [14]. The initial shortest path is determined with the normal shortest path algorithm. The complexity of finding the next shortest path in the ranking depends on the number of nodes in the previous shortest path and on the number of their incoming

edges. In a sequence graph, the number of nodes in a path is always n. The number of incoming edges per node will not exceed number of candidate configurations 2^m. Thus, one iteration requires $O(n2^m)$ time.

We have $2^m(2^m-1)^{k-1}$ ways to pick a sequence of k designs out of 2^m candidates and $\binom{n-1}{k-1}$ ways to pick $k-1$ design change points. Hence, $2^m(2^m-1)^{k-1}\binom{n-1}{k-1}$ solutions have exactly k designs and $\sum_{i=1}^{k} 2^m(2^m-1)^{i-1}\binom{n-1}{i-1}$ have k or fewer designs. In the worst case all possible paths with more than k designs are shorter than those with k or fewer designs, so the worst case time complexity is $O((n2^m)(2^{mn} - \sum_{i=1}^{k} 2^m(2^m-1)^{i-1}\binom{n-1}{i-1}))$. Although the worst case can be quite bad, particularly for small k, in practice this approach may be better. It can also be used as a means of generating an initial design sequence for use with the sequential merging technique that was discussed in Section 4.2.

6 Experiments

We conducted a series of experiments that were primarily intended to illustrate the behavior of a constrained dynamic physical design optimizer. In particular, we are interested in contrasting the dynamic physical designs produced by a constrained optimizer with those produced by an unconstrained optimizer. We have also compared the costs of generating constrained designs to those of generating unconstrained designs.

6.1 Experimental Setup

Our experiments were conducted on a machine with an Intel Xeon 2.80 GHz dual core processor and 4GB of RAM, running under Microsoft Windows 2003. We used Microsoft's SQL-Server 2005 as our database system. We defined a test database consisting of a single table with four integer columns (a,b,c,d) and 2.5 million rows. The table was populated using independently and uniformly selected random values in the range $[0, 500000)$.

We constructed workloads using simple SQL point queries of the form

```
SELECT <col> FROM ...
WHERE <col> = <randValue>
```

Using this template, we created specific queries by substituting one of the four column names for `<col>` and a randomly selected integer in the range $[0, 500000)$ for `<randValue>`. Using queries of this form, we defined the four different query mixes described in Table 1. For example, Query Mix A consists of randomly generated queries of the above form, with 55% of the queries against column a, 25% against column b, 10% against column c, and 10% against column d. To build time-varying workloads, we periodically switch among the query mixes defined in Table 1. These workloads are not intended to be representative of the workloads produced by any particular application. Rather, they are intended to provide us with a simple and controlled means of exercising the dynamic physical design advisors.

	Queried $< col >$			
	a	b	c	d
Query Mix A	55%	25%	10%	10%
Query Mix B	25%	55%	10%	10%
Query Mix C	10%	10%	55%	25%
Query Mix D	10%	10%	25%	55%

Table 1: Workload Query Mixes

In our experiments, we restricted the design advisors to a small design space to simplify our presentation. Specifically, a physical design configuration consists of at most one index, and the available indexes are single-column indexes on each of the four columns, denoted by $I(a)$, $I(b)$, $I(c)$, and $I(d)$, and two additional two-column indexes: $I(a, b)$ and $I(c, d)$. Thus, there are a total of seven possible configurations, including the empty configuration. In all of our experiments, we fixed the initial and final configuration to be empty.

6.2 Constrained Designs

For our first experiment, we defined a dynamic workload W_1 with three major phases. We call changes from one phase to another *major shifts*. In addition, the workload exhibits smaller fluctuations, called *minor shifts*, within each phase.

The second column of Table 2, which is labeled W_1, defines this workload. The first phase of W_1 lasts for 5000 queries, at which point a major shift to the second phase occurs. The second phase lasts for 5000 queries, followed by a major shift to the third phase. Within each phase, there are minor shifts every 1000 queries. Specifically, in the first and third phases we alternate between query mixes A and B, while in the second phase we alternate between query mixes C and D.

