Twisted Twin (Technical Report): Supplementary Material and Experiments

Jiani Yang
Zhejiang University
jianiyang_cs@zju.edu.cn

Sai Wu
Zhejiang University
wusai@zju.edu.cn

Yong Wang
Huawei
wangyong308@huawei.com

Dongxiang Zhang
Zhejiang University
zhangdongxiang@zju.edu.cn

Yifei Liu
Zhejiang University
liuyifei0@zju.edu.cn

Xiu Tang
Zhejiang University
tangxiu@zju.edu.cn

Gang Chen
Zhejiang University
cg@zju.edu.cn

1 APPENDIX

1.1 Extended Experiment Results

1.1.1 Performance Estimator. Tsuei et al. [13] models the impact of buffer size on OLTP throughput using a two-stage regression model. This method serves as a natural starting point for our study. As shown in Figure 1, this model takes the hit ratio and buffer size $\mathcal{M}_{\text{row}}$ as inputs and fits two curves:

$$HitRatio = w_0 \cdot \ln(\mathcal{M}_{\text{row}}) + w_1$$

$$TPS = w_2 \cdot HitRatio + w_3$$

Here, w_0, w_1, w_2, and w_3 are the model parameters determined through regression analysis. In the figure 1, the straight line represents the predicted TPS, while the scatter points indicate the actual sample values. In this figure, we can observe that the prediction performance for the data within the red box is poor, with some predicted values significantly higher than the actual values. Upon further investigation, we find that executing analytical queries may affect the OLTP throughput. If analytical queries need to retrieve data from the row store, they can cause transactional queries to experience longer wait times due to I/O requests.

Given this limitation, we turned to more advanced models to better capture the intricate relationships between OLTP and OLAP workloads. Additionally, it is insufficient to only consider the effect of the row store buffer size on TPS; we need to extract additional features to enhance the model's accuracy.

We observed that $\mathcal{M}_{\text{row}}$ is the most significant factor affecting the OLTP system's TPS, though TPS is also influenced by the execution of analytical queries. To systematically evaluate the impact of these analytical queries on OLTP performance, we identified major disruptions—primarily resulting from I/O requests or data page locks initiated by these queries.

To better capture such interference, we construct a richer feature space to train our TPS prediction model. Each sample is represented as a tuple $<\mathcal{M}_{\text{row}}, \mathcal{W}\text{ap}, C>$, where $\mathcal{M}_{\text{row}}$ and C denote the memory allocated to the row store and the columns loaded into the column store, respectively. From $\mathcal{W}_{\text{ap}}$, we extract two feature vectors: $\mathbf{scan}_{\text{seq}}$, with $\mathbf{scan}_{\text{seq}}[j]$ indicating the number of sequential scans on table j, and $\mathbf{scan}_{\text{index}}$, with $\mathbf{scan}_{\text{index}}[k]$ indicating the number of index scans on index k. These features quantify the impact of OLAP queries on OLTP performance.

Metric. The evaluation metrics in this study are Mean Absolute Error (MAE) and Mean Absolute Relative Error (MARE), defined as follows:

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|, \quad \text{MARE} = \frac{1}{N}\sum_{i=1}^{N}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \tag{1}$$

In this equation, y_i is the true value and $\hat{y}_i$ is the predicted value for the i-th sample.

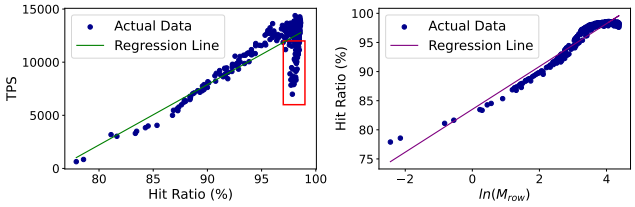

Figure 1: Row store buffer size's effect on TPS

Model	Log Reg.	RF	SVR	MLP	GBT
Training(ms)	47.67	674.64	13.31	1223.75	245.13
Inference(ms)	0.01	0.27	0.10	0.04	0.04
MAE(TPS)	1276.87	563.04	1889.10	2008.90	**556.16**
MARE	13.72%	5.34%	23.38%	19.33%	**5.23%**

Table 1: Performance Comparison of Different Algorithms for Predicting TPS

Training and Inference Overhead. Historical execution data from the database logs can be collected as training samples. Additionally, simulations can be run during idle periods to gather more samples. Table 1 shows the training and inference times of our model.

Comparison Approaches. We evaluated several models that are well-suited for regression tasks and particularly effective at capturing nonlinear relationships. The models include Logarithmic Regression (Log Reg.) [13], Random Forest (RF) [3], Support Vector Regression (SVR) [4], Multi-layer Perceptron (MLP) [11], and Gradient Boosted Trees (GBT) [5]. All models are trained on the same dataset and use the same set of features as the GBT model.

Accuracy and Overhead. As shown in Table 1, tree-based models, such as Random Forest (RF) and Gradient Boosting Trees (GBT), significantly outperform other models in terms of accuracy. Tree-based models tend to perform better, especially when the sample size is small and many discrete features are present. While RF and GBT exhibit similar performance, GBT demonstrates slightly higher accuracy and relatively fast inference times, making it well-suited for real-time applications. Therefore, in this paper, we chose GBT as the TPS Prediction Model.

