UniView: A Unified Autonomous Materialized View Management System for Various Databases

Zhenrong Xu, Pengfei Wang
Zhejiang University
Hangzhou, China
{xuzhenrong,wangpf}@zju.edu.cn

Guoze Xue, Qitong Yan
Zhejiang University
Hangzhou, China
{xuegz,qitong.yan}@zju.edu.cn

Shenghao Gong, Yelan Jiang
Zhejiang University
Hangzhou, China
{gongshenghao,jiangyelan}@zju.edu.cn

Yuren Mao, Yunjun Gao
Zhejiang University
Hangzhou, China
{yuren.mao,gaoyj}@zju.edu.cn

Shu Shen, Wei Zhang, Dan Luo
Huawei
Hangzhou, China
{shenshu,zhangwei09,luodan2}@huawei.com

Lu Chen
Zhejiang University
Hangzhou, China
luchen@zju.edu.cn

ABSTRACT

Materialized views (MVs) are critical for improving query performance of database systems, especially in online analytical processing (OLAP) databases. Typically, MVs are maintained by DBAs, which relies on prior knowledge and manual operations. Recently, autonomous solutions are designed for specific databases. However, a data warehouse for OLAP is typically hierarchical, which uses different database engines at different stages. Hence, existing methods have limitations in terms of autonomy and unification to support practical applications.

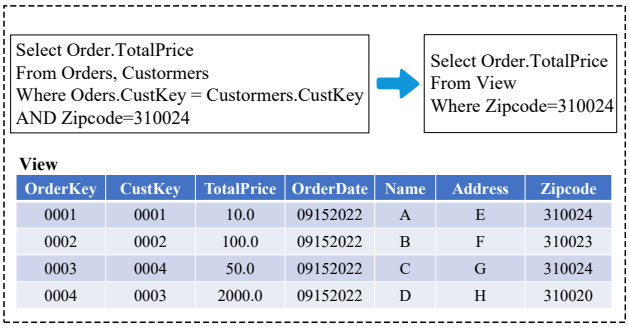
Motivated by these, we develop UniView, a unified autonomous materialized view management system that supports various popular databases, including Spark SQL, PostgreSQL, and ClickHouse. Moreover, we provide a cross-platform web user interface, where users can carry out the process of materialized views and evaluate the optimization performance. In the demonstration, we show that UniView is user-friendly and can achieve superior performance in the practical industry scenarios.

PVLDB Reference Format:
Zhenrong Xu, Pengfei Wang, Guoze Xue, Qitong Yan, Shenghao Gong, Yelan Jiang, Yuren Mao, Yunjun Gao, Shu Shen, Wei Zhang, Dan Luo, and Lu Chen. UniView: A Unified Autonomous Materialized View Management System for Various Databases. PVLDB, 17(12): 4353 - 4356, 2024.
doi:10.14778/3685800.3685873

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/ZJU-DAILY/UniView.

1 INTRODUCTION

Materialized views (MVs) are of critical importance to the query performance of database systems. As shown in Figure 1, we can materialize a view to speed up the query process. In online analytical processing (OLAP) databases, many SQL queries share common subqueries and there are lots of redundant computations among these queries. Materializing views on these subqueries can avoid redundant computation and improve query performance, which is a space-for-time trade-off principle. Therefore, it is vital to select the optimal MVs that can bring the most query performance improvement within a space budget. For example, in Amazon Redshift, automated materialized views are a powerful tool for improving query performance.



OrderKey	CustKey	TotalPrice	OrderDate	Name	Address	Zipcode
0001	0001	10.0	09152022	A	E	310024
0002	0002	100.0	09152022	B	F	310023
0003	0004	50.0	09152022	C	G	310024
0004	0003	2000.0	09152022	D	H	310020

Figure 1: An example of materialized views.

