Vertical partitioning of relational OLTP databases using integer programming

Rasmus Resen Amossen

IT University of Copenhagen
Rued Langaardsvej 7
2300 Kbh. S, Denmark
rasmus@resen.org

Abstract— A way to optimize performance of relational row store databases is to reduce the row widths by vertically partitioning tables into table fractions in order to minimize the number of irrelevant columns/attributes read by each transaction. This paper considers vertical partitioning algorithms for relational row-store OLTP databases with an H-store-like architecture, meaning that we would like to maximize the number of single-sited transactions. We present a model for the vertical partitioning problem that, given a schema together with a vertical partitioning and a workload, estimates the costs (bytes read/written by storage layer access methods and bytes transferred between sites) of evaluating the workload on the given partitioning. The cost model allows for arbitrarily prioritizing load balancing of sites vs. total cost minimization. We show that finding a minimum-cost vertical partitioning in this model is NP-hard and therefore the problem should obviously not be solved manually by a human DBA. We present two algorithms returning solutions in which single-sitedness of read queries is preserved while allowing column replication (which may allow a drastically reduced cost compared to disjoint partitioning). The first algorithm is a quadratic integer program that finds optimal minimum-cost solutions with respect to the model, and the second algorithm is a more scalable heuristic based on simulated annealing. Experiments show that the algorithms can reduce the cost of the model objective by 37% when applied to the TPC-C benchmark and the heuristic is shown to obtain solutions with costs close to the ones found using the quadratic program.

I. INTRODUCTION

In this paper we consider OLTP databases with an H-store [12] like architecture in which we would aim for maximizing the number of single-sited transactions (i.e. transactions that can be run to completion on a single site). The full version of the paper can be found in [3]. Given a database schema and a workload we would like to reduce the cost of evaluating the workload. In row-stores, where each row is stored as a contiguous segment and access is done in quantums of whole rows, a significant amount of superfluous columns/attributes (we will use the term *attribute* in the following) are likely to be accessed during evaluation of a workload. It is easy to see that this superfluous data access may have a negative impact on performance so in an optimal world the amount of data accessed by each query should be minimized. One approach to this is to perform a *vertical partitioning* of the tables in the schema. A vertical partitioning is a, possibly non-disjoint, distribution of attributes and transactions onto multiple physical or logical sites. (Notice, that vertical and

horizontal partitioning are not mutually exclusive and can perfectly be used together). The optimality of a vertical partitioning depends on the context: OLAP applications with lots of many-row aggregates will likely benefit from parallelizing the transactions on multiple sites and exchanging small sub-results between the sites after the aggregations. OLTP applications on the other hand, with many short-lived transactions, no many-row aggregates and with few or no few-row aggregates would likely benefit from gathering all attributes read by a query locally on the same site: inter-site transfers and the synchronization mechanisms needed for non-single-sited or parallel queries (e.g. undo and redo logs) are assumed to be bottlenecks in situations with short transaction durations. Stonebraker et al. [12] and Kallman et al. [8] discuss the benefits of single-sitedness in high-throughput OLTP databases in more details.

This paper presents a cost model together with two algorithms that find either optimal or close-to-optimal vertical partitionings with respect to the cost model. The two algorithms are based on quadratic programming and simulated annealing, respectively. For a given partitioning and a workload, the cost model estimates the number of bytes read/written by access methods in the storage layer and the amount of data transfer between sites. Our model is made with a specific setting in mind, captured by five headlines:

OLTP The database is a transaction processing system with many short lived transactions.

Aggregates No many-row aggregates and few (or no) aggregates on small row-subsets.

Preserve single-sitedness We should try to avoid breaking single-sitedness as a large number of single-sited transactions will reduce the need for inter-site transfers and completely eliminate the need for undo and redo logs for these queries if the partitioning is performed on an H-store like DMBS [12].

Workload known Transactions used in the workload together with some run-time statistics are assumed to be known when applying the algorithms.

Furthermore, following the consensus in the related work (see Section I-C) we simplify the model by not considering time spent on network latency (if all vertical partitions are placed locally on a single site, then time spend on network latency is trivially zero anyway). A description of how to include latency in the model at the expense of increased complexity can be

found in the appendix.

A. Outline of approach

The basic idea is as follows. We are given an input in form of a schema together with a workload in which queries are grouped into transactions, and each query is described by a set of statistical properties.