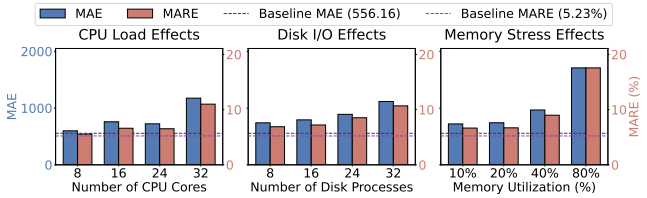

Figure 2: Impact of Unpredictable Environments on Performance Estimator

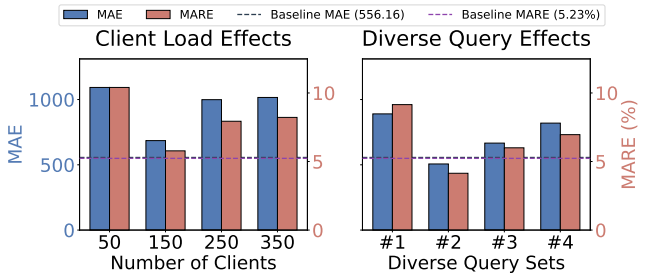

Figure 3: Impact of client numbers and diverse queries.

Handling Unpredictable Environments. GaussDB-HTAP is deployed either on a standalone machine or within resource-isolated cloud containers. In actual production environments, standalone machine deployments are very common due to their strong resource isolation, which helps to minimize external interference and ensure stable system performance. Based on this practical reality, we assume a stable deployment environment in our paper, which enables accurate estimation of the required memory mix. However, when deployed in a cloud environment, factors such as resource oversubscription may reduce the available CPU, memory, and IO resources. Our model does not account for the impact of such external resource contention. To address this, we conducted experiments to investigate how these factors influence the performance of our model.

Robustness of Performance Estimator. In this experiment, we utilized stress-ng [1] to perform stress testing on three key system resources: CPU, memory, and disk I/O. We used stress-ng to disturb the original program to test how it behaves under high load conditions. We still use the runtime data collected from isolated environments to train the Performance Estimator, which is employed here to predict the system throughput under the influence of additional factors. Figure 2 shows the experimental result, as evaluated by MAE and MARE (refer to Equation 1).

- **CPU Contention Test.** We used stress-ng to simulate varying CPU loads by increasing the number of active cores. The model maintained stable accuracy (MARE < 10%) up to 24 cores, but accuracy degraded beyond 32 cores.
- **Disk I/O Contention Test.** Disk stress was applied using multiple worker processes. Prediction accuracy remained stable up to 24 processes but dropped noticeably at 32.
- **Memory Contention Test.** We gradually increased memory usage up to 80%, simulating resource oversubscription. When 80% of the memory resources were occupied by other processes, the actual memory available for GaussDB was insufficient (due to the operating system's virtual memory mechanism, this situation does not result in an immediate error but instead relies on the swap mechanism to adjust), leading to a significant increase in MARE. However, in cases with lower memory utilization, this had minimal impact on model efficiency.

We also conduct experiments to find out how other factors may influence the performance, we show the results in Figure 3.

- **Parallelism.** Regarding inter-query parallelism, we simulated realistic application scenarios by using 200 TP clients and 5 AP clients that send queries at intervals controlled by specific query rates. To evaluate the effect of parallelism on system throughput, we varied the number of clients. The results show that although varying client counts introduce some interference in throughput predictions, the impact is not substantial. We plan to incorporate this effect in future work. For intra-query parallelism, we set the Query DOP (Degree of Parallelism) to 8 in our experiment.
- **Buffer pool state.** Our experiment shows that clearing the buffer pool prior to sample collection results in a MAPE of 9.67% and an MAE of 1126.71 for TPS prediction. This decrease in accuracy indicates that the buffer pool state does impact the TPS predictor, although the effect is not drastic—especially considering that clearing the buffer pool represents one of the most extreme state changes. In practice, the distribution of data pages, cache hit rate, number of free pages, and the state of dirty pages may all influence TPS prediction accuracy. Incorporating these runtime features into our TPS predictor remains an interesting direction for future work.
- **Queries.** Since we used the number of table scans and index scans as input features when designing the TPS predictor model, the model is not dependent on specific workloads. To test whether our model can adapt to diverse queries, we generated OLAP queries that differ from those used in the experiments. As shown in the Figure , we tested our model on four newly generated query sets and found that there was a slight impact on accuracy, but the impact was minimal, with MAPEs all below 10%. This indicates that the OLAP-related features we extracted are highly effective and applicable to diverse queries.

This experiment can also simulate the situation of cloud resource overprovisioning, where more resources are allocated than are physically available. This scenario is similar to cases where resources are contended by other processes. Therefore, from the above results, we can observe the model's performance when faced with resource oversubscription in CPU, memory, and I/O.

Table 2 illustrates the performance of T^2 under CPU, disk, and memory disturbances. While T^2 generally maintains high efficiency, in extreme cases—such as a 32-core CPU load, 32 concurrent disk I/O processes, or 80% memory usage—the reduced accuracy of the TPS predictor leads to insufficient row-store memory allocation, resulting in $FR < 1$.

While T^2 assumes stable system conditions for optimal prediction accuracy, we have now thoroughly evaluated its behavior under a variety of disturbance scenarios. The estimator demonstrates robust performance under moderate noise. In extreme cases (e.g., severe memory contention), accuracy degrades, occasionally leading to suboptimal memory allocation. We plan to incorporate runtime resource feedback (e.g., system metrics or resource usage) into the model in future work to further improve its adaptability.