Most existing methods rely on DBAs to generate and maintain MVs [3]. Nevertheless, these methods require prior knowledge and manual operations, which are costly, and thus, cannot efficiently and effectively support large-scale databases. Motivated by this, autonomous MV selection methods are proposed recently. The MV selection process within a space budget can be regarded as a 0-1 integer linear programming (0-1 ILP) problem. Solving 0-1 ILP is too expensive for large workloads since the complexity of the 0-1 ILP approach is $O(2^n)$. To efficiently solve the MV selection, two lines of methods exist. One line of MV selection methods is *heuristics*, such as the greedy strategy [1, 2]. Another line of studies uses *reinforcement learning (RL)* [3, 4, 7, 10] or *graph algorithm* [5] to solve the 0-1 ILP problem faster.

There exist many works[3, 4, 6, 7, 9–11] that leverage ML methods to solve database problems, indicating a growing trend.

However, existing automatic MV selection methods are designed for specific database systems, e.g., DQM [6] designed for Spark SQL, AutoView [3] designed for PostgreSQL, DBMind [11] designed for OpenGauss, and SOFOS [8] designed for knowledge graphs.

Proceedings of the VLDB Endowment, Vol. 17, No. 12 ISSN 2150-8097.
doi:10.14778/3685800.3685873

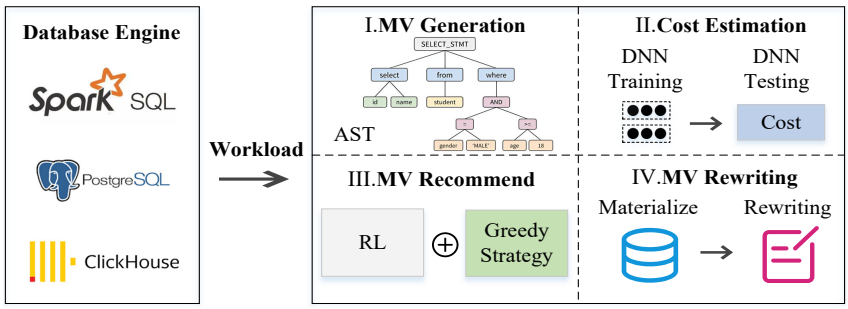


Figure 2: UniView system architecture.

```
def mv_generation(Q):
    V = []
    for q in Q:
        S = parse_one_execution_plan(q)
        U = []
        for s in S:
            if node_type(s) in ('join', 'agg'):
                U.append(s)
        for u in U:
            updated = 0
            for v in V:
                if can_merge(u, v):
                    v = mv_union(u, v)
                    updated = 1
                    break
            if updated == 0:
                V.append(u)
    return V
```

Figure 3: MV generation algorithm.

Their customized design makes them highly coupled and hard to migrate to other databases. In real life applications, different types of database systems are typically used at different stages. For example, a data warehouse hierarchical uses different database engines (e.g., Spark SQL and Clickhouse) at different stages for decision-making. Hence, a unified management system is preferred to support various popular databases while materializing views autonomously.

In this demonstration, we develop UniView, a **uni**fied autonomous materialized **view** management system for various databases. Specifically, UniView consists of four phases: (i) MV Generation aims to parse the queries and generate candidate views; (ii) Cost Estimation utilizes the deep network to estimate the cost of queries and MVs; (iii) MV Recommend aims to recommend the optimal MVs within a space budget based on the cost; and (iv) MV Rewriting aims at rewriting the query using the most appropriate views. UniView supports three different types of database systems, i.e., Spark SQL, PostgreSQL and ClickHouse. We also provide a cross-platform web UI, where users can submit queries and get recommended views to materialize. The web UI can demonstrate the difference in execution performance of queries with/without materialized views so that users are able to understand better how UniView improves the query performance. Moreover, UniView has been deployed in the Huawei Consumer Business Group (CBG) to manage materialized views for query performance improvement. We summarize the contributions as follows:

- We demonstrate UniView, a unified autonomous materialized view management. To the best of our knowledge, UniView is the first autonomous materialized view management supporting various popular databases simultaneously.
- We implement a cross-platform web UI to interact with users and demonstrate the improvements brought by UniView. We have open-sourced UniView, and it is available at GitHub https://github.com/ZJU-DAILY/UniView.
- UniView has been deployed in Huawei CBG to improve query efficiency. Our preliminary results show that UniView is able to reduce query execution time using recommended materialized views, which verifies the effectiveness of UniView in the real-world industry scenario.

