Adaptive Physical Design for Curated Archives

Tanu Malik[1], Xiaodan Wang[2], Debabrata Dash[3],
Amitabh Chaudhary[4], Anastasia Ailamaki[5], and Randal Burns[2]

[1] Purdue University, USA
tmalik@purdue.edu
[2] Johns Hopkins University, USA
{xwang,randal}@cs.jhu.edu
[3] Carnegie Mellon University, USA
ddash@cs.cmu.edu
[4] University of Notre Dame, USA
achaudha@cse.nd.edu
[5] Swiss Federal Institutes of Technology, Switzerland
anastasia.ailamaki@epfl.ch

Abstract. We introduce AdaptPD, an automated physical design tool that improves database performance by continuously monitoring changes in the workload and adapting the physical design to suit the incoming workload. Current physical design tools are offline and require specification of a representative workload. AdaptPD is "always on" and incorporates online algorithms which profile the incoming workload to calculate the relative benefit of transitioning to an alternative design. Efficient query and transition cost estimation modules allow AdaptPD to quickly decide between various design configurations. We evaluate AdaptPD with the SkyServer Astronomy database using queries submitted by SkyServer's users. Experiments show that AdaptPD adapts to changes in the workload, improves query performance substantially over offline tools, and introduces minor computational overhead.

1 Introduction

Automated physical design tools are vital for large-scale databases to ensure optimal performance. Major database vendors such as Microsoft, IBM, and Oracle now include tuning and design advisers as part of their commercial offerings. The goal is to reduce a DBMS' total cost of ownership by automating physical design tuning and providing DBAs with useful recommendations about the physical design of their databases. However, current tools [1,2,3,1] provide limited automation; they take an offline approach to physical design and leave several significant decisions during the tuning process to DBAs. Specifically, DBAs need to explicitly specify representative workloads for the tuning tool. DBAs are also required to know when a tuning session is needed and guesstimate the relative benefit of implementing the recommendations.

Complete automation is a critical requirement of libraries which will soon become data centers for curation of large scientific data. A notable example is

M. Winslett (Ed.): SSDBM 2009, LNCS 5566, pp. 148–166, 2009.

the Sloan Digital Sky Survey (SDSS) [4] project whose data will soon be curated. The project receives a diverse workload, which exceeds a million queries every month. As such, finding a representative workload is challenging because query access patterns exhibit considerable evolution within a week [5].

The straightforward approach of running an offline tool after each query or invoking it periodically for continuous evaluation of physical design achieves most of the automation objectives. However, this approach requires further tuning by a DBA to ensure that the tool does not react too quickly or slowly and result in poor design choices. Recent research [6,7,8] on the online physical design problem focuses on index design. Bruno and Chaudhari [8] infer costs and plan properties during the query optimization phase to efficiently decide between various index configurations in an online fashion. In this paper, we focus on vertical partitioning, which is complementary to index selection. In SDSS and other scientific databases, vertical partitioning is often used because it does not replicate data, thereby reducing space requirements [1].

Contributions. To provide complete automation, we model the physical design problem in AdaptPD as an online problem and develop algorithms that minimize the combined cost of query execution and the cost of transitioning between configurations. We also develop efficient and accurate cost estimation modules that reduce the overhead of AdaptPD. AdaptPD is evaluated within the Astronomy database of SDSS. Experiments indicate up to two fold improvement in query response time when compared with offline tuning techniques.

We develop online algorithms that search the space of physical design alternatives without making assumptions about the workload. Analysis shows that the algorithm provides a minimum level of guarantee of adapting to workload changes. Current tools provide such guarantees only for two configurations.

Our algorithms assume a general transition model in which transition costs between configurations are asymmetric and positive. This is in contrast to current works for index design, which assume a constant cost of creating a physical design structure and zero cost of deleting them [8]. We validate our model through experiments and show that transition costs are asymmetrical and the asymmetry is bounded by a constant factor.

We develop a novel "cache-and-reuse" technique for query cost estimation. The technique caches distinct query plans that do not change across several configurations and reuses the plans for estimating query costs. By reusing cached plans, the technique minimizes computationally-intensive optimizer invocations by as much as 90%. Current tools, both offline and online, employ no such methods for query estimation and are therefore much slower to run. We also develop the first-known technique, based on bulk-inserts, for estimating the cost of transitioning from one configuration to another. In current online tools, transition costs are either fixed or assigned arbitrarily.

Our online vertical partitioning techniques have applicability beyond the automation of curated relational databases. For example, our algorithm for the regrouping of columns can also provide automation for column-store databases [9]. In particular, the algorithm is independent of whether the database is

implemented as a row-store or a column-store. The techniques also have applicability in schema design of proxy database caches. We recently showed that inefficiencies in the physical design of cached objects offsets some of the benefits of deploying a cache and automated physical design can recoup that loss [10].

2 Related Work

Automated physical design tools use cost estimates from the optimizer or analytical I/O cost-based models to evaluate attribute groupings [1,2,3] for a given query workload. These solutions are offline, i.e., they assume a priori knowledge of the entire workload stream and therefore provide a single, static physical design for the entire workload.

Current research [7,8] emphasizes the need for automated design tools that are *always-on* and, as new queries arrive, continuously adapt the physical design to changes in the workload [11]. Quiet [6] describes an incremental adaptive algorithm for index selection which is not fully integrated with the optimizer. In Colt [7], Schnaitter et al. present a similar algorithm which relies heavily on the optimizer for cost estimation. Both approaches do not take into account transition costs. Bruno et al. present a formal approach to online index selection [8] that takes into account transition costs. Their algorithms are limited to choosing among configurations in which the only difference is the set of indices being used. Our core algorithm is general purpose in that physical design decisions are not limited to index selection. In this paper, the system is developed for configurations that are vertical partitions. We also assume that transition costs are asymmetric which is not the case in [8].

Our formulation is similar to that of task systems introduced by Borodin et al. [12]. Task systems have been researched extensively, particularly when the transition costs form a metric [12]. Our costs are not symmetric and do not form a metric. This asymmetry in transition costs exists because the sequence of operations (i.e. insertion or deletion of tables or columns) required for making physical design changes in a database exhibit different costs. The Work-Function algorithm [13] is an online algorithm for such asymmetrical task systems, but it is impractical with respect to the efficiency goals of AdaptPD. The algorithm solves a dynamic program with each query that takes $\theta(N^2)$ time, even in the best case, in which N is the number of configurations. In AdaptPD we present a simpler algorithm that takes $O(N)$ time at each step in the worst case.

Read-optimized column-stores have been used for commercial workloads with considerable success [9,14]. They perform better than row-stores by storing compressed columns contiguously on disk. Column-stores, however, pose several hurdles for SDDS implementation. The implementation is well-optimized for commercial row-store databases on existing workloads and a complete migration to column-store is prohibitively expensive. Moreover, it consists of mostly floating point data that are not compressible using the RLE and bitmap compression schemes used by column-stores, thereby eliminating a crucial advantage of column-store. Our solution is an intermediate step at the storage-layer that

performs workload-based regrouping of columns on a row-store and avoids increased tuple reconstruction cost associated with a column-store.

