Skew-Aware Automatic Database Partitioning in Shared-Nothing, Parallel OLTP Systems

Andrew Pavlo, Carlo Curino, Stan Zdonik
Motivation

- **OLTP** systems: operational systems that support transactional applications online (e.g., online banking and airline booking)

- It is difficult to derive an optimal database design for enterprise-class **OLTP** systems

- Challenges:
  1) the use of stored procedures
  2) the need for load balancing in the presence of time-varying skew
  3) complex schemas
  4) deployments with larger number of partitions

- Solution: Horticulture
H-Store

Figure 1: An overview of the H-Store parallel OLTP DBMS.
Key Issue #1: Distributed Transactions

- How the presence of distributed transactions affects performance:
  - all single-partition transactions is better than distributed transactions
  - the throughput decreased as # of distributed transactions increased
  - performance difference increases as the number of partitions increases

**Conclusion**: a design that minimizes both the # of distributed transactions and the # of partitions will obtain the best performance
Key Issue #2: Temporal Skew

- The impact of temporal skew:

- For a large number of partitions, an extra 5% of the overall load is skewed towards a single-partition can lead to a significant reduction of the throughput

Conclusion: both distributed transactions and temporal workload skew must be considered when deriving the optimal database design.
Figure 4: The Horticulture tool generates a database design that splits tables into horizontal partitions (Fig. 4a), replicates tables on all partitions (Fig. 4b), replicates secondary indexes on all partitions (Fig. 4c), and routes transaction requests to the best base partition (Fig. 4d).
Design Options

- Horizontal Partitioning
- Table Replication
- Secondary Indexes
- Stored Procedure Routing

Figure 4: The Horticulture tool generates a database design that splits tables into horizontal partitions (Fig. 4a), replicates tables on all partitions (Fig. 4b), replicates secondary indexes on all partitions (Fig. 4c), and routes transaction requests to the best base partition (Fig. 4d).
Horizontal Partitioning

• A table can be horizontally divided into multiple, disjoint fragments whose boundaries are based on the values of one (or more) of the table’s columns.

• The DBMS assigns each tuple to a particular fragment based on the values of these attributes and then combines related fragments from multiple tables together into a partition
Table Replication

- A table can be replicated across all partitions.
- Replication is useful for read-only or read-mostly tables that are accessed together with other tables without sharing the same foreign key ancestors.
Secondary Indexes

- When a query accesses a table through an attribute that is not the partitioning attribute, it is broadcasted to all nodes.

- In some cases, however, these queries can become single-partitioned if the database includes a secondary index for a subset of a table’s columns that is replicated across all partitions.
Stored Procedure Routing

- The DBMS uses a procedure’s routing attribute(s) to redirect a new transaction request to a node that will execute it to ensure that transaction requests can be effectively routed.

- The best routing attribute for each procedure enables the DBMS to identify which node has the data that each transaction needs, as this allows them to potentially execute with reduced concurrency control.
Figure 5: An overview of Horticulture’s LNS design algorithm.

LARGE-NEIGHBORHOOD SEARCH
Horticulture’s design algorithm
Outline of LNS

1. Analyze the sample workload trace to pre-compute information used to guide the search process.

2. Generate an initial “best” design $D_{best}$ based on the database’s most frequently accessed columns.

3. Create a new incomplete design $D_{relax}$ by “relaxing” (i.e., resetting) a subset of $D_{best}$.

4. Perform a local search for a new design using $D_{relax}$ as a starting point. If any new design has a lower cost than $D_{best}$, then mark it as the new $D_{best}$. The search stops when a certain number of designs fail to improve on $D_{best}$, or there are no designs remaining in $D_{relax}$’s neighborhood.

5. If the total time spent thus far exceeds a limit, then halt the algorithm and return $D_{best}$. Otherwise, repeat Step 3 for a new $D_{relax}$ derived from $D_{best}$.

Figure 5: An overview of Horticulture’s LNS design algorithm.
Initial Design

1. **Horizontal Partitioning:** Select the most frequently accessed column in the workload as the horizontal partitioning attribute for each table.

2. **Table Replication:** Greedily replicate read-only tables if they fit within the partitions’ storage space limit.