The rest of the paper is organized as below. Section 2 presents system overview. Section 3 provides demonstration overview. Section 4 makes conclusions with promising future directions.

2 SYSTEM OVERVIEW

We first offer some preliminaries of UniView, and then, we present the system architecture and a detailed workflow of UniView.

2.1 Preliminaries

We first introduce the query tree to represent a SQL query, based on which, materialized view management is present.

Query Tree. Given a SQL query q, we parse it as a query tree (abstract syntax tree, AST). Each subtree rooted at a node corresponds to a subquery, and each node indicates an operator. All subqueries except the leaves in the query tree can be materialized as views.

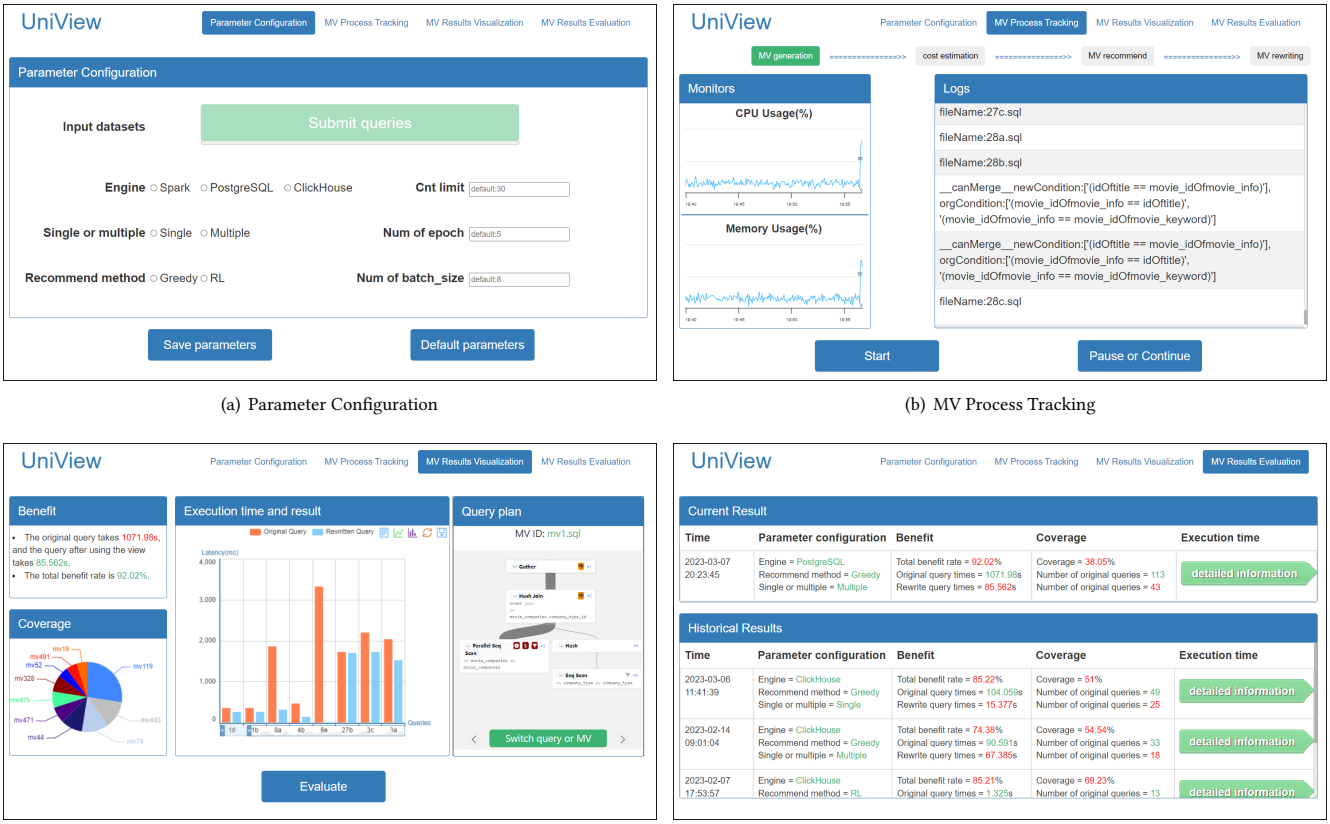
Materialized View Management. Given a query workload Q, there exists a set $\mathcal{V}$ of candidate MVs. It is vital to select a subset $V^* \subseteq \mathcal{V}$ to materialize within a given space budget τ, which can minimize the total execution time of the query workload.