Our query cost estimation module is similar to other configuration parametric query optimization techniques, such as INUM [15], and C-PQO [16]. These techniques exploit the fact that the plan of a query across several configurations is an invariant and can be reused to reduce optimizer calls. These techniques reuse the plans when configurations are limited to sets of indices on tables. We extend plan reuse to configurations that correspond to vertical partitions.

3 The AdaptPD Tool

The AdaptPD tool automates several tasks of a DBA. The DBA often performs the following tasks to maintain a workload-responsive physical design: a) Identifies when workload characteristics have changed significantly such that the current physical design is no longer optimal. b) Chooses a new physical design such that excessive costs are not incurred in moving from the current physical design, relative to the benefit. The AdaptPD tool performs these tasks in an integrated fashion by continuously monitoring the workload at the granularity of a query; DBAs often monitor at the granularity of thousands of queries. It uses cost-benefit analysis to decide if the current physical design is no longer efficient and a change is required. The tool consists of three components : the core algorithm behind adaptive physical design (Section 4), a cost estimator (Section 5), and a configuration manager (Section 6).

The core algorithm solves an online problem in which the objective is to adaptively transition between different database *configurations* in order to minimize the total costs in processing a given query sequence. Given a data model, let $D = \{o_1, \ldots, o_n\}$ be the set of all possible *physical design structures* that can be constructed, which includes vertical partitions of tables, materialized views, and indices[1]. A database instance is a combination of physical design structures subject to a storage size constraint T and is termed as a *configuration*. Let $\mathcal{S} = \{S_1, \ldots, S_N\}$ be the set of all possible configurations on D. The cost of processing a query q in a configuration S_i is denoted $q(S_i)$ (if q cannot be processed in S_i we set $q(S_i) = \infty$). Often it is necessary to change configurations to reduce query processing costs. The cost for *transitioning* between any two given configurations is given by the function $d : \mathcal{S} \times \mathcal{S} \rightarrow \Re^+$. d is any function that satisfies the following properties:

1. $d(S_i, S_j) \geq 0, \forall i \neq j, S_i, S_j \in \mathcal{S}$ (positivity);
2. $d(S_i, S_i) = 0, \forall i \in \mathcal{S}$ (reflexivity); and
3. $d(S_i, S_j) + d(S_j, S_k) \geq d(S_i, S_k), \forall S_i, S_j, S_k \in \mathcal{S}$ (triangle inequality)

In particular, d does not satisfy the symmetry property, *i.e.*,$\exists S_i, S_j \in \mathcal{S} d(S_i, S_j) \neq d(S_j, S_i)$. Asymmetry exists because the sequence of operations (i.e. insertion or deletion) required for making physical design changes exhibit different costs.

[1] In [11], physical design structures are referred to as *access paths*.

Given $\sigma = q_1, \ldots, q_n$, a finite sequence of queries, the objective in AdaptPD is to obtain a sequence of configurations $S = (S_0, S_1, ..., S_n), S_i \in \mathcal{S}$ such that the total cost of σ under S is minimized. The total cost is defined as

$$cost(\sigma, S) = \sum_{i=1}^{n} q_i(S_i) + \sum_{i=0}^{n-1} d(S_i, S_{i+1}), \tag{1}$$

in which the first term is the sum of costs of each query in σ under the corresponding configuration and the second term is the total cost to transition between configurations in S. Note, if $S_{i+1} = S_i$ there is no real change in the configuration schedule. An offline optimal algorithm OPT knows the entire σ and obtains a configuration schedule S with the minimum cost. An online algorithm ALG for AdaptPD determines $S = (S_0, ..., S_n)$ without seeing the complete workload $\sigma = (q_1, ..., q_n)$. Thus, ALG determines each configuration, S_i, based on the workload $(q_1, ..., q_i)$ seen so far.

In this paper we focus on configurations that arise from different *vertical partitions* in the data model [1]. Let $R = \{R_1, \ldots, R_k\}$ be the given set of relations in the data model. Each configuration $S \in \mathcal{S}$ now consists of a set of *fragments* $F = \{F_1, \ldots, F_N\}$ that satisfies the following two conditions: (1) every fragment F_i consists of an identifier column and a subset of attributes of a relation $R_j \in R$; and (2) each attribute of every relation R_j is contained in exactly one fragment $F_i \in F$, except for the primary key.

4 Algorithms in AdaptPD

In this section we describe two online algorithms for the AdaptPD tool: OnlinePD and HeuPD. OnlinePD provides a minimum level of performance guarantee and makes no assumptions about the incoming workload. HeuPD is greedy and adapts quickly to changes in the incoming workload.

4.1 OnlinePD

We present OnlinePD, which achieves a minimum level of performance for any workload. In particular, we show its cost is always at most $8(N-1)\rho$ times that of the optimal algorithm, where N is the total number of configurations in the set $\mathcal{S}$ and ρ is the *asymmetry constant* of $\mathcal{S}$. Further, to achieve this performance, OnlinePD does not need to be trained with a representative workload. OnlinePD is an amalgamation of algorithms for two online sub-problems: (1) the on-line ski rental problem and (2) the online physical design problem in which the cost function $d(\cdot)$ is symmetrical. We first describe the sub-problems.

Related Problems. Online ski rental is a classical rent-or-buy problem. A skier, who does not own skis, needs to decide before every skiing trip that she makes whether she should rent skis for the trip or buy them. If she decides to buy skis, she will not have to rent for this or any future trips. Unfortunately, she does not know how many ski trips she will make in future, if any. This lack of

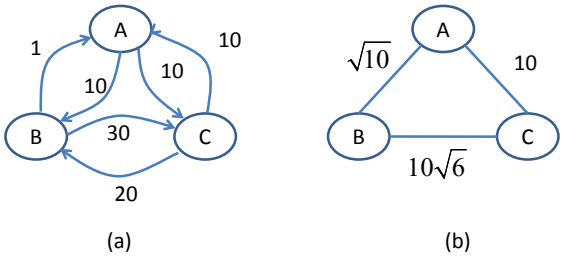


Fig. 1. Example of conversion from asymmetric transition costs to symmetric costs

knowledge about the future is a defining characteristic of on-line problems [17]. A well known on-line algorithm for this problem is rent skis as long as the total paid in rental costs does not match or exceed the purchase cost. Irrespective of the number of future trips, the cost incurred by this online algorithm is at most twice that of the optimal offline algorithm.

If there were only two configurations and the cost function $d(\cdot)$ satisfies symmetry, the OnlinePD problem will be nearly identical to online ski rental. Staying in the current configuration corresponds to renting skis and transitioning to another configuration corresponds to buying skis. Since the algorithm can start in any state, this leads to an algorithm that cost no more than four times the optimal.