For each query q in the workload and for each table r accessed by q the input provides the average number n_r of rows from table r that is retrieved from or written to storage by query q. Together with the width w_a of each attribute a from table r this generally gives a good estimate for how much attribute a costs in retrievals/writes by access methods for each evaluation of query q, namely $W'_{a,q} = w_a \cdot n_r$. Notice, that w_a may be an average in cases with variable-length attributes.

Given a set of sites, the challenge is now to find a non-disjoint distribution of all attributes, and a disjoint distribution of transactions to these sites so that the costs of retrievals, writes and inter-site transfers, each defined in terms of $W'_{a,q}$ as explained in details below, is minimized. This means, that the primary executing site of any given query is assumed to be the site that hosts the transaction holding that query.

As mentioned above, our algorithms will not break single-sitedness for read queries and therefore no additional costs are added to the execution of read queries by applying this algorithm. In contrast, since the storage costs (the sum of retrieval, write and inter-site transfer costs) for a query is minimized and each tuple become as narrow as possible, the total costs of evaluating the queries (e.g. processing joins, handling intermediate subresults, etc.) are assumed to be, if not minimized, then reduced too.

B. Contributions

This work contributes with the following:

- an algorithm optimized for H-store like architectures, preserving single-sitedness for read queries and in which load balance among sites versus minimization of total costs can be prioritized arbitrarily,
- a more scalable heuristic, and
- a micro benchmark of a) both algorithms based on TPC-C and a set of random instances, b) a comparison between the benefits of local versus remote partition location, and c) a comparison between disjoint and non-disjoint partitioning.

C. Related work

A lot of work has been done on data allocation and vertical partitioning but to the best of our knowledge, no work solves the exact same problem as the present paper: distributing both transactions and attributes to a set of sites, allowing attribute replication, preserving single-sitedness for read queries and prioritizing load balancing vs. total cost minimization. We therefore order the references below by increasing estimated problem similarity and do not mention work dedicated on vertical partitioning of OLAP databases. The full version of this paper [3] describes some additional related work.

Cornell and Yu [5] generated a non-remote, disjoint partitioning minimizing the amount of disk access by recursively applying a binary partitioning. The partitioning decisions were based on an integer program and with strong assumptions on a System-R like architecture when estimating the amount of disk access.

Agrawal et al. [2] also constructed a disjoint partitioning with non-remote partition placement. They used a two-phase strategy where the first phase generated all relevant attribute groups using association rules [1] considering only one query at a time, and the second phase merged the attribute groups that were useful across queries.

Son and Kim [11] presented an algorithm for generating disjoint partitioning by either minimizing costs or by ensuring that exactly k vertical fragments were produced. Inter-site transfer costs were not considered. The partitioning was produced using a bottom-up strategy, iteratively merging two selected partitions with the best "merge profit" until only one large super-partition existed. The k-way partitioning was found at the iteration having exactly k partitions and the lowest-cost partitioning was found at the iteration with the lowest cost.

Chu and Ieong [4] minimized the amount of disk access by constructing a non-remote and non-disjoint vertical partitioning. Two binary partitioning algorithms based on the branch-and-bound method were presented with varying complexity and accuracy. The partitionings were formed by recursively applying the binary partitioning algorithms on the set of "reasonable cuts".

Navathe et al. [10] considered the vertical partitioning problem for three different environments: a) single site with one memory level, b) single site with several memory levels, and c) multiple sites. The partitions could be both disjoint and non-disjoint. A clustering algorithm grouped attributes with high affinity by using an attribute affinity matrix together with a bond energy algorithm [7]. Three basic algorithms for generating partitions were presented which, depending on the desired environment, used different prioritization of four access and transfer cost classes.

D. Outline of paper

In section II we derive a cost model together with a quadratic program defining the first algorithm. Section III describes a heuristic based on the cost model found in Section II, and Section IV discusses a couple of ideas for improvements. Computational results are shown in Section V.

II. A LINEARIZED QP APPROACH

In this section we develop our base model, a quadratic program (QP), which will later be extended to handle load balancing and then linearized in order to solve it using a conventional mixed integer program (MIP) solver.

A. The base model

In a vertical partitioning for a schema and a workload we would like to minimize the sum

$$A + pB \tag{1}$$

where A is the amount of data accessed locally in the storage layer, B is the amount of data needed to be transferred over the network during query updates and p is a penalty factor.