2.2 Workflow

Figure 2 illustrates the overall system architecture of UniView. It is composed of four phases, namely, (i) MV generation, (ii) cost estimation, (iii) MV recommend, and (iv) MV rewriting.

MV Generation aims to find common subqueries for generating candidate MVs $\mathcal{V}$. First of all, we parse all SQL queries in the query workload Q as query trees. Common subqueries are the equivalent subtrees among different query trees of queries. After finding all common subqueries, we are able to generate $\mathcal{V}$. Specifically, we compute the qualities of all common subqueries. The qualities are formulated as the weighted sum of some important factors, e.g., the number of MV that matched the original queries, the size of the table that the MV contains, and the number of predicates. After that, the common subqueries with high qualities will be selected as candidate MVs $\mathcal{V}$. Our UniView supports three popular databases, including Spark SQL, PostgreSQL, and ClickHouse. Figure 3 illustrates the abstract code of MV generation for Spark SQL, PostgreSQL, and ClickHouse. Note that, since execution plan structures and operator types generated by different database engines are different, it is required to customize the analysis of the execution plans for the three databases.

Cost Estimation aims at conducting cost estimation to estimate the benefit, including the execution time and the space cost. The benefit estimation is the difference between the cost of a query and the corresponding rewritten query. We also estimate the space cost (i.e., storage cost) of each MV candidate. We adopt a deep



(a) Parameter Configuration

(b) MV Process Tracking

(c) MV Results Visualization

(d) MV Results Evaluation

Figure 4: UniView visualization.

neural network (DNN) to predict the cost. The DNN takes in two parts of information: i) execution plans of queries and MVs, and ii) metadata. In particular, the metadata includes the schema of input tables (e.g., table names, column names, and column types) and the statistics of input tables (e.g., the number of tables and the number of columns).// The DNN is trained in a regressive manner and we use some actual cost as ground truth to train our model. Specifically, we execute several (e.g., 100) rewrite queries to get the actual execution time cost. And we materialize several MVs to get the actual space cost. After training, we predict the cost of all rewrite queries and all MVs.

MV Recommend aims to select a subset $V^* \subseteq \mathcal{V}$ to materialize when given a space budget τ. Our current implementation develops two selection strategies: *reinforcement learning (RL)* and *greedy algorithm*. Based on the results of cost estimation, we can get the benefit and the storage cost of each view. As mentioned before, the MV selection process within a space budget can be regarded as a 0-1 integer linear programming (0-1 ILP) problem. RL is an efficient method to solve 0-1 ILP, which considers the global optimal solution and needs a training process. Let $e_{ij} \in \{0, 1\}$ denotes whether we use $v_j \in \mathcal{V}$ to rewrite $q_i \in Q$, $x_i \in \{0, 1\}$ represents whether v_j will be materialized, and τ is the space budget. We optimize: $\arg\max_{e_{ij}} \sum_{i=1}^{|Q|} \mathcal{B}(q_i, V_i)$, s.t., $\left(\sum_{j=1}^{|\mathcal{V}|} x_j |v_j|\right) \leq \tau$, where $\mathcal{B}$ denotes the benefit, $V_i = \{v_j \mid e_{ij} = 1, j \in [1, |\mathcal{V}|]\}, \forall i \in [1, |Q|]$, and $x_j = \max\{e_{ij} \mid i \in [1, |Q|]\}, \forall j \in [1, |\mathcal{V}|]$. The greedy algorithm iteratively selects a view with largest benefit considering the local optimal solution, and hence, its execution time is very short.