In larger number of configurations, the key issue in establishing a correspondence with the online ski rental problem is in deciding which configuration to compare with the current one. When the costs are symmetrical, Borodin et. al [12] use *components* instead of configurations to perform an online ski rental. In particular, their algorithm recursively traverses one component until the query execution cost incurred in that component is approximately that of moving to the other component. A decision is then made to move to the other component (traversing it recursively) before returning to the first component and so on. To identify the components, they consider a complete, undirected graph $G(V, E)$ on $\mathcal{S}$ in which V represents the set of all configurations, E represents the transitions, and the edge weights are the transition costs. By fixing a minimum spanning tree (MST) on G, components are recursively determined by pick the maximum weight edge in the MST and removing it. This partitions all the configurations into two smaller components and the MST into two smaller trees.

This algorithm is shown to be $8(N-1)$-competitive [12]. ALG is α-competitive if there exists a constant b such that for every finite query sequence σ,

$$cost(\texttt{ALG}\ on\ \sigma) \leq \alpha * cost(\texttt{OPT}\ on\ \sigma) + b. \tag{2}$$

OPT is the offline optimal that has complete knowledge of σ. OnlinePD extends the above algorithm to solve the problem in which costs are asymmetrical. It does so by *transforming* its complete, directed graph on $\mathcal{S}$ and $d(\cdot)$ into a complete, undirected graph and applying any algorithm for online physical design in which costs are symmetrical. We describe the transformation and use Borodin's algorithm to show that it increases cost at most $8(N-1)\rho$ times of OPT.

Transformation in OnlinePD. When there are only two configurations, a simple transformation in which graph edges are replaced by the *sum* of transition costs gives a 3-competitive algorithm [10]. However, adding transition costs provides poor bounds for an N node graph. To achieve better competitive performance, we transform the directed graph into a undirected graph as follows:

Let G' be the directed graph. In G', replace every pair of directed edges (u, v) and (v, u) with an undirected edge (u, v) and a corresponding transition cost equal to $\sqrt{d(u,v).d(v,u)}$ irrespective of the direction. This transforms G' into H. H has the following two properties because of the transformation: a) If p is a path in H and p' is the corresponding path in G' (in any one direction), then $\frac{cost(p)}{\sqrt{\rho}} \leq cost(p') \leq \sqrt{\rho} cost(p)$. The inequality allows us to bound the error introduced by using H instead of G'. b) H violates the triangle inequality constraint. This is shown by a simple three-node example in Figure 1(a). In this example, a three node directed, fully connected graph with $\rho = 10$ is transformed to an undirected graph in Figure 1(b). The resulting triangle does not obey triangle inequality. OnlinePD exploits the fact that Borodin's algorithm constructs an MST, which makes it resilient to the triangle inequality violation.

Algorithm 1 details OnlinePD in which Algorithm 2 is a subroutine. To construct the traversal before processing queries, the MST is built on a graph in which edge weights are rounded to the next highest power of two. Let the maximum rounded weight in the MST, denoted by F in the Algorithm 1, be 2^M. We establish the proof using F.

Input: Directed Graph: $G(V, E_o)$ with weights $d(\cdot)$, Query Sequence: σ
Output: Vertex Sequence to process σ: $u_0, u_1, \ldots$
Transform G to undirected graph $H(V, E)$ s.t. $\forall (u, v) \in E$ weight
$d_H(u, v) \leftarrow \sqrt{d(u, v) \cdot d(v, u)}$;
Let $B(V, E)$ be the graph H modified s.t. $\forall (u, v) \in E$ weight
$d_B(u, v) \leftarrow d_H(u, v)$ rounded to next highest power of 2;
Let F be a minimum spanning tree on B;
$\mathcal{T} \leftarrow \mathsf{traversal}(F); u \leftarrow S_0$;
while *there is a query q to process* **do**
  $c \leftarrow q(u)$;
  Let v be the node after u in $\mathcal{T}$;
  while $c \geq d_B(u, v)$ **do**
    $c \leftarrow c - d_B(u, v); u \leftarrow v$;
    $v \leftarrow$ the node after v in $\mathcal{T}$;
  end
  Process q in u;
end

Algorithm 1: OnlinePD(G)

Lemma 1. *Any edge in $\mathcal{T}$ of rounded weight 2^m is traversed exactly 2^{M-m} times in each direction.*

Proof. We prove by induction on the number of edges in F. For the base case, there are no edges in F, and the lemma is trivially true. For the inductive case,

Input: Tree: $F(V, E)$
Output: Traversal for F: $\mathcal{T}$
if $E = \{\}$ **then**
  $\mathcal{T} \leftarrow \{\}$;
else if $E = \{(u, v)\}$ **then**
  Return $\mathcal{T}$: Start at u, traverse to v, traverse back to u;
else
  Let (u, v) be a maximum weight edge in E, with weight 2^M;
  On removing (u, v) let the resulting trees be $F_1(V_1, E_1)$ and $F_2(V_2, E_2)$, where $u \in V_1$, and $v \in V_2$;
  Let maximum weight edges in E_1 and E_2 have weights 2^{M_1} and 2^{M_2} respectively; $\mathcal{T}_1 \leftarrow \mathsf{traversal}(F_1)$;
  $\mathcal{T}_2 \leftarrow \mathsf{traversal}(F_2)$;
  Return $\mathcal{T}$: Start at u, follow $\mathcal{T}_1$ 2^{M-M_1} times, traverse (u, v), follow $\mathcal{T}_2$ 2^{M-M_2} times;
end

Algorithm 2: $\mathsf{traversal}(F)$

let (u, v) be the maximum weight edge in F used in $\mathsf{traversal}(\cdot)$, and similarly let F_1 and F_2 be the trees obtained by removing (u, v). Now the edge (u, v) is traversed exactly once in each direction as required by the lemma. By the inductive hypothesis, each edge of F_1 of rounded weight 2^m is traversed exactly 2^{M_1-m} times in each direction in the traversal $\mathcal{T}_1$, in which M_1 is the maximum rounded weight in F_1. Since $\mathcal{T}$ includes exactly 2^{M-M_1} traversals of $\mathcal{T}_1$, it follows that each such edge is traversed 2^{M-m} times in each direction in $\mathcal{T}$. The same reasoning applies to edges in F_2.

Theorem 1 *Algorithm* OnlinePD *is* $4(N-1)(\rho + \sqrt{\rho})$*-competitive for the* OnlinePD *problem with* N *configurations and asymmetry constant* ρ*.*

Proof. During each traversal of F, the following two statements are true: (i) the cost of OnlinePD is at most $2(N-1)2^M(1+\sqrt{\rho})$, and (ii) the cost of the offline optimal is at least $2^{M-1}/\sqrt{\rho}$. The theorem will then follow as the cost of OnlinePD during any single traversal is constant with respect to the length of σ. We prove (i) following Lemma 1 and (ii) from induction. (See proof in Appendix).