In this experiment, we directly compare the dynamic physical design produced for W_1 by an unconstrained design advisor with the design generated for W_1 by a constrained advisor with $k = 2$. Note that we have chosen the constraint k to match the number of major shifts in W_1. We expect to see the unconstrained design closely track both the major and minor workload shifts. In contrast, we expect to see the constrained design track only the major shifts and ignore the minor ones.

The third column of Table 2, labeled $k = \infty$, describes the recommended unconstrained dynamic physical design.

query number	W_1	design $k=\infty$	design $k=2$	W_2	W_3
1-500	A	$I(a,b)$	$I(a,b)$	A	B
501-1000	A	$I(a,b)$	$I(a,b)$	B	B
1001-1500	B	$I(b)$	$I(a,b)$	A	A
1501-2000	B	$I(b)$	$I(a,b)$	B	A
2001-2500	A	$I(a,b)$	$I(a,b)$	A	B
2501-3000	A	$I(a,b)$	$I(a,b)$	B	B
3001-3500	B	$I(b)$	$I(a,b)$	A	A
3501-4000	B	$I(b)$	$I(a,b)$	B	A
4001-4500	A	$I(a,b)$	$I(a,b)$	A	B
4501-5000	A	$I(a,b)$	$I(a,b)$	B	B
5001-5500	C	$I(c,d)$	$I(c,d)$	C	D
5501-6000	C	$I(c,d)$	$I(c,d)$	D	D
6001-6500	D	$I(d)$	$I(c,d)$	C	C
6501-7000	D	$I(d)$	$I(c,d)$	D	C
7001-7500	C	$I(c,d)$	$I(c,d)$	C	D
7501-8000	C	$I(c,d)$	$I(c,d)$	D	D
8001-8500	D	$I(d)$	$I(c,d)$	C	C
8501-9000	D	$I(d)$	$I(c,d)$	D	C
9001-9500	C	$I(c,d)$	$I(c,d)$	C	D
9501-10000	C	$I(c,d)$	$I(c,d)$	D	D
10001-10500	A	$I(a,b)$	$I(a,b)$	A	B
10501-11000	A	$I(a,b)$	$I(a,b)$	B	B
11001-11500	B	$I(b)$	$I(a,b)$	A	A
11501-12000	B	$I(b)$	$I(a,b)$	B	A
12001-12500	A	$I(a,b)$	$I(a,b)$	A	B
12501-13000	A	$I(a,b)$	$I(a,b)$	B	B
13001-13500	B	$I(b)$	$I(a,b)$	A	A
13501-14000	B	$I(b)$	$I(a,b)$	B	A
14001-14500	A	$I(a,b)$	$I(a,b)$	A	B
14501-15000	A	$I(a,b)$	$I(a,b)$	B	B

Table 2: Dynamic Workloads and Physical Designs

As expected, the design changes with each minor workload shift. Between each minor shift, the design consists of the index that provides benefit to the dominant query type in the query mix during that interval. The fourth column of Table 2, labeled $k = 2$, shows the constrained design. As anticipated, it reflects only the major shifts in the workload. Note that the unconstrained design is, by definition, an optimal design for workload W_1, and the constrained design is suboptimal. Nonetheless, the unconstrained design may be interesting because it is suitable for other workloads that are similar to, but not identical to, W_1. We illustrate this in the next experiment.

6.3 Workload Variations

In this experiment, we constructed two additional workloads, W_2 and W_3 and measured the performance of those workloads using the dynamic physical designs that were recommended based on W_1. Workloads W_2 and W_3 are similar to W_1, in that they have the same major shifts and three phase structure as W_1. However, W_2 and W_3 have different minor shifts than W_1. W_2 and W_3 are defined in the two rightmost columns of Table 2. Workload W_2 undergoes more frequent minor shifts than W_1, e.g., in the first phase it alternates between Query Mix A and Query Mix B every 500 queries, rather than every 1000 queries. Workload W_3 has the same number of minor shifts as W_1, but their query mixtures are out of phase, e.g., W_3 uses query mix B when W_1 uses query mix A, and vice versa.