MV Rewriting aims at rewriting SQL queries using the recommended MVs during the optimization phase of the execution plan. Execution plans are represented as query trees. We match the execution plan of the query and MVs in three parts: i) input (i.e., tables they used); ii) intermediate processing (e.g., the conditions of join and filter); and iii) output (e.g., projection and aggregation). If the matching is successful, we will rewrite this part of the execution plan with the execution plan of the MV.

3 DEMONSTRATION OVERVIEW

UniView is built as a cross-platform web-based application, where users can submit their queries and get recommended views to materialize. Figure 4 shows the main interfaces of UniView, including parameter configuration, MV process tracking, MV results visualization, and MV result evaluation.

Parameter Configuration. First of all, a user needs to configure parameters and submit queries before materializing views. As depicted in Figure 4(a), a user can submit their queries by clicking

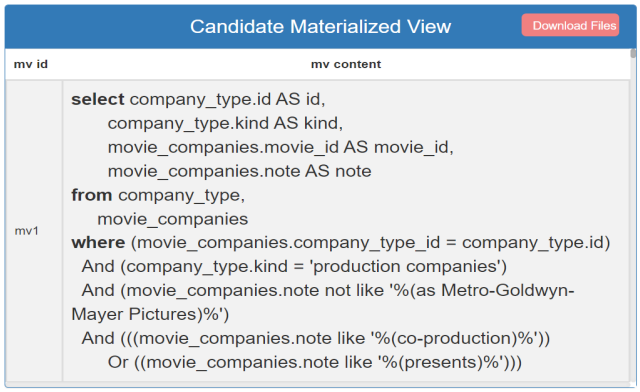


Figure 5: An example of candidate materialized view.

the "submit queries" button. In addition, the users also need to input some important hyper-parameters, e.g., selecting a database engine and a suitable recommend method. UniView provides a detailed explanation of parameters and their default values to simplify parameter configuration. Since the target group of UniView may include both domain experts and ordinary users, UniView provides two options for the parameter configuration: (i) domain experts can configure all parameters manually; (ii) ordinary users only need to select the engine of the database, and other parameters can use the default values.

MV Process Tracking. After the parameters are configured, the user can click the "Start" button to start materialized view management. Note that, the user can click the "Pause or Continue" button to pause the process. The process is dynamic and purely automatic, which includes four pipelined modules as detailed in Section 2, i.e., MV generation, cost estimation, MV recommend, and MV rewriting. As shown in Figure 4(b), the detailed information is reported as logs in real-time. The progress of the current execution is displayed on the progress bar. After each stage, the user can obtain intermediate results and download them, e.g., candidate materialized view. Figure 5 provides an example of candidate materialized view, which can be downloaded for further analysis. At the same time, the user can monitor the real-time changes in CPU and memory along with the MV process. Finally, the user will get recommended views to materialize when four modules have been executed.

MV Results Visualization. This result visualization can demonstrate the difference in execution performance of queries with/without materialized views, so that users are able to understand better how UniView improves the query performance. Queries will be rewritten by the recommended MVs. We present the results in four aspects (as depicted in Figure 4(c)) to help users understand the performance. "Benefit" denotes the query time reduction ratio brought by the recommended MVs and "Coverage" represents the proportion of queries that can be rewritten by the recommended MVs. "Execution time and result" show the execution performance of queries with/without materialized views. We visualize the query plan of the query and its corresponding MV in "Query plan" using pev2[1]. Users can see the overall performance brought by MVs, as well as the find-grained impact of each recommended MV. Besides, we have deployed UniView in Huawei CBG and conducted the preliminary experimental evaluation. The results show that UniView can reduce query time by 85.22% with a coverage of 51.02%.