The bound of $8(N-1)\rho$ in OnlinePDis only a worst case bound. In our experiments, OnlinePD performs much better than best known offline algorithms for this problem and tracks closely with the workload adaptive algorithm HeuPD.

4.2 HeuPD

HeuPD chooses between neighboring configurations greedily. The current configuration in HeuPD ranks its neighboring configurations based on the estimated query execution costs in the neighboring configurations. HeuPD keeps track of the cumulative penalty of remaining in the current configuration relative to every other neighboring configuration for each incoming query. A transition is

made once HeuPD observes that the benefit of a new configuration exceeds a threshold. The threshold is defined as the sum of the costs of the most recent transition and next transition. HeuPD is described in detail in [10] and presented here as an alternative algorithm in AdaptPD. AdaptPD combines HeuPD with cost-estimation procedures described in this paper.

Let $x \in \mathcal{S}$ be the current configuration and $y \in \mathcal{S}$ be the neighboring configuration in which $y \neq x$. Define $\delta^y_{\max}(k)$ as the maximum cumulative penalty of remaining in x rather than transitioning to y at query q_k (the penalty of remaining in x for q_k is $q_k(x) - q_k(y)$). In HeuPD, this transition threshold is a function of the configuration immediately prior to x and the alternative configuration being considered. Let z be the configuration immediately prior to x in which the threshold required for transitioning to y is $d(z, x) + d(x, y)$. The decision to transition is greedy; that is, HeuPD transitions to the *first* configuration y that satisfies $\delta^y_{\max}(k) > d(z, x) + d(x, y)$.

5 Cost Estimation in AdaptPD

OnlinePD and HeuPD require $O(NlogN)$ time and space for pre-processing and $O(N)$ processing time per query. In this section we describe techniques to reduce N. Physical design tools also incur significant overhead in query cost estimation. Transition costs are often assigned arbitrarily, providing no correlation between the costs and actual time required to make transitions. Thus, we describe techniques for accurate and efficient cost estimation for vertical partitioning.

5.1 Transition Cost Estimation

We present an analytical transition model that estimates the cost of transitions between configurations. In an actual transition, data is first copied out of a database and then copied into new tables according to the specification of the new configuration. Gray and Heber [18] recently experimented with several data loading operations in which they observed that SQL bulk commands such as `BULK_INSERT` command in SQL Server work much like standard bulk copy (bcp) tools but are far more efficient than bcp as it runs inside the database. We base our analytical model on performance results obtained from using `BULK_INSERT` on a 300 column table.

We observe two artifacts of the `BULK_INSERT` operation. First, copying data into the database is far more expensive than copying data out of the database. Second, cost of importing the data scales linearly with the amount of data being copied into the database. The first artifact is because data is normally copied out in native format but is loaded into the database with type conversions and key constraints. The linear scaling is true because `BULK_INSERT` operations mostly incur sequential IO. We model the cost of importing a partition P:

$$BCP(P) = cR_PW_P + kR_P \tag{3}$$

in which R_P is the number of rows, W_P is the sum of column widths, c is the per byte cost of copying data into the database, and k is the per row cost

of constructing the primary key index. Thus, the estimated cost is the sum of importing data and the cost of creating a primary key index. Index creation cost is linear due to a constant overhead with each new insert. Constants c and k are system dependent and can be easily determined using regression on few "sample" `BULK_INSERT` operations. In this model, we assume no cost for transaction logging, which is disabled for fast copy.

The transition cost model uses Equation 3 to model the cost of moving from a configuration S_i to another configuration S_j. Let configuration S_i consists of partitions $\{T1_i,...,Tm_i\}$, S_j consists of partitions $\{T1_j,...,Tn_j\}$ and Δ_{ij} be the partition set difference $\{T1_j,...,Tn_j\}$ - $\{T1_i,...,Tm_i\}$, The transition cost is:

$$d(S_i, S_j) = \sum_{t \in \Delta_{ij}} BCP(t) \tag{4}$$

5.2 Query Cost Estimation

We present an efficient and yet accurate technique for estimating query costs across several configurations. The technique is based on the idea that cached query plans can be reused for query cost estimation. The traditional approach of asking the optimizer for the cost of each query on each configuration is well-known to be very expensive [15]. By caching and reusing query plans, the technique avoids invoking the optimizer for cost estimation and achieves an order of magnitude improvement in efficiency. To maintain high accuracy, the technique relies on recent observations that the plan of a query across several configurations is an invariant. By correctly determining the right plan to reuse and estimating its cost, the technique achieves the complementary goals of accuracy and efficiency. We describe conditions under which plans remain invariant across configurations and therefore can be cached for reuse. We then describe methods to cache the plans efficiently and methods to estimate costs on cached plans.

Plan Invariance. We illustrate with an example when the plan remains invariant across configurations and when it does not. Figure 2 shows three different configurations S_1, S_2, S_3 on two tables `T1(a,b,c,d)` and `T2(e,f,g,h)` with primary and join keys as `a` and `e`, respectively. Consider a query q that has predicate clauses on c and d and a join clause on a and e: `select T1.b, T2.f from T1, T2 where T1.a = T2.e and T1.c > 10 and T1.d = 0`. Let the query be optimized in S_1 with the shown join methods and join orders. The same plan is optimal in S_2 and can be reused. This is because S_2's partitions with respect to columns c and d are identical to S_1's partitions. In S_3, however, the plan cannot be used as columns c and d are now merged into a single partition. This is also reflected by the optimizer's choice which actually comes with a different plan involving different join methods and join orders.

Plan invariance can be guaranteed if the optimizer chooses to construct the same plan across different configurations.

Theorem 1. *The optimizer constructs the same query plan across two configurations S_1 and S_2, if the following three conditions are met:*

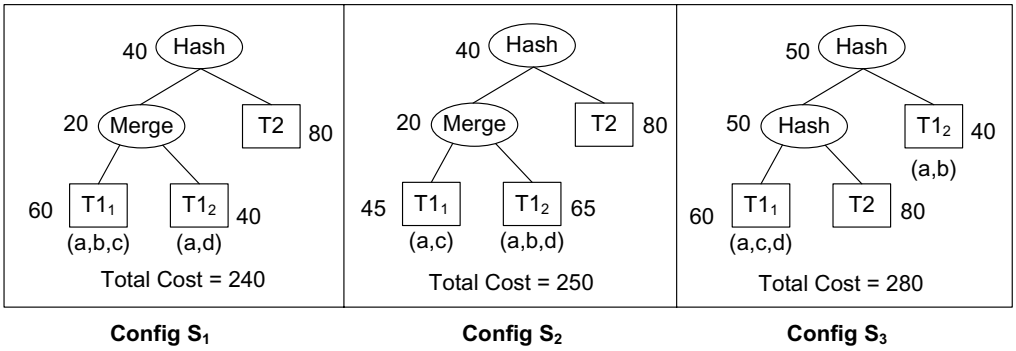


Fig. 2. Plan reuse across configurations. (Estimated costs are for illustration only).