Figure 3 shows the total execution time for each of the three workloads using both the constrained and unconstrained designs that were recommended based on W_1. All of the times are shown relative to the execution time of W_1 under the optimal, unconstrained dynamic physical design.

As was indicated in Section 6.2, the constrained W_1-based design is suboptimal for W_1. As shown in Figure 3, W_1 is 14% slower under the constrained design. However, the situation is different for W_2 and W_3. Both of these workloads are better off using the constrained design than the unconstrained design, because the constrained design is not tied as tightly as the unconstrained design to the original W_1 workload.

Of course, these workloads were contrived for the purposes of this experiment, and we can not claim that the kinds of fluctuations exhibited by these workloads reflect any realistic application. Our objective is simply to illustrate that a constrained dynamic physical design that does not fit the design workload too closely may be beneficial for other, similar workloads.

6.4 Design Optimization Cost

In our final experiment, we consider the cost of producing constrained dynamic physical designs. We implemented both the optimal technique described in Section 3 and the sequential design merging heuristic described in Section 4.2. Figure 4 shows the time required to generate a design recommendation for each of these techniques, as a function of the change constraint k. These times are shown relative to the cost of generating an optimal unconstrained design recommendation.

The time required to generate optimal constrained design recommendations increases linearly with k, as expected. This is because the size of the k-aware sequence graph grows with k. In contrast, the time required for the merging heuristic is inversely related to k, since a larger k means that fewer merging steps are required. Together, this suggests that a hybrid technique that switches to the merging approach for larger k will be an appropriate means of generating constrained designs.

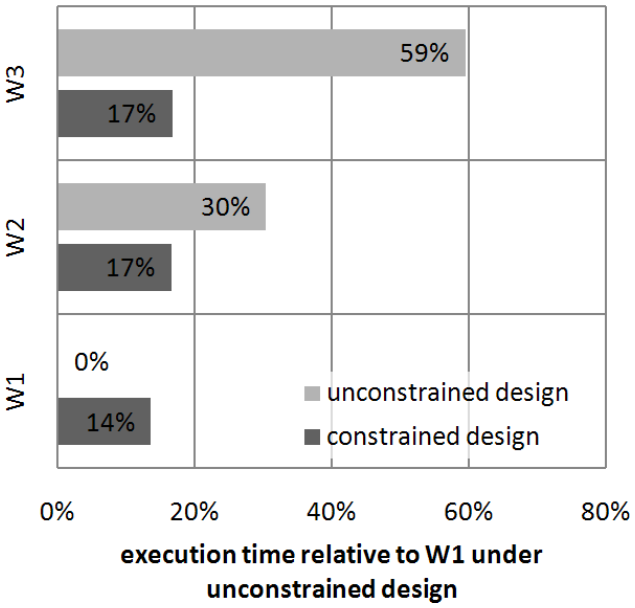

Figure 3: Relative Execution Times of Different Workloads Under Constrained and Unconstrained W_1 Designs