Disturb		CPU(cores)		Disk		Memory	
		16	32	16	32	40%	80%
Original GaussDB	FR	1.00	1.00	1.00	0.99	1.00	0.98
	AP RT(s)	1.82	1.87	1.92	1.97	2.01	2.02
STMM	FR	1.00	1.00	1.00	0.99	1.00	0.98
	Impr(%)	53.30	52.94	47.40	42.13	51.74	51.49
	AP RT(s)	0.85	0.88	1.01	1.14	0.97	0.98
T^2 static	FR	1.00	0.99	1.00	0.96	1.00	0.98
	Impr(%)	54.40	54.01	53.13	52.79	53.23	52.48
	AP RT(s)	0.83	0.86	0.90	0.93	0.94	0.96
T^2 dynamic	FR	1.00	0.99	1.00	0.97	1.00	0.99
	Impr(%)	68.13	66.31	64.58	64.47	68.66	67.33
	AP RT(s)	0.58	0.63	0.68	0.70	0.63	0.66

Table 2: Performance of T^2 under various disturbance scenarios.

Model	Train Time (s)	Inference Time (s)	MAPE (%)
PatchTST	33.53	0.0051	7.52
ARIMA	1.10	0.0073	37.27
Prophet	2.96	0.0847	26.46
LSTM	15.14	0.0441	41.83

Table 3: Workload Predictor Comparison.

Setting	RT<0.9s	RT<0.8s	RT<0.7s	RT<0.6s
AP RT(s)	0.81	0.75	0.68	0.67
TP FR	1.00	0.98	0.97	0.96
Memory Ratio	21:59	14:66	12:68	11:69

Table 4: Performance of T^2-Static with OLAP as the Top Priority. Memory Ratio indicates the ratio of $\mathcal{M}_{\text{row}}$ with $\mathcal{M}_{\text{col}}$.($\mathcal{M}_{\text{total}} = 80GB$)

1.1.2 Workload Predictor. The workload predictor is used to forecast the future query rates of both OLTP and OLAP workloads. We extracted the request rates for OLTP and OLAP queries from client logs. Using a 24-minute interval as a time segment, we recorded the number of requests in each segment. We collected 15 days of request data and used the first 14 days as the training set, leaving the final day for prediction of request volume in each time segment.

We evaluated several common time series forecasting models, including ARIMA[2], Prophet[12], LSTM[6], and PatchTST[9]. PatchTST can jointly train and predict across multiple time series at once, capturing inter-series relationships. Consequently, it requires training only a single model. By contrast, ARIMA, Prophet, and LSTM each need a separate model per time series. Table 3 reports training time, inference time, and accuracy for each model. While PatchTST has a longer training time, its accuracy is substantially higher, and the total training time remains on the order of seconds, which meets our system requirements. Therefore, we chose PatchTST as the model for our Workload Predictor.

1.1.3 Applied to Priority Shift Scenario. Our proposed framework can be readily adapted to accommodate changes in the priority between OLTP and OLAP workloads, based on user-defined requirements. In real-world deployments, we occasionally work with clients who prioritize OLAP performance over OLTP, requiring

$\mathcal{M}_{\text{total}}$	Objective	STMM	T^2-Static	T^2-Dynamic
80G	Cost	51.17%	54.44%	67.07%
	IO	50.37%	53.03%	66.40%
60G	Cost	36.78%	39.33%	53.69%
	IO	36.01%	36.73%	51.01%

Table 5: OLAP Impr under Different Optimization Objectives (IO and Cost)

that the average response time of OLAP queries be bounded within a specific threshold. In such cases, the optimization objective in Equation 2.2.1 of our paper can be reformulated as follows:

$$\begin{aligned} \text{Maximize} \quad & \mathcal{P}_{OLTP}(\mathcal{M}_{\text{row}}) \\ \text{Subject to} \quad & \frac{\text{Cost}_{\mathcal{W}_{\text{ap}}}(C) + \text{Cost}_{\Delta}(C, \mathcal{K})}{N_{ap}} \leq \tau \\ & \mathcal{M}_{\text{row}} + \mathcal{M}_{\text{col}} \leq \mathcal{M}_{\text{total}}, \\ & W(C) \leq \mathcal{M}_{\text{col}}. \end{aligned}$$

Here, τ is set by clients as the response time limit for OLAP queries, and N_{ap} denotes the number of OLAP queries being executed. To solve this optimization problem, we can still follow the Bayesian optimization algorithm proposed in Section 3.3, with a little modification on the objective function of $\mathcal{U}(\mathcal{M}_{\text{col}})$:

$$\mathcal{U}'(\mathcal{M}_{\text{col}}) = \begin{cases} \mathcal{P}_{OLTP}(\mathcal{M}_{\text{row}}, \mathcal{W}_{\text{ap}}, C), & \text{if } \frac{\text{Cost}_{\mathcal{W}_{\text{ap}}}(C)+\text{Cost}_{\Delta}(C,\mathcal{K})}{N_{ap}} \leq \tau, \\ -\left(\frac{\text{Cost}_{\mathcal{W}_{\text{ap}}}(C)+\text{Cost}_{\Delta}(C,\mathcal{K})}{N_{ap}} - \tau\right), & \text{if } \frac{\text{Cost}_{\mathcal{W}_{\text{ap}}}(C)+\text{Cost}_{\Delta}(C,\mathcal{K})}{N_{ap}} > \tau. \end{cases}$$

In this case, the objective of Bayesian optimization is to find a configuration of $\mathcal{M}_{\text{row}}$ and $\mathcal{M}_{\text{col}}$ that maximizes $\mathcal{U}'(\mathcal{M}_{\text{col}})$. When the OLAP response time does not meet the requirement, the objective function guides the algorithm to select configurations that bring the response time as close to τ as possible.