We assume that each transaction has a primary executing site. For each transaction $t \in \mathcal{T}$, each table attribute $a \in \mathcal{A}$, and each site $s \in \mathcal{S}$ consider two decision variables $x_{t,s} \in \{0,1\}$ and $y_{a,s} \in \{0,1\}$ indicating if transaction t is executed on site s and if attribute a is located on site s, respectively. All transactions must be located at exactly one site (their primary executing site), that is

$$\sum_{t \in \mathcal{T}} x_{t,s} = 1 \quad, \forall s \in S \qquad (2)$$

and all attributes must be located at at least one site, that is

$$\sum_{a \in \mathcal{A}} y_{a,s} \geq 1 \quad, \forall s \in S.$$

To determine the size of A and B from equation (1) introduce five new static binary constants describing the database schema:

- $\alpha_{a,q}$ indicates if attribute a itself is accessed by query q
- $\beta_{a,q}$ indicates if attribute a is part of a table that q accesses
- $\gamma_{q,t}$ indicates if query q is used in transaction t
- δ_q indicates if query q is a write query
- $\varphi_{a,t}$ indicates if any query in transaction t reads attribute a

Single-sitedness should be maintained for reads. That is, if a read query in transaction t accesses attribute a then a and t must be co-located:

$$x_{t,s}\varphi_{a,t} = 1 \Rightarrow y_{a,s} = 1 \quad, \forall t \in \mathcal{T}, a \in \mathcal{A}$$

or equivalently

$$y_{a,s} - x_{t,s}\varphi_{a,t} \geq 0 \quad, \forall t \in \mathcal{T}, a \in \mathcal{A}.$$

In order to estimate the cost of reading, writing and transferring data, introduce the following weights:

- w_a denotes the average width of attribute a
- f_q denotes the frequency of query q
- $n_{a,q}$ denotes for query q the average number of rows retrieved from or written to the table holding attribute a

Then the cost of reading or writing a in query q is estimated to $W_{a,q} = w_a \cdot f_q \cdot n_{a,q}$ and the cost of transferring attribute a over the network is estimated to $pW_{a,q}$. Notice, that $W_{a,q}$ is only an estimate due to f_q and $n_{a,q}$.

Consider the amount of local data access, A, and let $A = A_R + A_W$ where A_R and A_W is the amount of read and write access, respectively. For a given site r and query q, A_R is the sum of all attribute weights $W_{a,q}$ for which 1) q is a read query, 2) attribute a is stored on r, 3) the transaction that executes query q is executed on r and 4) q accesses any attribute in the table fraction that holds a. As we maintain single-sitedness for reads, $\beta_{a,q}$ can be used to handle 4), resulting in

$$A_R = \sum_{a,t,s,q} W_{a,q}\beta_{a,q}\gamma_{q,t}(1 - \delta_q)x_{t,s}y_{a,s}.$$

Accounting for local access of write queries, A_W, is less trivial. Consider the following three approaches:

Access relevant attributes An attribute a at site s should be accounted for if and only if there exists an attribute a' on s that q updates so that a and a' are attributes of the same table. While this accounting is the most accurate of the three it is also the most expensive as it implies an element of the form $y_{a,s}y_{a',s}$ in the objective function which adds an undesirable amount of $|\mathcal{A}|^2|\mathcal{S}|$ variables and $3|\mathcal{A}|^2|\mathcal{S}|$ constraints to the problem when linearized (see Section II-C).

Access all attributes We can get around the increased complexity by assuming that write queries q always writes to all sites containing table fractions of tables accessed by q, regardless of whether q actually accesses any of the attributes of the fractions. While this is correct for insert statements (assuming that inserts always write complete rows) it is likely an overestimation for updates: imagine a lot of single-attribute updates on a wide table where the above method would have split the attribute in question to a separate partition. This overestimation will imply that the model will partition tables that are updated often or replicate attributes less often than the accounting model described above.

Access no attributes Another approach to simplify the cost function is to completely avoid accounting for local access for writes and solely let the network transfer define the write costs. With this underestimation of write costs, attributes will then tend to be replicated more often than in the first accounting model.