3. **Secondary Indexes:** Select the next most frequently accessed, read-only column in the workload as the secondary index attribute for each table if they fit within the partitions’ storage space limit.

4. **Stored Procedure Routing:** Select the routing parameter for stored procedures based on how often the parameters are referenced in queries that use the table partitioning columns selected in Step 1.
Relaxation

Relaxation is the process of selecting random tables in the database and resetting their chosen partitioning attributes in the current best design. Relaxation allows LNS to escape a local minimum and to jump to a new neighborhood of potential solutions.

To generate a new relaxed design, Horticulture must decide:

(1) How many tables to relax

(2) Which tables to relax

(3) What design options will be examined for each relaxed table in the local search.
Horticulture executes a two-phase search algorithm to iteratively explore solutions using the relaxed design $D_{relax}$ produced in the previous step.

This process is represented as a search tree, where each level of the tree coincides with one of the relaxed database elements.
SKEW-AWARE COST MODEL

\[ \text{cost}(\mathcal{D}, \mathcal{W}) = \frac{(\alpha \times \text{CoordinationCost}(\mathcal{D}, \mathcal{W})) + (\beta \times \text{SkewFactor}(\mathcal{D}, \mathcal{W}))}{(\alpha + \beta)} \]

- W: workload
- D: design
- $\alpha$ & $\beta$: parameters that are configured by the administrator
- The model measures:
  1) how much of a workload executes as single-partition transactions
  2) how uniformly load is distributed across the cluster
Coordination Cost

- \( \text{CoordinationCost}(D, W) \) calculates how well \( D \) minimizes \# multi-partition transactions in \( W \).

- The coordination cost is calculated as the ratio of the total \# of partitions accessed (\( \text{partitionCount} \)) divided by the total number of partitions that could have been accessed (\( \text{txnCount} \times \text{numPartitions} \)).

- The result is then scaled based on the ratio of distributed (\( \text{txnCount} \)) to single-partition transactions (\( \text{dtxnCount} \)).

\[
\text{CoordinationCost}(D, W) = \frac{\text{partitionCount}}{(\text{txnCount} \times \text{numPartitions})} \times \left(1.0 + \frac{\text{dtxnCount}}{\text{txnCount}}\right)
\]

- \( D \): design
- \( W \): workload
- \( \text{txn} \): transaction
- \( \text{txnCount} \): total \# single-partition transactions
- \( \text{dtxnCount} \): total \# distributed transactions
- \( P \): partitions
- \( \text{partitionCount} \): total \# partitions accessed
Skew Factor

- $SkewFactor(D, W)$ calculates how well the design minimizes skew in the database.

- To ensure that skew measurements are not masked by time, the $SkewFactor$ function divides $W$ into finite intervals ($numIntervals$) and calculates the final estimate as the arithmetic mean of the skew factors weighted by the number of transactions executed in each interval.

Algorithm 2 $SkewFactor(D, W)$

```plaintext
skew ← [], txnCounts ← []
for i ← 0 to numIntervals do
    skew[i] ← CalculateSkew(D, W, i)
    txnCounts[i] ← NumTransactions(W, i)
end for
return \[
\left( \frac{\sum_{i=0}^{numIntervals} skew[i] \times txnCounts[i]}{\sum txnCounts} \right)
\]
```

- $D$: design
- $W$: workload
- $txn$: transaction
- $txnCount$: total # single-partition transactions
- $numIntervals$: administrator-defined parameter
Optimizations

Access Graphs

- Horticulture extracts the key properties of transactions from a workload trace and stores them in undirected, weighted graphs, called access graphs.

- Access graphs allow the tool to quickly identify important relationships between tables without repeatedly reprocessing the trace.

![Figure 7: An access graph derived from a workload trace.](image)

Workload Compression

- Compresses redundant transactions and redundant queries to reduce the computation time.

- Compressing a transactional workload is a two-step process:

  1) combine sets of similar queries in individual transactions into fewer weighted records

  2) combine similar transactions into a smaller number of weighted records in the same manner
Evaluation: Throughput

Figure 9: Transaction throughput measurements for the HR+, SCH, and MFA design algorithms.
Evaluation: Search Times

**Figure 13:** The best solution found by Horticulture over time. The red dotted lines represent known optimal designs (when available).
References