As shown in Figure 4, we conducted tests to evaluate the performance of the T^2 optimization framework under various OLAP response time constraints: 0.9s, 0.8s, 0.7s, and 0.6s. The configuration used for these tests was the same as in Section 5.2 of the paper. When we adjust the objective of the Bayesian optimization algorithm, T^2 prioritizes satisfying clients' OLAP response time requirements. According to our experimental results, the average response times for OLAP queries meet clients' demands in all cases, except when the response time is set to below 0.6s. In this case, the OLAP query response time encounters bottlenecks beyond memory size limitations.

1.1.4 Evaluation of I/O-Based vs. Cost-Based Objectives. To evaluate the feasibility of using I/O cost as a proxy for full query cost in the T^2 algorithm, we conducted a comparative experiment. Table 5 shows the OLAP improvement under two different optimization objectives: minimizing I/O cost and minimizing overall execution cost. As shown, the performance of T^2 under the I/O-based objective is nearly identical to that under the cost-based objective, suggesting that I/O cost is a reasonable proxy in many cases.

However, this simplification overlooks two important factors. First, data read from the row store undergoes expression filtering within the row engine, which limits opportunities for vectorized execution. Second, since OLAP queries are executed using a vectorized engine, row-store data must be transformed into columnar

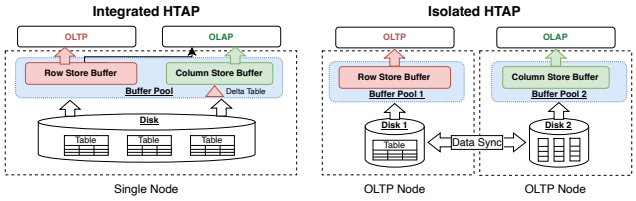

Figure 4: The Architecture of Integrated v.s. Isolated HTAP System

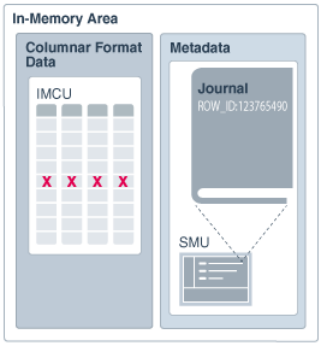

Figure 5: IMCU and SMU in Oracle[10]

format after filtering. These steps introduce additional CPU overhead not captured by I/O alone, and in some cases, account for up to 30% of the total cost of the SeqScan operator.

Therefore, while optimizing for I/O cost may be sufficient in many cases, using full execution cost as the objective yields a more accurate performance evaluation when possible.

1.2 Applied to Other HTAP Systems

We collaborate closely with the GaussDB development team to support their cloud users. As a highly complex system with numerous configuration options, GaussDB is designed to flexibly adapt to diverse application scenarios. To better understand how our approach can benefit such systems, we first examine the architectural categories of HTAP systems. As illustrated in Figure 4, HTAP systems can be broadly classified into two types: Integrated HTAP systems, which co-locate OLTP and OLAP components within the same physical node, and Isolated HTAP systems, which separate them across different nodes, often using dedicated storage engines. GaussDB falls into the integrated category, along with other widely adopted commercial systems such as Oracle Dual[7], SQL Server[8], and PolarDB-IMCI[14], which support OLAP queries by maintaining columnar replicas of hot data. While this architecture ensures efficient data synchronization, it also introduces challenges such as memory contention, column selection, and limited resource isolation. In the following, we demonstrate how our proposed method, T^2, can be effectively applied to integrated HTAP systems to address these issues.

1) **Oracle Dual[7][10].** Oracle's HTAP architecture closely resembles that of GaussDB, with minor differences in data synchronization mechanisms. Based on these differences, we demonstrate how T^2 can be adapted to Oracle.

As illustrated in Figure 5, Oracle maintains transactional consistency in its in-memory column store through a mapping between each In-Memory Column Unit (IMCU) and a Snapshot Metadata Unit (SMU). Each SMU contains a transaction journal that logs modifications. When a row in an IMCU is updated, the system records the row ID in the journal and marks it as stale. Subsequent queries that require the latest version of the row retrieve it from the buffer cache. When the threshold of modified rows stored in the Delta Table or transaction journal is reached, GaussDB-HTAP triggers a rebuild thread. This thread merges each recorded datum from the Delta Table into the columnar data. In contrast, Oracle directly triggers a "Repopulate" command to reload row-based data from disk. This method's data synchronization cost is not proportional to the number of modified rows but to the total volume of data need to be reloaded. However, with minor modifications, our model can still adopt this approach.