1. *Configurations have the same number of partitions with respect to all columns mentioned in the query.*
2. *The division of the predicate columns in S_1 and S_2 is exactly the same.*
3. *If $\delta(S_1)$ and $\delta(S_2)$ define the page size distributions of S_1 and S_2 respectively, then $dist(\delta(S_1), \delta(S_2)) < \epsilon$. Where dist function determines the distance between two page distributions, and ϵ is a DBMS dependent constant.*

Condition 1 guarantees the same number of joins in the plan for any two configurations. For instance, if S_2 partitions `T1` into three partitions, then the optimizer joins twice instead of once to reconstruct the rows of the original table. Since the resulting plan is different from S_1, plan reuse cannot be guaranteed. If the query does not select on b, then the same plan can still be reused.

Condition 2 guarantees similar cardinality of the intermediate join results so that the optimizer selects the same join order and method to find the optimal plan. Condition 2 is illustrated in Figure 2 in which keeping c and d in different partitions leads to a merge joins in S_1 and S_2. A hash join is preferred in S_3 when c and d are grouped together.

Condition 3 avoids comparing drastically different configurations in terms of page distribution. The *dist* function can be a standard distribution distance, such as KL-divergence [19], and ϵ can be determined by experimenting over large number of plans. That is if a large table, with say 100 columns, has two configurations and if in the first configuration partitions are of uniform sizes, (*i.e.* two partitions with each partition containing 50 columns) and in the second configuration partitions are highly skewed (*i.e.* one partition has one column and the second has all the remaining columns), then the optimizer does not construct the same plan. In particular, the optimizer prefers to join equi-sized partition tables earlier and delays joining skewed tables as long as possible. Hence reusing the plan of one configuration for the other provides inaccurate results.

If above three conditions are satisfied, we prove by contradiction that the optimizer generates the same plan for any given two configurations. Suppose the join method and join order for S_i is J_1 and for S_j is J_2. By our assumption, J_1 and J_2 are different. Without loss of generality, let J_1 costs less than J_2 if we ignore the costs of scanning partitions. Since the configurations S_i and S_j have the same orders (primary key orders for all partitions), select the same number

of rows from the partitions, and the page size of the filtered rows are similar, J_1 can still be used for S_j. Since using J_1 reduces the total cost for running the query on S_j, it implies that J_2 is not the optimal plan, which contradicts our assumption that J_2 is part of the optimal plan.

The Plan Cache. We cache the query plan tree with its corresponding join methods, join-orders, and partition scans. The stripped plan tree is uniquely identified by $<query\_id, partition\_list, page\_distribution>$. The first part of the string identifies the query for which the plan is cached, the second part specifies the list of partitions in which columns in the predicate clause of the query occur, and the third part specifies the page distribution of each partition.

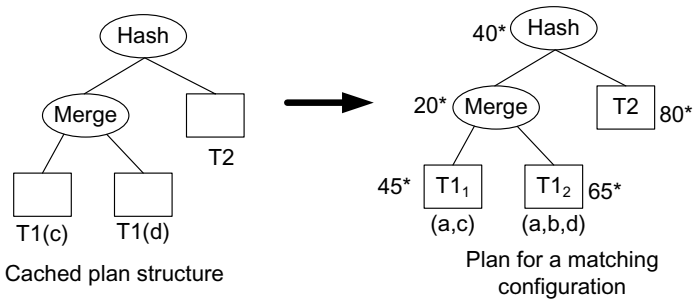


Fig. 3. The plan for S_1 cached with the key $[query - id, T1((c),(d)), T1(128,64)]$ on the left. On the right is the estimated plan for the new configuration S_2.

Cost Estimation. Cost estimation involves accurate estimation of partition scan costs and join costs. Thus given a new configuration, we first retrieve its corresponding plan from the cache using the key and then estimate the costs of partition scans and join methods. Partition scan costs are estimated by computing the average cost of scanning the partition in the cached plan and multiplying it with actual size of partition in the new configuration. Thus if c is the I/O cost of the scanning operation in the cached plan, and s_0 is the size of the vertical partition in the cached plan, the cost of partition scan in the new configuration is estimated as $f = c \times \frac{s}{s_0}$. In this s is the size of the new partition. To estimate the cost for joining partitions using the join methods from the cache, we adopt the System-R's cost model, developed by Selinger et al. [20]. The System-R cost model gives us an upper bound on the actual join costs, and according to our experiments predicts the plan cost with 98.7% accuracy on average.

6 Experiments

We implement our online partitioning algorithms and cost estimation techniques in the SDSS [4] Astronomy database. We describe the experimental setup before presenting our main results. This includes analysis of workload evolution over time, performance of various online and offline algorithms, and accuracy of cost estimation.

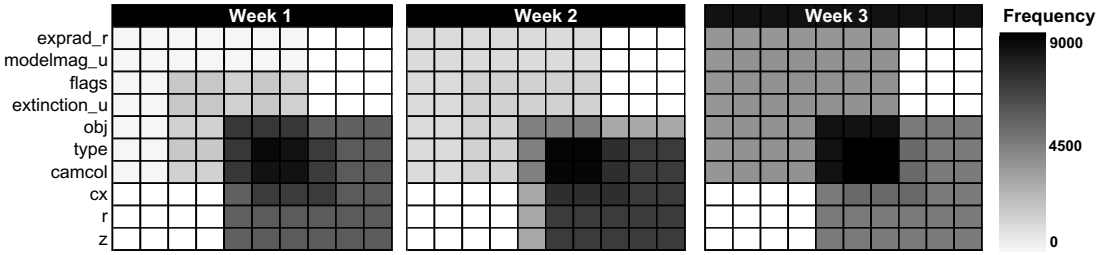


Fig. 4. Affinity matrix (co-access frequency) for ten select attributes from the *PhotoObjAll* table

6.1 Experimental Setup

Workload Characteristics. We use a month-long trace from SDSS consisting of 1.4 million read-only queries. The queries consist of both simple queries on single tables and complex queries joining multiple tables. Also, queries are template-based [5] and can be summarized compactly using templates. However, considerable *evolution* occurs in the workload in that new templates are introduced continually and prior templates may disappear entirely.

Figure 4 captures workload evolution for the first three weeks of the trace. It shows the affinity matrix for ten attributes from a single table in which each grid entry corresponds to the frequency with which a pair of attributes are accessed together (ordering of attributes are the same along the row and column). The basic premise is that columns that occur together and have similar frequencies should be grouped together in the same relation [21]. The results show that column groupings change on a weekly basis. An online physical design tool which continuously monitors the workload can evaluate whether transitioning to a new configuration will lead to a improvement in overall cost.