In this paper we choose the second approach, which gives a conservative overestimate of the write costs as we then obtain more accurate costs for inserts and avoid extending the model with undesirably many variables and constraints. Intuitively speaking, this choice implies that read queries will tend to partition the tables for best possible read-performance, and the write queries will tend to minimize the amount of attribute replication. We now have

$$A_W = \sum_{q,a,s} W_{a,q}\beta_{a,q}\delta_q y_{a,s}$$

and thus

$$A = \sum_{a,t,s,q} W_{a,q}\beta_{a,q}\gamma_{q,t}(1 - \delta_q)x_{t,s}y_{a,s}$$
$$+ \sum_{q,a,s} W_{a,q}\beta_{a,q}\delta_q y_{a,s}. \qquad (3)$$

B accounts for the amount of network transfer and since we enforce single-sitedness for all reads B is solely the sum of transfer costs for write queries. We assume that write queries only transfer the attributes they update and does not transfer to the site that holds their own transaction:

$$B = \sum_{a,t,s,q} W_{a,q}\alpha_{a,q}\gamma_{q,t}\delta_q(1 - x_{t,s})y_{a,s}.$$

By noticing that $\sum_{a,t,s,q} \alpha_{a,q}\gamma_{q,t}y_{a,s} = \sum_{a,s,q} \alpha_{a,q}y_{a,s}$ we

can construct the minimization problem as

$$
\begin{aligned}
\min \quad & \sum_{t,a,s} c_1(a,t)x_{t,s}y_{a,s} + \sum_{a,s} c_2(a)y_{a,s} \\
\text{s.t.} \quad & \sum_s x_{t,s} = 1 && \forall t \\
& \sum_s y_{a,s} \geq 1 && \forall a \\
& y_{a,s} - x_{t,s}\varphi_{a,t} \geq 0 && \forall a,t \\
& x_{t,s}, y_{a,s} \in \{0,1\} && \forall t,a,s
\end{aligned} \tag{4}
$$

where

$$
c_1(a,t) = \sum_q W_{q,a}\gamma_{q,t}(\beta_{a,q}(1-\delta_q) - p\alpha_{a,q}\delta_q)
$$

and

$$
c_2(a) = \sum_q W_{a,q}\delta_q(\beta_{a,q} + p\alpha_{a,q}).
$$

Both c_1 and c_2 are completely induced by the schema, query workload and statistics and can therefore be considered static when the partitioning process starts.

B. Adding load balancing

We are interested in extending the model in (4) to also handle load balancing of the sites instead of just minimizing the sum of all data access/transfer. From equation (3) define the work of a single site $s \in \mathcal{S}$ as

$$
\sum_{a,t} c_3(a,t)x_{t,s}y_{a,s} + \sum_a c_4(a)y_{a,s} \tag{5}
$$

where $c_3(a,t) = \sum_q W_{a,q}\gamma_{q,t}\beta_{a,q}(1-\delta_q)$ and $c_4(a) = \sum_q W_{a,q}\beta_{a,q}\delta_q$. Introduce the variable m and for each site s let the value of (5) be a lower bound for m. Adding m to the objective function is then equivalent to also minimizing the work of the maximally loaded site.

In order to decide how to prioritize cost minimization versus load balancing in the model, introduce a scalar $0 \leq \lambda \leq 1$ and weight the original cost from (4) and m by λ and $(1-\lambda)$, respectively. The new objective is then

$$
\lambda \sum_{a,t,s} c_1(a,t)x_{t,s}y_{a,s} + \lambda \sum_{a,s} c_2(a)y_{a,s} + (1-\lambda)m \tag{6}
$$

where m is constrained as follows:

$$
\sum_{a,t} c_3(a,t)x_{t,s}y_{a,s} + \sum_{a,q} c_4(a)y_{a,s} \leq m \quad , \forall s \in \mathcal{S}.
$$

Notice that while we are now minimizing (6), the objective of (4) should still be considered as the actual cost of a solution.

C. Linerarization

We use the technique discussed in [6, Chapter IV, Theorem 4] to linearize the model. This is done by replacing the quadratic terms in the model with a variable $u_{t,a,s}$ and adding the following new constraints:

$$
\begin{aligned}
u_{t,a,s} &\leq x_{t,s} && \forall t,a,s \\
u_{t,a,s} &\leq y_{a,s} && \forall t,a,s \\
u_{t,a,s} &\geq x_{t,s} + y_{a,s} - 1 && \forall t,a,s
\end{aligned}
$$

For $u_{t,a,s} \geq 0$ notice that $u_{t,a,s} = 1$ if and only if $x_{t,s} = y_{a,s} = 1$, and that $u_{t,a,s}$ is guaranteed to be binary if both

$x_{t,s}$ and $y_{a,s}$ are binary (thus, there is no need for requiring it explicitly in the model).