Since the volume of data in the table does not increase rapidly within a time window, we assume that the cost of "repopulating" is a constant w_2:

$$\text{Cost}_{\text{sync}}(t) = w_2$$

$$\text{Cost}_{\text{sync}} = \frac{b}{\alpha} \times w_2$$

Similarly, by modeling the cost within a time window, we calculate the total cost of reading and synchronization:

$$\text{Cost}_{\Delta} = \frac{b}{\alpha} w_2 + \nu \left(\frac{w_0 \alpha}{2} + w_1 \right)$$

Deriving with respect to α, and setting $\frac{d\text{Cost}_{\Delta}}{d\alpha} = 0$, we obtain the synchronization threshold that minimizes the cost:

$$\alpha = \sqrt{\frac{2bw_2}{\nu w_0}}$$

Subsequently, this value of α can be incorporated into other formulas as needed. The remaining components, including Column Selection, Memory Allocation, and the Dynamic Reallocation Trigger Algorithm, can be directly implemented using the T^2 method. From this, it is evident that our approach can be seamlessly applied to Oracle.

2) **SQL Server[8].** SQL Server introduces a more complex HTAP architecture, offering greater flexibility but also increasing the number of memory-related factors that need to be managed by optimization algorithms. Nevertheless, the T^2 optimization framework remains applicable and can be adapted to address these challenges in SQL Server. As shown in Figure 6, it offers three storage options: one for the column store and two for the row store (the disk-based row store and the memory-based row store). Similarly, SQL Server faces challenges in selecting appropriate columns for the column store as well as in choosing the right data for the memory-based row store. Moreover, memory allocation becomes even more complex because the Hekaton engine requires reserved memory for the memory-based row store. Despite these complications, the core idea behind our approach remains applicable.

In this context, OLTP performance is highly dependent on both the data managed by the Hekaton engine and the buffer of the classical SQL Server engine. The overall optimization problem can be formulated as follows:

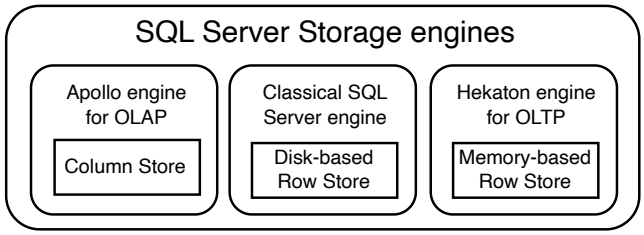

Figure 6: The Storage Architecture of Sql Server

$$\begin{aligned} \text{Minimize} \quad & \text{Cost}_{\mathcal{W}_{\text{ap}}}(C) + \text{Cost}_{\Delta}(C, \mathcal{K}) \\ \text{Subject to} \quad & \mathcal{P}_{\text{OLTP}}(\mathcal{M}_{\text{ClassicalBuffer}}, \mathcal{M}_{\text{Hekaton}}) \geq \theta, \\ & \mathcal{M}_{\text{ClassicalBuffer}} + \mathcal{M}_{\text{Apollo}} + \mathcal{M}_{\text{Hekaton}} \leq \mathcal{M}_{\text{total}}, \\ & W(C) \leq \mathcal{M}_{\text{Apollo}}. \end{aligned}$$

To address this optimization problem, we can adopt the framework described in Section 3.3 with a slight modification: an additional module for Hekaton data selection is introduced to choose beneficial data for the Hekaton engine, thereby accelerating OLTP performance as much as possible. Furthermore, the OLTP performance prediction model must consider two aspects because transactional queries can be executed either by the classical SQL Server engine or by the Hekaton engine. Consequently, the concept of T^2 can be adapted to SQL Server with minor customization.

3) **Polardb-IMCI[14].** Although PolarDB-IMCI is a cloud-native database, our column selection and memory allocation algorithms remain applicable to it. It adheres to the critical design principle of separating the computation and storage layers. The storage layer is powered by a user-space distributed file system called PolarFS. The computation layer comprises multiple nodes: a primary node (RW node) that handles both read and write requests, and several read-only nodes (RO nodes).

To accelerate analytical queries, PolarDB-IMCI supports the construction of in-memory column indexes on the row store of the RO nodes. On these RO nodes, both OLTP and OLAP read-only queries are executed concurrently. Given the limited memory resources, our column selection algorithm can be employed to determine which columns to load into memory. Furthermore, the system maintains both row store buffers and column store buffers, which can be dynamically managed using a T^2 memory allocation algorithm.

Regarding data synchronization, since data updates are performed exclusively on the RW node and propagated via the REDO log, the process for updating data on the RO nodes becomes considerably more complex and necessitates detailed design considerations for the synchronization algorithm.

1.3 Pseudocode for GACC algorithm

We designed a greedy algorithm for column selection, and the pseudocode is shown below. First, an empty set *selected_columns* is initialized, and the available memory is set to $\mathcal{M}_{\text{col}}$. Then, a greedy strategy is applied for column selection.

The set p contains all column combinations and their associated performance gains, denoted as "score", which is calculated as $\text{cost}^{qi}_{\text{row}} - \text{cost}^{qi}_{\text{col}}$. During the greedy selection process, at each step, the column combination with the highest performance gain to memory cost ratio is selected. However, it is necessary to update the memory cost of each column combination during the selection process, as some columns already included in $selected_columns$ do not incur additional memory costs. Thus, lines 9–13 are used to recalculate the memory cost of each column combination. Lines 14–17 select the column combination with the highest benefit-to-weight ratio. After selecting the best column combination, lines 22–25 update the remaining available memory and the *selected_columns* set. By repeating this process, the available memory is gradually consumed until it is fully utilized.