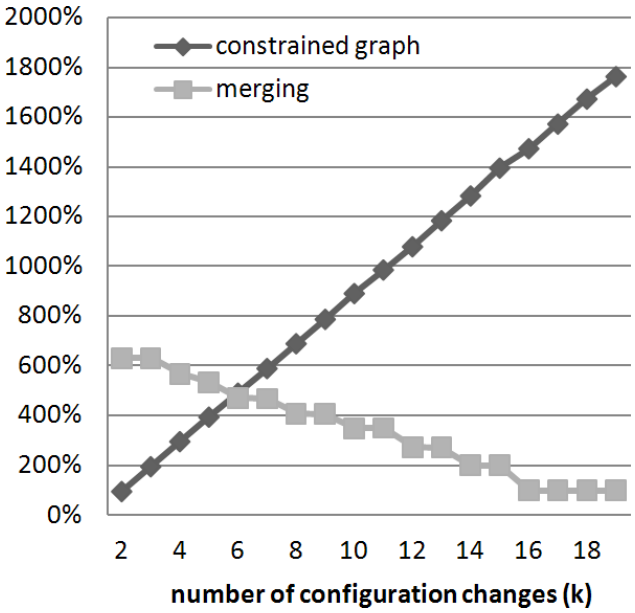

Figure 4: Runtimes of Constrained Design Optimizers Relative to Runtime of Unconstrained Design Optimizer

7 Related Work

As was noted in Section 1, there has been a substantial amount of work done on the problem of automatic physical design tuning. The classical and well understood approach results in advisory tools that take a workload description (a set of SQL queries) as input and return a recommended static physical design. Initial solutions only considered the problem of index selection [11, 16, 17, 9, 22] and recommendation of a set of materialized views [2], while state-of-the-art techniques [1, 13, 23] consider a broad range of database objects, such as indexes, materialized views, and partitions. All of these techniques assume a static workload and leave it to the database administrator to trigger the advisory tool in order to compute a new configuration. A first step towards a dynamic system is incorporated in design refinement strategies [10, 5, 6] by incrementally updating configurations in order to partially reflect changes in the workload. However, the techniques do not tackle the problem of computing the 'best' number of anticipated changes. Rather, they focus on the problem of merging existing and new configurations.

While offline physical design advisor tools can already be found in commercial solutions, research has turned its focus towards on-line design tuning [18, 20, 7, 8, 19, 21]. The basic idea is to constantly monitor the database system, analyze the current workload, and adjust the physical design accordingly. Design alerters [7] periodically check the quality of the existing physical configuration and send an alert to the database administrators if the quality appears to be deteriorating. Within our framework, we might rely on these technologies to trigger an off-line dynamic optimizer such as the one presented here.

8 Conclusion

In this paper, we defined the notion of *change-constrained dynamic physical design*. The objective is a dynamic physical design method that reflects trends in a given workload one the one hand, but does not fit the exact input query sequence too tightly. We presented several techniques for solving the problem, including an optimal solution based on k-aware sequence graphs.

There are a number of open questions that arise from this work. One question is how to choose an appropriate change constraint (k). A second question is how to characterize scenarios or classes of workloads for which constrained dynamic physical designs will be beneficial.

References

[1] S. Agrawal, S. Chaudhuri, L. Kollár, A. P. Marathe, V. R. Narasayya, and M. Syamala. Database tuning advisor for microsoft sql server 2005. In *Proceedings of 30th International Conference on Very Large Data Bases, August 31 - September 3, 2004, Toronto, Canada*, pages 1110–1121, 2004.

[2] S. Agrawal, S. Chaudhuri, and V. R. Narasayya. Automated selection of materialized views and indexes in sql databases.

In *Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt*, pages 496–505, 2000.

[3] S. Agrawal, E. Chu, and V. R. Narasayya. Automatic physical database tuning: Workload as a sequence. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, USA, June 27-29, 2006*, pages 683–694, 2006.

[4] S. Agrawal, V. R. Narasayya, and B. Yang. Integrating vertical and horizontal partitioning into automated physical database design. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data, Paris, France, June 13-18, 2004*, pages 359–370, 2004.

[5] 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, Baltimore, Maryland, USA, June 14-16, 2005*, pages 227–238, 2005.

[6] N. Bruno and S. Chaudhuri. Physical design refinement: The "merge-reduce" approach. In *10th International Conference on Extending Database Technology, Munich, Germany, March 26-31, 2006, Proceedings*, pages 386–404, 2006.