Now, the model in (4) extended with load balancing looks as follows when linearized:

$$
\begin{aligned}
\min \quad & \lambda \sum_{t,a,s} c_1(a,t)u_{t,a,s} \\
& + \lambda \sum_{a,s} c_2(a)y_{a,s} + (1-\lambda)m \\
\text{s.t.} \quad & \sum_s x_{t,s} = 1 && \forall t \\
& \sum_s y_{a,s} \geq 1 && \forall a \\
& y_{a,s} - x_{t,s}\varphi_{a,t} \geq 0 && \forall a,t \\
& \sum_{a,t} c_3(a,t)u_{a,t,s} \\
& \quad + \sum_{a,q} c_4(a)y_{a,s} \leq m && \forall s \\
& u_{t,a,s} - x_{t,s} \leq 0 && \forall t,a,s \\
& u_{t,a,s} - y_{a,s} \leq 0 && \forall t,a,s \\
& u_{t,a,s} - x_{t,s} - y_{a,s} + 1 \geq 0 && \forall t,a,s \\
& x_{t,s}, y_{a,s} \in \{0,1\} && \forall t,a,s \\
& u_{t,a,s} \geq 0 && \forall t,a,s
\end{aligned} \tag{7}
$$

D. Complexity

The objective function in quadratic programs can be written on the form

$$
\tfrac{1}{2}z^{\mathrm{T}}Qz + cz + d
$$

where in our case $z = (x_{1,1}, \ldots, x_{|\mathcal{T}|,|\mathcal{S}|}, y_{1,1}, \ldots, x_{|\mathcal{A}|,|\mathcal{S}|})$ is a vector containing the decision variables, Q is a cost matrix, c is a cost vector and d a constant. Q can be easily defined from (6) by dividing Q into four quadrants, letting the sub-matrices in the upper-left and lower-right quadrant equal zero and letting the upper-right and lower-left submatrices be defined by $c_1(a,t)$. Q is indefinite and the cost function (6) therefore not convex. As shown by Marty and Judice [9] finding optimum when Q is indefinite is NP-hard.

III. The SA solver – a heuristic approach

We develop a heuristic based on simulated annealing (see [13]) and will refer to it as the *SA*-solver from now on. The base idea is to alternately fix x and y and only optimize the not-fixed vector, thereby simplifying the problem. In each iteration we search in the neighborhood of the found solution and accept a worse solution as base for a further search with decreasing probability.

Let $x_{t,s}$ hold an assignment of transactions to sites and define the neighborhood x' of x as a change of location for a subset of the transactions so that for each $t \in \mathcal{T}$ we still have $\sum_s x'_{t,s} = 1$. Similarly, let $y_{a,s}$ hold an assignment of attributes to sites but define the neighborhood y' of y as an extended replication of a subset of the attributes. That is, for each $a \in \mathcal{A}$ in that subset we have $y_{a,s} = 1 \Rightarrow y'_{a,s}$ and $\sum_s y'_{a,s} > \sum_s y_{a,s}$. We found that altering the location for a constant number of 10% of both transactions/attributes yielded the best results. The heuristic now looks as pictured in Algorithm 1. Notice, that the linearization constraints is not needed since either x or y will be constant in each iteration. This reduces the size of the problem considerably.

Algorithm 1 The heuristic based on simulated annealing (SA). It iteratively fixes x and y and accepts a worse solution from the neighborhood with decreasing probability.

 1: Initialize temperature $\tau > 0$ and reduction factor $\rho \in \,]0;1[$
 2: Set the number L of inner loops
 3: Initialize x randomly so that (2) is satisfied
 4: fix $\leftarrow$ "x"
 5: $S \leftarrow$ findSolution(fix)
 6: **while** not frozen **do**
 7:    **for** $i \in \{1, \ldots, L\}$ **do**
 8:       $x \leftarrow$ neighborhood of x
 9:       $y \leftarrow$ neighborhood of y
10:       $S' \leftarrow$ findSolution(fix)
11:       $\Delta \leftarrow \text{cost}(S') - \text{cost}(S)$
12:       $p \leftarrow$ a randomly chosen number in $[0;1]$
13:       **if** $\Delta \leq 0$ or $p < e^{-\Delta/\tau}$ **then**
14:          $S \leftarrow S'$
15:       **end if**
16:       fix $\leftarrow$ the element in $\{$"x","y"$\} \setminus \{$fix$\}$
17:    **end for**
18:    $\tau \leftarrow \rho \cdot \tau$
19: **end while**