Algorithm 1 Greedy Algorithm for Column Combinations (GACC)

```
1:  Initialize selected_columns ← ∅
2:  Initialize available_cost ← M_col
3:  while available_cost > 0 do
4:      best_ratio ← 0
5:      best_combination ← None
6:      for combination ∈ p do
7:          total_score ← p[combination]['score']
8:          total_cost ← 0
9:          for column ∈ combination do
10:             if column ∉ selected_columns then
11:                 total_cost ← total_cost + w[column]
12:             end if
13:         end for
14:         if total_cost ≤ available_cost and
    total_score/total_cost > best_ratio then
15:             best_ratio ← total_score/total_cost
16:             best_combination ← {(col, w[col]) : col ∈
    combination \ selected_columns}
17:         end if
18:     end for
19:     if best_combination = None then
20:         break
21:     end if
22:     for (column, cost) ∈ best_combination do
23:         Add column to selected_columns
24:         available_cost ← available_cost − cost
25:     end for
26: end while
27: return selected_columns
```

1.4 Supplementing the Optimal Data Synchronization Strategy

We present a detailed derivation of the optimal synchronization threshold α, followed by a robustness analysis under non-uniform assumptions.

We model this problem by selecting a time period $[0, T]$. We assume that data updates are uniformly distributed within this time period, meaning the number of records in the delta table grows linearly at a constant rate of $r = \frac{b}{T}$, where b is the total number of data update records during the time period $[0, T]$. A synchronization is performed whenever the number of records reaches the threshold α. Given the assumption of a constant data growth rate, the time interval between each synchronization is

uniform. Since synchronization is triggered each time the number of records reaches α, there will be $\frac{b}{\alpha}$ synchronization events over the time period $[0, T]$. The cost of a single synchronization operation is $w_2\alpha + w_3$, resulting in a total synchronization cost during the time period $[0, T]$ of:

$$\text{Cost}_{\text{sync}} = \frac{b}{\alpha} \times (w_2\alpha + w_3)$$

Assume that reading events are uniformly distributed within the time period $[0, T]$. There are a total of ν reads in the time period $[0, T]$. Due to the assumption of a constant data growth rate, the time interval between each synchronization trigger is also the same, denoted as $T_\alpha = \frac{\alpha}{r}$. The expected time for each read is

$$E[\text{Cost}_{\text{read}}(t)] = \int_0^{T_\alpha} \text{Cost}_{read}(t)h(t)dt$$

where $h(t)$ is the probability density function of t, which is $\frac{1}{T_\alpha}$ in the case of a uniform distribution. Based on the previous discussion, $N(t) = rt$ and we have:

$$\begin{aligned} E[\text{Cost}_{read}(t)] =& \int_0^{T_\alpha} (w_0 rt + w_1)\frac{1}{T_\alpha}dt = \frac{w_0 r}{T_\alpha}\int_0^{T_\alpha} tdt + \frac{w_1}{T_\alpha}\int_0^{T_\alpha} dt \\ =& \frac{w_0 r}{T_\alpha} \cdot \frac{t^2}{2}\Big|_0^{T_\alpha} + \frac{w_1}{T_\alpha} \cdot t\Big|_0^{T_\alpha} = \frac{w_0 r \frac{\alpha}{r}}{2} + w_1 = \frac{w_0\alpha}{2} + w_1 \end{aligned}$$

During each synchronization cycle of T_α, $\nu\frac{T_\alpha}{T}$ reads will occur. The total read cost during T_α is

$$\text{Cost}_{\text{read}}^{T_\alpha} = \nu\frac{T_\alpha}{T} \times (\frac{w_0\alpha}{2} + w_1)$$

The overall cost, Cost_Δ, is the sum of the synchronization and reading costs in each synchronization cycle, given by $\text{Cost}_\Delta = \text{Cost}_{\text{sync}} + \text{Cost}_{\text{read}}$:

$$\begin{aligned} \text{Cost}_\Delta =& \frac{b}{\alpha}(w_2\alpha + w_3) + \frac{b}{\alpha} \cdot \nu\frac{T_\alpha}{T} \times (\frac{w_0\alpha}{2} + w_1) \\ =& \frac{b}{\alpha}(w_2\alpha + w_3) + \nu(\frac{w_0\alpha}{2} + w_1) \end{aligned} \tag{2}$$

To find the value of α that minimizes Cost_Δ, we need to take the derivative of Cost_Δ with respect to α and set it to zero:

$$\begin{aligned} \frac{d\text{Cost}_\Delta}{d\alpha} =& \frac{d}{d\alpha}(\frac{b}{\alpha}(w_2\alpha + w_3) + \nu(\frac{w_0\alpha}{2} + w_1)) \\ =& \frac{dbw_2}{d\alpha} + \frac{d\frac{bw_3}{\alpha}}{d\alpha} + \frac{d\frac{\nu w_0\alpha}{2}}{d\alpha} + \frac{d\nu w_1}{d\alpha} = -\frac{bw_3}{\alpha^2} + \frac{\nu w_0}{2} = 0 \end{aligned}$$

By solving the equation above, value of α can be obtained:

$$\alpha = \sqrt{\frac{2bw_3}{\nu w_0}} \tag{3}$$

This value of α minimizes the overall synchronization and read cost under the uniform distribution assumption. Under a worst-case scenario where each read incurs the maximum cost (i.e., scanning the full α records), the expected read cost becomes:

$$\Delta\text{Cost}'_{\text{read}} = (w_0\alpha + w_1)$$

To determine the threshold α under this worst-case scenario, we have:

$$\alpha' = \sqrt{\frac{bw_3}{\nu w_0}}$$

Subsequently, the overall cost becomes:

$$\text{Cost}'_\Delta = \frac{b}{\alpha'}(w_2\alpha' + w_3) + \nu(w_0\alpha' + w_1) = bw_2 + 2\sqrt{bw_3\nu w_0} + \nu w_1$$

Thus, the deviation in the overall cost is given by:

$$\Delta\text{Cost}_\Delta = \text{Cost}'_\Delta - \text{Cost}_\Delta = \sqrt{2bw_3\nu w_0}$$

This result shows that even under worst-case conditions, the deviation in total cost is bounded and grows sub-linearly with respect to the input size. Hence, the model remains robust despite the simplified assumption of uniform distribution.