[7] N. Bruno and S. Chaudhuri. To tune or not to tune? a lightweight physical design alerter. In *Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12-15, 2006*, pages 499–510, 2006.

[8] N. Bruno and S. Chaudhuri. An online approach to physical design tuning. In *Proceedings of the 23nd International Conference on Data Engineering, April 15-20, 2007, The Marmara Hotel, Istanbul, Turkey*, pages 826–835, 2007.

[9] S. Chaudhuri and V. R. Narasayya. An efficient cost-driven index selection tool for microsoft sql server. In *Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece*, pages 146–155, 1997.

[10] S. Chaudhuri and V. R. Narasayya. Index merging. In *Proceedings of the 15th International Conference on Data Engineering, 23-26 March 1999, Sydney, Austrialia*, pages 296–303, 1999.

[11] D. Comer. The difficulty of optimum index selection. *ACM Transactions on Database Systems*, 3(4):440–445, 1978.

[12] T. H. Cormen, C. E. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition*. MIT Press, 2001.

[13] B. Dageville, D. Das, K. Dias, K. Yagoub, M. Zaït, and M. Ziauddin. Automatic sql tuning in oracle 10g. In *Proceedings of 30th International Conference on Very Large Data Bases, August 31 - September 3, 2004, Toronto, Canada*, pages 1098–1109, 2004.

[14] J. A. de Azevedo, J. J. E. R. S. Madeira, E. de Queirs Vieira Martins, and F. M. A. Pires. A shortest paths ranking algorithm. In *Proceedings of the Annual Conference AIRO'90, Models and Methods for Decision Support, Operational Research Society of Italy*, pages 1001–1011, 1990.

[15] D. Eppstein. Finding the k shortest paths. In *Proceedings of 35th Symposium Foundations of Computer Science, 20-22 November 1994, Santa Fe, New Mexico, USA*, pages 154–165, 1994.

[16] S. J. Finkelstein, M. Schkolnick, and P. Tiberio. Physical database design for relational databases. *ACM Transactions on Database Systems*, 13(1):91–128, 1988.

[17] H. Gupta, V. Harinarayan, A. Rajaraman, and J. D. Ullman. Index selection for olap. In *Proceedings of the Thirteenth International Conference on Data Engineering, April 7-11, 1997, Birmingham, U.K*, pages 208–219, 1997.

[18] K.-U. Sattler, I. Geist, and E. Schallehn. Quiet: Continuous query-driven index tuning. In *Proceedings of 29th International Conference on Very Large Data Bases, September 9-12, 2003, Berlin, Germany*, pages 1129–1132, 2003.

[19] K.-U. Sattler, M. Luehring, K. Schmidt, and E. Schallehn. Autonomous management of soft indexes. In *2nd International Workshop on Self-Managing Database Systems, 16 April 2007, Istanbul, Turkey*, 2007.

[20] K. Schnaitter, S. Abiteboul, T. Milo, and N. Polyzotis. Colt: continuous on-line tuning. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, USA, June 27-29, 2006*, pages 793–795, 2006.

[21] K. Schnaitter, S. Abiteboul, T. Milo, and N. Polyzotis. On-line index selection for shifting workloads. In *2nd International Workshop on Self-Managing Database Systems, 16 April 2007, Istanbul, Turkey*, 2007.

[22] G. Valentin, M. Zuliani, D. C. Zilio, G. M. Lohman, and A. Skelley. Db2 advisor: An optimizer smart enough to recommend its own indexes. In *Proceedings of the 16th International Conference on Data Engineering, 28 February - 3 March, 2000, San Diego, California, USA*, pages 101–110, 2000.

[23] D. C. Zilio, J. Rao, S. Lightstone, G. M. Lohman, A. Storm, C. Garcia-Arellano, and S. Fadden. Db2 design advisor: Integrated automatic physical database design. In *Proceedings of 30th International Conference on Very Large Data Bases, August 31 - September 3, 2004, Toronto, Canada*, pages 1087–1097, 2004.