Comparison Methods. We contrast the performance of OnlinePD with several online and offline algorithms. OnlinePD has polynomial-time complexity and finds the minimal spanning tree using the Prim's algorithm. While it makes no assumptions about the workload, this generality comes at a cost. Namely, given domain specific knowledge about the workload, highly tuned workload adaptive algorithms can be designed. To measure the cost of generality, we compare OnlinePD with HeuPD (Section 4.2). We also compare against `AutoPart`, an existing, offline vertical partitioning algorithm. `AutoPart` is presented with the entire workload as input during initialization. This incurs an initial overhead to produce a physical design layout for the workload, but it can service the workload with no further tuning. `AutoPartPD` is another physical design strategy that employs the offline `AutoPart` algorithm. Unlike `AutoPart`, it is adaptive by running `AutoPart` daily (incurs one transition at the beginning of each day) and it is prescient in that the workload for each day is provided as input *a priori*. Finally, `NoPart` serves as the base case in which no vertical partitioning is used.

Costs. The transition costs are estimated using the analytical model each time a new template is introduced. The estimates show that the asymmetry constant

(Section 4) ρ is bounded and its maximum value is approximately 3.25. The model itself estimates the transition costs with 87% accuracy in which transition costs are considered accurate if they are within 10% of the actual costs. Query cost estimation is done using the cache and reuse technique, which provides 94% accuracy in our experiments.

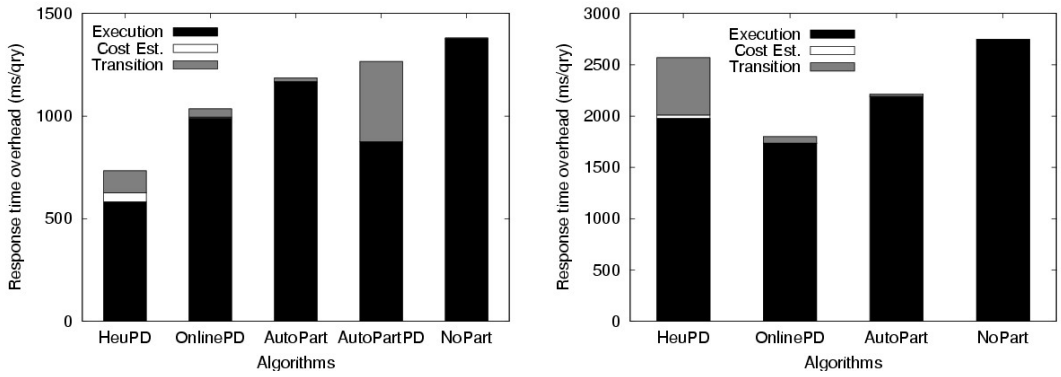
Database. For I/O experiments, we execute queries against a five percent sample (roughly 100GB in size) of the DR4 database. Although sampling the database is less than ideal, it is necessary to finish I/O experiments in a reasonable time for real workloads. Given the time constraints, we compromised database size in order to accommodate a larger workload, which captures workload evolution over a longer period. To sample the database, we first sample the fact table consisting of all celestial objects (*PhotoObjAll*) and then sample the remaining tables through foreign key constraints.

The data is stored in Microsoft's SQL Server 2000 on a 3GHz Pentium IV workstation with 2GB of main memory and two SATA disks (a separate disk is assigned for logging to ensure sequential I/O). Microsoft SQL Server does not allow for queries that join on more than 255 physical tables. This is required in extreme cases in which the algorithm partitions each column in a logical relation into separate tables. Hammer and Namir [22] show that between the two configurations with each column stored separately or all columns stored together, the preferred configuration is always the latter. In practice, this configuration does not arise because the cost of joining across 255 tables is so prohibitive that our algorithm never selects this configuration. To reduce the configuration space, we do not partition tables that are less than 5% of the database size. This leads to a large reduction in the number of configurations with negligible impact on performance. The total number of configurations is around 5000.

Performance Criteria. We measure the cost of algorithms in terms of average query response time. This is the measure from the time a query is submitted until the results are returned. If a transition to a new configuration is necessary, the algorithm undergoes a transition before executing the query. This increases the response time of the current query but amortizes the benefit over future queries. Our results reflect average response time over the entire workload.

6.2 Results

We compute the query performance by measuring its response time on the proxy cache using the sampled database. Figure 5(a) provides the division of response time for query execution, cost estimation using the optimizer, and transitions between configurations. (The total response time is averaged over all queries). OnlinePD improves on the performance of `NoPart` by a factor of 1.5 with an average query execution time of 991 ms. Not surprisingly, HeuPD, which is tuned specifically for SDSS workloads, further improves performance by 40% and exhibits two times speedup over `NoPart`. This improvement is low considering that OnlinePD is general and makes no assumptions regarding workload access patterns. `NoPart` suffers due to higher scan costs associated with reading extraneous



(a) SDSS Workload (b) Adversarial Workload

Fig. 5. Distribution of response time overhead

columns from disk. Likewise, `AutoPart` suffers by treating the entire workload as an unordered set of queries and providing a single, static configuration during initialization. Even `AutoPartPD` did not improve response time beyond the offline solution because the benefits of periodic physical design tuning is offset by high, daily transition costs. Thus, adapting offline solutions such as `AutoPart` to evolving workloads is challenging because they do not continuously monitor for workload changes nor account for transition cost in tuning decisions.

Another interesting feature of the results is that OnlinePD incurs much lower transition costs than HeuPD. This artifact is due to the conservative nature of OnlinePD. It evaluates only two alternatives at a time and transitions only if it expects significant performance advantages. On the other hand, HeuPD responds quicker to workload changes by evaluating all candidate configurations simultaneously and choosing a configuration that benefits the most recent sequence of queries. This optimism of HeuPD is tolerable in this workload but can account for significant transition costs in workloads that change more rapidly relative to SDSS. To appreciate the generality of OnlinePD over a heuristic solution, we evaluated a synthetic SDSS workload that is adversarial with respect to HeuPD in Figure 5(b). In particular, the workload is volatile and exhibits no stable state in the access pattern, which causes HeuPD to make frequent, non-beneficial transitions. As a result, Figure 5(b) shows that OnlinePD exhibits a lower query execution time and a factor of 1.4 improvement over HeuPD.