We conduct an experiment to show the effect. As Table 6 shows, we test the synchronization and reading times of delta tables. And compare our method T^2's data synchronization strategy with fixed thresholds as 100,1000,10000,100000. We design two query scheduling modes. The first is a uniform distribution mode, where all queries are sent according to their arrival rate in each interval, resulting in uniformly distributed reads and updates. The second is a skewed distribution mode, where the query sending pattern is intentionally imbalanced to simulate load skew. As shown in results, the skewness here didn't low down T^2's effectiveness very much.

Uniformly Distributed					
Threshold	100	1000	10000	100000	T^2
Sync	6608.97	1101.95	566.63	512.45	637.98
Read	173.73	225.71	780.71	6416.22	267.34
All	6782.7	1327.66	1347.34	6928.67	**905.32**
Skewly Distributed					
Threshold	100	1000	10000	100000	T2
Sync	6519.15	1087.86	528.06	507.41	642.29
Read	178.16	256.25	1039.97	8802.9	315.4
All	6697.31	1344.11	1568.03	9310.31	**957.69**

Table 6: Data Synchronization Threshold Effect on execution time(s).

2 SYMBOLS

To make the reading process more convenient for readers, we provide a symbol table containing the key symbols used in the paper, allowing readers to easily reference their corresponding meanings.

Symbols	Description
$\mathcal{M}_{\text{total}}$	The total memory available for the whole buffer pool.
$\mathcal{M}_{\text{row}}, \mathcal{M}_{\text{col}}$	The memory allocated to the row and column store buffers, respectively.
$W(C)$	The size of the selected column set.
$\mathcal{P}_{OLTP}$	A model or function that predicts the TPS for the current OLTP workload.
$\mathcal{K}$	The data synchronization strategy.
$t_0, t_1, \ldots, t_n$	Equal-sized time windows, where t_i represents the i-th time window.
θ_{t_i},θ'_{t_i}	Required TPS and its forecasted value for the upcoming time window t_i. The forecasted value (θ'_{t_i}) is predicted by the workload predictor.
$\mathcal{W}\text{ap}(t_i), \mathcal{W}'\text{ap}(t_i)$	The OLAP workload for time window t_i, where $\mathcal{W}\text{ap}(t_i)$ is the actual workload and $\mathcal{W}'\text{ap}(t_i)$ is the forecasted value.
C_{t_i}	The chosen column set in time window t_i.
$\text{Cost}_{\mathcal{W}_{\text{ap}}(t_i)}(C_{t_i})$	The OLAP workload cost at time t_i.
$\text{Cost}_{\Delta}(C_{t_i}, \mathcal{K})$	The delta synchronization cost at time t_i.
$\text{Cost}^{t_i}_{\text{switch}}$	The switching cost incurred when triggering dynamic memory allocation at time window t_i.
$F(t_i)$	The memory reallocation strategy that determines whether to trigger reallocation at the start of time window t_i.

Table 7: Symbol Table (Section 2: Problem Definition)

Symbols	Description
$Q_1, Q_2, \ldots, Q_s$	Original queries in the OLAP workload $\mathcal{W}_{\text{ap}}$.
M	The total number of columns in the database.
$C_1, C_2, \ldots, C_M$	The columns in the database.
K	The total number of subqueries in the OLAP workload $\mathcal{W}_{\text{ap}}$.
q_l	The subquery responsible for data scanning and filtering. The original query Q_s may contain more than one subquery.
G_l	The group of columns involved in subquery q_l. $\lvert G_l \rvert$ denotes the number of columns in group G_l.
f_l	The execution frequency of subquery q_l.
$\text{cost}^{q_l}_{\text{row}}$,$\text{cost}^{q_l}_{\text{col}}$	The cost of retrieving data through sequential scan or column scan for subquery q_l, respectively.
x_m	A decision variable indicating whether column C_m is selected.
w_m	The memory usage of column C_m.
z_l	A decision variable indicating whether query q_l is scanning from column storage.
$\text{Cost}_{\text{read}}$,$\text{Cost}_{\text{sync}}$	Costs of reading and synchronizing the delta table, respectively.
$N(t)$	The number of records in the delta table at time t.
w_0, w_1, w_2, w_3	Coefficients for the linear cost models of reading and synchronizing the delta table.
α	The synchronization threshold.
J	The total number of tables in the database.
b_j	The number of UID operations on table j.
v_j	The number of times delta table j is accessed.
Cost^{j}_{Δ}	The total cost associated with reading and synchronizing the delta table of table j.
u_j	A decision variable indicating whether the columns of table j are loaded into memory.
S_j	The set of columns in table j.
$\mathcal{U}(\mathcal{M}_{\text{col}})$	Objective function representing the total cost associated with a given column memory allocation, where smaller values are preferable.
K'	Reduced number of communities after spectral clustering.
λ	Large penelty coefficient.