MV Results Evaluation. We also provide "MV Results Evaluation" page (as depicted in Figure 4(d)) to better manage MVs. On this page, we will demonstrate the most recent view recommendation results and past view recommendation results. On the one hand, for the same query workload, users can easily know the performance of the MVs under different parameter configurations to choose the best parameter configuration. On the other hand, queries sometimes change little in some scenarios, hence, users can reuse existing MVs without any computational cost to improve query performance.

[1]https://github.com/dalibo/pev2

4 CONCLUSION

In this demonstration, we present a unified autonomous materialized view management system, termed as UniView. UniView is able to provide views with high quality and help users to speed up the query execution time. Compared with existing approaches, UniView is more practical as it supports automatic MV management on multiple database engines (including Spark SQL, PostgreSQL, and ClickHouse). We also provide a cross-platform web UI, where users can submit queries and get recommended views to materialize. UniView has been deployed in the Huawei Consumer Business Group (CBG) to manage materialized views for query performance improvement. In the future, it is interesting to enhance the incremental materialized views using promising AI technology.

ACKNOWLEDGMENTS

This work was supported in part by the NSFC under Grants No. (62102351, 62025206, U23A20296). Lu Chen is the corresponding author of the work.

REFERENCES

[1] Rafi Ahmed, Randall Bello, Andrew Witkowski, and Praveen Kumar. 2020. Automated generation of materialized views in oracle. *PVLDB* 13, 12 (2020), 3046–3058.
[2] Yue Han, Chengliang Chai, Jiabin Liu, Guoliang Li, Chuangxian Wei, and Chaoqun Zhan. 2023. Dynamic Materialized View Management using Graph Neural Network. In *ICDE*.
[3] Yue Han, Guoliang Li, Haitao Yuan, and Ji Sun. 2021. An autonomous materialized view management system with deep reinforcement learning. In *ICDE*. 2159–2164.
[4] Yue Han, Guoliang Li, Haitao Yuan, and Ji Sun. 2022. AutoView: An Autonomous Materialized View Management System with Encoder-Reducer. *TKDE* (2022).
[5] Alekh Jindal, Konstantinos Karanasos, Sriram Rao, and Hiren Patel. 2018. Selecting subexpressions to materialize at datacenter scale. *PVLDB* 11, 7 (2018), 800–812.
[6] Xi Liang, Aaron J Elmore, and Sanjay Krishnan. 2019. Opportunistic view materialization with deep reinforcement learning. *arXiv preprint arXiv:1903.01363* (2019).
[7] R Malinga Perera, Bastian Oetomo, Benjamin IP Rubinstein, and Renata Borovica-Gajic. 2022. HMAB: self-driving hierarchy of bandits for integrated physical database design tuning. *PVLDB* 16, 2 (2022), 216–229.
[8] Georgia Troullinou, Haridimos Kondylakis, Matteo Lissandrini, and Davide Mottin. 2021. SOFOS: demonstrating the challenges of materialized view selection on knowledge graphs. In *SIGMOD*. 2789–2793.
[9] Sai Wu, Ying Li, Haoqi Zhu, Junbo Zhao, and Gang Chen. 2022. Dynamic Index Construction with Deep Reinforcement Learning. *Data Science and Engineering* 7, 2 (2022), 87–101.
[10] Haitao Yuan, Guoliang Li, Ling Feng, Ji Sun, and Yue Han. 2020. Automatic view generation with deep learning and reinforcement learning. In *ICDE*. 1501–1512.
[11] Xuanhe Zhou, Lianyuan Jin, Ji Sun, Xinyang Zhao, Xiang Yu, Jianhua Feng, Shifu Li, Tianqing Wang, Kun Li, and Luyang Liu. 2021. Dbmind: A self-driving platform in opengauss. *PVLDB* 14, 12 (2021), 2743–2746.