Figure 5(a) also shows the average response time of performing cost estimation (time spent querying the optimizer). For `AutoPart`, this is a one-time cost incurred during initialization. In contrast, cost estimation is an incremental overhead in OnlinePD and HeuPD. HeuPD incurs a ten folds overhead in cost estimation over OnlinePD (43 ms versus 4 ms). This is because HeuPD incurs 93 calls to the optimizer per query. Thus, HeuPD benefits immensely from QCE due to the large number of configurations that it evaluates for each query. Reusing cached query plans allow HeuPD to reduce cost estimation overhead by ten folds and avoid 91% of calls to the optimizer. Without QCE, the total average response time of HeuPD is 1150 ms, which would lag the response time of OnlinePD by 4 ms. As such, HeuPD scales poorly as the number of alternative

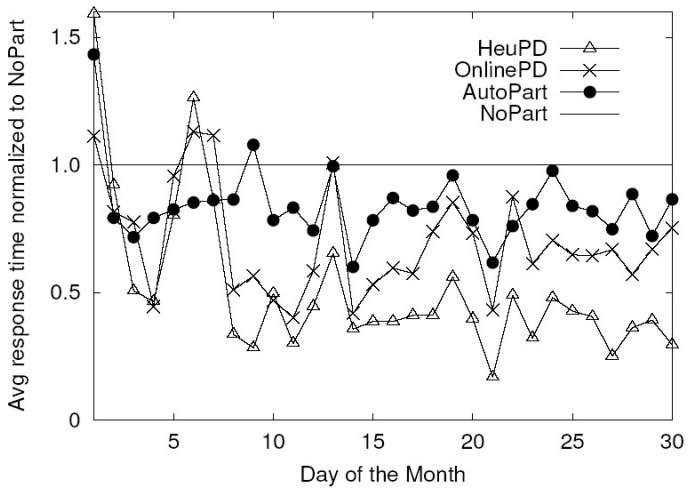


Fig. 6. Average daily response time overhead normalized to NoPart

configurations increases. This make OnlinePD attractive for proxy caches which receive a continuous stream of queries and decisions have to be made rapidly.

Finally, we refer to the average transition cost (cost of changing between configurations) from Figure 5(a). `AutoPart` only incurs a single transition during initialization while `NoPart` incurs no transition cost. `AutoPartPD` incurs the highest overhead, requiring a complete reorganization of the database on a daily basis. HeuPD makes 768 minor configuration changes compared with 92 for OnlinePD which leads to a three times per query overhead in transition cost (113 ms compared with 43 ms). Thus, while OnlinePD is slower than HeuPD at detecting and adapting to changes in the workload, it benefits with fewer transitions that disrupt the execution of incoming queries.

Figure 6 charts the average daily response time (both query execution and transition cost) for various algorithms normalized to `NoPart`. There is significant fluctuations in average response times resulting from workload changes over time. While all algorithms improve on `NoPart`, `AutoPart` tracks most closely with `NoPart` since neither implements changes to the physical design after initialization. OnlinePD and HeuPD further improves response time, but exhibits several performance spikes (most notably on days one, six, and thirteen) that perform no better than `NoPart`. These indicate significant workload changes that cause more transitions to occur that delay completion of certain queries. The transition overhead is greatest for OnlinePD and HeuPD on day one and remains more stable afterward because at initialization, all tables are unpartitioned.

Figure 7 shows the cumulative distribution function (CDF) of the error in cost estimation using QCE instead of the optimizer. This error is determined by:

$$abs\left(1 - \left(\frac{\texttt{QCE est. query cost}}{\texttt{Optimizer est. query cost}}\right)\right) \tag{5}$$

Consider the dashed-line in the plot, which corresponds to the errors in cost estimation for all queries. Although the average cost estimation error is only 1.3%, the plot shows that the maximum error in cost estimation is about 46%,

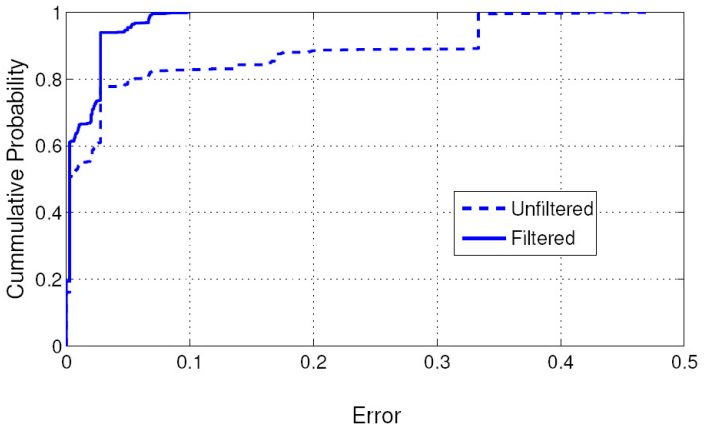


Fig. 7. Error in cost estimation for QCE. Dashed line represents all queries and solid line represents queries with higher than 5 unit cost.

with about 14% of the estimations with more than 10% error. Inspecting high error estimations reveals that the errors occur in queries with estimated costs below 5 optimizer cost units. We plot the CDF for errors after removing those light queries. The solid line in Figure 7 shows the cost estimation error for these filtered set of queries. The maximum error for the filtered queries is about 11%, and about 94% of the estimations have less than 5% error.

The inaccuracies in the light queries comes from the approximations discussed in Section 5.2. Since the contribution of light queries to workload cost is insignificant compared to the heavy queries (4% of our workload), the inaccuracy in estimating their costs does not affect the configurations selected by our algorithm.

7 Summary and Future Work

In this paper, we have presented AdaptPD, a workload adaptive physical design tool that automates some of the DBA tasks such as estimating when to tune the current physical design and finding representative workloads to feed the physical design tool. The tool quantitatively compares the current configuration with other possible configurations, giving the DBA a good justification of the usefulness of the recommended design. Automation of such tasks reduces the cost of ownership of large database systems such as the SDSS in which physical design tuning is routinely performed by DBAs. Since these tools gradually change ownership from DBAs to curators, it is essential to minimize the overhead of administration and yet ensure good performance.

We have developed novel online techniques that adapt to drastic changes in the workload without sacrificing the generality of the solution. The techniques are supported by efficient cost estimation modules that make them practical for continuous evaluation. Experimental results for the online algorithm show significant performance improvement over existing offline methods and tracks closely with heuristic solution tuned specifically for SDSS workloads. These tuning tools are not specific to vertical partitions and can be extended to index design, which is our primary focus going forward.