Table 8: Symbol Table (Section 3: T^2 FOR STATIC WORKLOADS)

Symbols	Description
Q_s	A specific OLAP query template.
$R_t^{(Q_s)}$	The request rate for query template Q_s at time t.
L	The total number of query templates in the workload.
$\mathbf{R_h}$	The sequence of request rates for all query templates up to time h.
γ	The number of future time intervals to predict.
$\hat{R}_{t+\gamma}^{(Q_s)}$	The predicted request rate for query template Q_s at time interval $t+\gamma$.
ΔC_{t_i}	The set of columns that are present at time interval t_i but not at the previous interval t_{i-1}.
$Tables(\Delta C_{t_i})$	A function returning the set of unique tables containing columns in ΔC_{t_i}.
$\text{Cost}_{\text{row}}(T_j)$	The cost for performing a full table scan on table T_j from the row store.

Table 9: Symbol Table (Section 4: T^2 FOR DYNAMIC WORKLOADS)

REFERENCES

[1] 2025. stress-ng. https://github.com/ColinIanKing/stress-ng. Accessed: 2025-03-05.

[2] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. 2015. Time series analysis: forecasting and control. John Wiley & Sons.

[3] Leo Breiman. 2001. Random Forests. Mach. Learn. 45, 1 (2001), 5–32. https://doi.org/10.1023/A:1010933404324

[4] Harris Drucker, Christopher J. C. Burges, Linda Kaufman, Alexander J. Smola, and Vladimir Vapnik. 1996. Support Vector Regression Machines. In Advances in Neural Information Processing Systems 9, NIPS, Denver, CO, USA, December 2-5, 1996, Michael Mozer, Michael I. Jordan, and Thomas Petsche (Eds.). MIT Press, 155–161. http://papers.nips.cc/paper/1238-support-vector-regression-machines

[5] Jerome H. Friedman. 2001. Greedy function approximation: A gradient boosting machine. The Annals of Statistics 29, 5 (2001), 1189 – 1232. https://doi.org/10.1214/aos/1013203451

[6] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. Neural Comput. 9, 8 (1997), 1735–1780. https://doi.org/10.1162/NECO.1997.9.8.1735

[7] Tirthankar Lahiri, Shasank Chavan, Maria Colgan, Dinesh Das, Amit Ganesh, Mike Gleeson, Sanket Hase, Allison Holloway, Jesse Kamp, Teck-Hua Lee, Juan Loaiza, Neil MacNaughton, Vineet Marwah, Niloy Mukherjee, Atrayee Mullick, Sujatha Muthulingam, Vivekanandhan Raja, Marty Roth, Ekrem Soylemez, and Mohamed Zaït. 2015. Oracle Database In-Memory: A dual format in-memory database. In 31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13-17, 2015, Johannes Gehrke, Wolfgang Lehner, Kyuseok Shim, Sang Kyun Cha, and Guy M. Lohman (Eds.). IEEE Computer Society, 1253–1258. https://doi.org/10.1109/ICDE.2015.7113373

[8] Per-Åke Larson, Adrian Birka, Eric N. Hanson, Weiyun Huang, Michal Nowakiewicz, and Vassilis Papadimos. 2015. Real-Time Analytical Processing with SQL Server. Proc. VLDB Endow. 8, 12 (2015), 1740–1751. https://doi.org/10.14778/2824032.2824071

[9] Yuqi Nie, Nam H. Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. 2023. A Time Series is Worth 64 Words: Long-term Forecasting with Transformers. In The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023. OpenReview.net. https://openreview.net/forum?id=Jbdc0vTOcol

[10] Oracle. 2019. In-Memory Column Store Architecture. https://docs.oracle.com/en/database/oracle/oracle-database/19/inmem/in-memory-column-store-architecture.html#GUID-850DDEFB-6B9F-461B-AAF0-DB6DAFAFCBA2 Accessed: 2025-03-02.

[11] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. 1986. Learning representations by back-propagating errors. Nature 323 (1986), 533–536. https://api.semanticscholar.org/CorpusID:205001834

[12] Sean J. Taylor and Benjamin Letham. 2017. Forecasting at Scale. PeerJ Prepr. 5 (2017), e3190. https://doi.org/10.7287/PEERJ.PREPRINTS.3190V1

[13] Thin-Fong Tsuei, Allan Packer, and Keng-Tai Ko. 1997. Database Buffer Size Investigation for OLTP Workloads (Experience Paper). In SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data, May 13-15, 1997, Tucson, Arizona, USA, Joan Peckham (Ed.). ACM Press, 112–122. https://doi.org/10.1145/253260.253279

[14] Jianying Wang, Tongliang Li, Haoze Song, Xinjun Yang, Wenchao Zhou, Feifei Li, Baoyue Yan, Qianqian Wu, Yukun Liang, ChengJun Ying, Yujie Wang, Baokai Chen, Chang Cai, Yubin Ruan, Xiaoyi Weng, Shibin Chen, Liang Yin, Chengzhong Yang, Xin Cai, Hongyan Xing, Nanlong Yu, Xiaofei Chen, Dapeng Huang, and Jianling Sun. 2023. PolarDB-IMCI: A Cloud-Native HTAP Database System at Alibaba. Proc. ACM Manag. Data 1, 2, Article 199 (June 2023), 25 pages. https://doi.org/10.1145/3589785