IV. FURTHER IMPROVEMENTS

Consider a table with n attributes together with two queries: one accessing attribute 1 through k and one accessing attribute k through n. Then it is sufficient to find an optimal distribution for the three attribute groupings $\{1, \ldots, k-1\}$, $\{k\}$ and $\{k+1, \ldots, n\}$, considering each group as an atomic unit and thereby reducing the problem size. In general, it is only necessary to distribute groups of attributes induced by query access overlaps. Chu and Ieong [4] refer to these attribute overlaps as *reasonable cuts*. Even though this will not improve the worst-case complexity, this reduction may still have a large performance impact on some instances.

Also, assuming that transactions follow the 20/80 rule (20% of the transactions generate 80% of the load), the problem can be solved iteratively over $\mathcal{T}$ starting with a small set of the most heavy transactions.

V. COMPUTATIONAL RESULTS

In this section we will briefly cover our computational results. A more elaborate description, including comparison tables, can be found in the full version of this paper [3].

We assume that the context is a database with a very high transaction count like the memory-only database H-store [12] (now VoltDB[1]). With a bus bandwith between 32 Gbit/s and 128 Gbit/s, and a PC3 DDR3-SDRAM bandwith of at least 136 Gbit/s, the bus is assumed to be the bottleneck in RAM accesses. We assume the use of a 10-gigabit network and therefore set $p = 8$ in our tests unless otherwise stated.

We furthermore mainly focus on minimizing the total costs of execution and therefore choose a low $\lambda = 0.1$ assignment.

We have performed tests on a model of the TPC-C benchmark[2] version 5.10.1 and a set of randomly generated instances. The TPC-C specification describes transactions, queries and database schema but does not provide the statistics needed to create a problem instance. In our model of TPC-C we have therefore tried to construct a set of statistics that seem realistic.

To the best of our knowledge there is no standard library of typical OLTP instances with schemas, workloads and statistics so in order to explore the characteristics of the algorithms we performed some experiments on a set of randomly generated instances instead as it showed up to be a considerable administrative and bureaucratic challenge (if possible at all) to collect appropriate instances from "real life" databases. The randomly generated instances vary in several parameters in order to clarify which characteristics that influence the potential cost reduction by applying our vertical partitioning algorithms. The parameters include: number of transactions in workload, number of tables in schema, maximum number of attributes per table, maximum number of queries per transaction, percentage of queries being updates, maximum number of different tables being referred to from a single query, maximum number of different attributes being referred to by a single query, and the set of allowed attribute widths. We defined classes of problem instances by upper bounds on all parameters. Individual instances were then generated by choosing the value of each parameter evenly distributed between 1 and its upper bound. That is, if e.g. the maximum allowed number of attributes in tables was k, the number of table attributes for each table in the generated instance would be evenly distributed between 1 and k with a mean of $k/2$.

Our experiments (described in details in [3]) revealed that the SA solver is generally faster than the QP solver but the QP solver obtains lower costs when it does not exceed its time limit. For TPC-C, we obtained a cost reduction of 37% when applying the algorithms and for the random instances where we expected to see a reduction, the reduction was between 25% and 85%.

We ran a series of experiments comparing disjoint and nondisjoint partitioning. A far greater cost reduction could be obtained when allowing replication but in exchange to increased computation time.

We also compared local vs. remote partition placements, modelled by setting $p = 0$ and $p > 0$, respectively. The benefits of local placements are given by the amount of updates in the workload as only updates cause inter-site transfers. We saw up to 33% cost reduction in our experiments by placing partitions locally.

VI. CONCLUSION

We have constructed a cost model for vertical partitioning of relational OLTP databases together with a quadratic integer program that distributes both attributes and transactions to a set

---

[1]http://voltdb.com

[2]http://www.tpc.org/tpcc

of sites while allowing attribute replication, preserving single-sitedness for read queries and in which load balancing vs. total cost minimization can be prioritized arbitrarily. We also presented a more scalable heuristic which seems to deliver good results. For both algorithms we obtained a cost reduction of 37% in our model of TPC-C and promising results for the random instances. Even though the latter theoretically can be constructed with arbitrary high/low benefits from vertical partitioning, the test runs on our selected subset of random instances seem to indicate that 1) our heuristic scales far better than the QP-solver, and 2) it can obtain valuable cost reductions on many real-world OLTP databases, as we tried to select the parameters realistically.