References

1. Papadomanolakis, S., Ailamaki, A.: AutoPart: Automating Schema Design for Large Scientific Databases Using Data Partitioning. In: SSDBM (2004)
2. Agrawal, S., Narasayya, V.R., Yang, B.: Integrating Vertical and Horizontal Partitioning Into Automated Physical Database Design. In: SIGMOD (2004)
3. Chu, W.W., Ieong, I.T.: A Transaction-Based Approach to Vertical Partitioning for Relational Database Systems. IEEE Trans. Software Eng. 19(8), 804–812 (1993)
4. The Sloan Digital Sky Survey, `http://www.sdss.org`
5. Wang, X., Malik, T., Burns, R., Papadomanolakis, S., Ailamaki, A.: A Workload-Driven Unit of Cache Replacement for Mid-Tier Database Caching. In: Kotagiri, R., Radha Krishna, P., Mohania, M., Nantajeewarawat, E. (eds.) DASFAA 2007. LNCS, vol. 4443, pp. 374–385. Springer, Heidelberg (2007)
6. Sattler, K.U., Geist, I., Schallehn, E.: QUIET: Continuous Query-Driven Index Tuning. In: VLDB (2003)
7. Schnaitter, K., Abiteboul, S., Milo, T., Polyzotis, N.: COLT: Continuous On-line Tuning. In: SIGMOD (2006)
8. Bruno, N., Chaudhuri, S.: An Online Approach to Physical Design Tuning. In: ICDE (2007)
9. Stonebraker, M., Abadi, D., Batkin, A., Chen, X., Cherniack, M., Ferreira, M., Lau, E., Lin, A., Madden, S., O'Neil, E., O'Neil, P., Rasin, A., Tran, N., Zdonik, S.: C-Store: A Column Oriented DBMS. In: VLDB (2005)
10. Malik, T., Wang, X., Burns, R., Dash, D., Ailamaki, A.: Automated Physical Design in Database Caches. In: SMDB (2008)
11. Agrawal, S., Chu, E., Narasayya, V.: Automatic Physical Design Tuning: Workload as a Sequence. In: SIGMOD (2006)
12. Borodin, A., Linial, N., Saks, M.E.: An Optimal Online Algorithm for Metrical Task System. J. ACM 39(4), 745–763 (1992)
13. Manasse, M.S., McGeoch, L.A., Sleator, D.D.: Competitive algorithms for server problems. J. Algorithms 11(2), 208–230 (1990)
14. Abadi, D., Madden, S., Hachem, N.: Column-Stores Vs. Row-Stores: How Different Are They Really. In: SIGMOD (2008)
15. Papadomanolakis, S., Dash, D., Ailamaki, A.: Efficient Use of the Query Optimizer for Automated Database Design. In: VLDB (2007)
16. Bruno, N., Nehme, R.: Configuration Parametric Query Optimization for Physical Design Tuning. In: SIGMOD (2008)
17. Borodin, A., El-Yaniv, R.: Online computation and competitive analysis. Cambridge University Press, New York (1998)
18. Heber, G., Gray, J.: Supporting Finite Element Analysis with a Relational Database Backend Part II: Database Design and Access. Technical Report, Microsoft Research (2006), `http://research.microsoft.com/apps/pubs/default.aspx?id=64571`
19. Kullback, S., Leibler, R.A.: On information and sufficiency. Ann. Math. Statistics (1951)
20. Selinger, P., Astrahan, M., Chamberlin, D., Lorie, R., Price, T.: Access Path Selection in a Relational Database Management System. In: SIGMOD (1979)
21. Navathe, S., Ceri, S., Wiederhold, G., Dou, J.: Vertical Partitioning Algorithms for Database Design. ACM Trans. Database Syst. 9(4), 680–710 (1984)
22. Hammer, M., Niamir, B.: A Heuristic Approach to Attribute Partitioning. In: SIGMOD (1979)

Appendix: Competitiveness of OnlinePD

We prove that OnlinePD algorithm is $O(4\rho(N-1))$ competitive. Reusing the notations from Section 4, in Algorithm 1 we construct the graph B by rounding up the cost of edges on the undirected transition graph to a power of two. We build an MST on B and call it F. We then build a traversal on F using Algorithm 2 and denote that traversal as $\mathcal{T}$. Let the maximum rounded weight in the tree F be 2^M. The following proof is inspired by the proof in [12]

Lemma 2. *If the maximum edge weight in $\mathcal{T}$ is 2^M, any edge in $\mathcal{T}$ of rounded weight 2^m is traversed exactly 2^{M-m} times in each direction.*

Proof. We prove by induction on the number of edges in F. For the base case, there are no edges in F, and the lemma is trivially true. For the inductive case, let (u, v) be the maximum weight edge in F used in the traversal$(\cdot)$, and similarly let F_1 and F_2 be the trees obtained by removing (u, v). Now the edge (u, v) is traversed exactly once in each direction as required by the lemma. By the inductive hypothesis, each edge of F_1 of rounded weight 2^m is traversed exactly 2^{M_1-m} times in each direction in the traversal $\mathcal{T}_1$, in which M_1 is the maximum rounded weight in F_1. Since $\mathcal{T}$ includes exactly 2^{M-M_1} traversals of $\mathcal{T}_1$, it follows that each such edge is traversed 2^{M-m} times in each direction in $\mathcal{T}$. Exactly the same reasoning applies for edges in F_2.

Theorem 2 *Algorithm* OPDA *is $4(N-1)(\rho+\sqrt{\rho})$-competitive for the* OnlinePD *problem with N configurations and asymmetry constant ρ.*

Proof. We shall prove that during each traversal of F: (i) the cost of OPDA is at most $2(N-1)2^M(1+\sqrt{\rho})$, and (ii) the cost of the offline optimal is at least $2^{M-1}/\sqrt{\rho}$. The theorem will then follow as the cost of OPDA during any single traversal is constant with respect to the length of σ.

To prove (i), recall from Lemma 2 that any edge in $\mathcal{T}$ of rounded weight 2^m is traversed exactly 2^{M-m} times. Thus the total rounded weight traversed for an edge is $2 \cdot 2^{M-m} \cdot 2^m = 2 \cdot 2^M$. By construction of the algorithm the total processing cost incurred during $\mathcal{T}$ at a node just before a traversal of this edge is $2 \cdot 2^M$. The total transition cost incurred during $\mathcal{T}$ in a traversal of this edge is at most $2 \cdot 2^M\sqrt{\rho}$, since the cost $d(\cdot)$ can be at most $\sqrt{\rho}$ times larger than the corresponding $d_B(\cdot)$. This proves (i) as there are exactly $N-1$ such edges.

We prove (ii) by induction on the number of edges in F. Suppose F has at least one edge, and (u, v), F_1, and F_2 are as defined in traversal$(\cdot)$. If during a cycle of $\mathcal{T}$, OPT moves from a vertex in F_1 to a vertex in F_2, then since F is a minimum spanning tree, there is no path connecting F_1 to F_2 with a total weight smaller than $d_B(u, v)/(2\sqrt{\rho}) = 2^{M-1}/\sqrt{\rho}$. Otherwise during the cycle of $\mathcal{T}$, OPT only stays in one of F_1 or F_2; w.l.o.g. assume F_1. If F_1 consists of just one node u, and OPT stays there throughout the cycle of $\mathcal{T}$, then by definition of the algorithm, OPT incurs a cost of at least $d_B(u, v) = 2^M \geq 2^{M-1}/\sqrt{\rho}$. If F_1 consists of more than one node, then by the induction hypothesis, OPT incurs a cost of at least $2^{M_1-1}/\sqrt{\rho}$ per cycle of T_1. Since during one cycle of $\mathcal{T}$ there are 2^{M-M_1} cycles of $\mathcal{T}_1$, OPT incurs a cost of at least $2^{M-1}/\sqrt{\rho}$. This completes the proof.