One thing we miss, however, is an official OLTP testbed – a library containing realistic OLTP workloads, schemas and statistics. Such a collection of realistic instances could serve as base for several insteresting and important studies for understanding the nature and characteristics of OLTP databases.

REFERENCES

[1] R. Agarwal, C. Aggarwal, and V. Prasad. A tree projection algorithm for generation of frequent item sets. *Journal of Parallel and Distributed Computing*, Jan 2001.

[2] S Agrawal, V Narasayya, and B Yang. Integrating vertical and horizontal partitioning into automated physical database design. *Proceedings of the 2004 ACM SIGMOD international ...*, Jan 2004.

[3] Rasmus Resen Amossen. Vertical partitioning of relational oltp databases using integer programming. 2009. doi: arXiv:0911.1691. URL http://arxiv.org/abs/0911.1691.

[4] W. Chu and I. Ieong. A transaction-based approach to vertical partitioning forrelational database systems. *IEEE Transactions on Software Engineering*, Jan 1993.

[5] Douglas W. Cornell and Philip S. Yu. An effective approach to vertical partitioning for physical design of relational databases. *IEEE Trans. Softw. Eng.*, 16(2):248–258, 1990. ISSN 0098-5589.

[6] P. L. Hammer and S. Rudeanu. *Boolean Methods in Operations Research and Related Areas*. Springer Verlag, 1968. ISBN 0-387-04291-1.

[7] W. McCormick Jr, P. Schweitzer, and T. White. Problem decomposition and data reorganization by a clustering technique. *Operations Research*, Jan 1972.

[8] Robert Kallman, Hideaki Kimura, Jonathan Natkins, Andrew Pavlo, Alexander Rasin, Stanley Zdonik, Evan P. C. Jones, Samuel Madden, Michael Stonebraker, Yang Zhang, John Hugg, and Daniel J. Abadi. H-store: a high-performance, distributed main memory transaction processing system. *Proc. VLDB Endow.*, 1(2):1496–1499, 2008.

[9] Katta G. Marty and Joaquim Judice. On the complexity of finding stationary points of nonconvex quadratic programs. *Opsearch*, 33(3):162–166, 1996.

[10] Shamkant Navathe, Stefano Ceri, Gio Wiederhold, and Jinglie Dou. Vertical partitioning algorithms for database design. *ACM Trans. Database Syst.*, 9(4):680–710, December 1984. ISSN 0362-5915.

[11] J. Son and M. Kim. An adaptable vertical partitioning method in distributed systems. *The Journal of Systems & Software*, Jan 2004.

[12] Michael Stonebraker, Samuel R. Madden, Daniel J. Abadi, Stavros Harizopoulos, Nabil Hachem, and Pat Helland. The end of an architectural era (it's time for a complete rewrite). In *VLDB*, Vienna, Austria, 2007.

[13] Laurence A. Wolsey. *Integer Programming*. Wiley-Interscience, 1998. ISBN 0-471-28366-5.

APPENDIX

This section describes how to extend the algorithms to also estimate costs of network latency for queries accessing attributes on remote sites. We assume, that all remote access (if any) for queries are done in parallel and with a constant number of requests per query per remote site. Let p_l denote a latency penalty factor and introduce a new binary variable ψ_q for each query q indicating with $\psi_q = 1$ if q accesses any remotely placed attributes. Letting n denote the number of remotely accessed attributes by q we have $n > 0 \Rightarrow \psi_q = 1$ and $n = 0 \Rightarrow \psi_q = 0$, or equivalently $(\psi_q - 1)n = 0$ and $\psi_q - n \leq 0$. This results in the following two classes of new constraints:

$$(\psi_q - 1) \sum_{a,s} \delta_q \alpha_{a,q} \gamma_{q,t} (1 - x_{t,s}) y_{a,s} = 0 \quad , \forall q, t$$

and

$$\psi_q - \sum_{a,s} \delta_q \alpha_{a,q} \gamma_{q,t} (1 - x_{t,s}) y_{a,s} \leq 0 \quad , \forall q, t$$

The total latency in a given partitioning can now be estimated by the sum $p_l \sum_q f_q \psi_q$ which can be added to the cost objective function (4).