DeFiHap: Detecting and Fixing HiveQL Anti-Patterns

Yuetian Mao, Shuai Yuan, Nan Cui, Tianjiao Du, Beijun Shen, Yuting Chen
Shanghai Jiao Tong University, Shanghai, China
{mytkeroro,yssjtu,cuinan,tjsoulshe,bjshen,chenyt}@sjtu.edu.cn

ABSTRACT

The emergence of Hive greatly facilitates the management of massive data stored in various places. Meanwhile, data scientists face challenges during HiveQL programming – they may not use correct and/or efficient HiveQL statements in their programs; developers may also introduce anti-patterns indeliberately into HiveQL programs, leading to poor performance, low maintainability, and/or program crashes. This paper presents an empirical study on HiveQL programming, in which 38 HiveQL anti-patterns are revealed. We then design and implement DeFiHap, the first tool for automatically detecting and fixing HiveQL anti-patterns. DeFiHap detects HiveQL anti-patterns via analyzing the abstract syntax trees of HiveQL statements and Hive configurations, and generates fix suggestions by rule-based rewriting and performance tuning techniques. The experimental results show that DeFiHap is effective. In particular, DeFiHap detects 25 anti-patterns and generates fix suggestions for 17 of them.

PVLDB Reference Format:
Yuetian Mao, Shuai Yuan, Nan Cui, Tianjiao Du, Beijun Shen, Yuting Chen. DeFiHap: Detecting and Fixing HiveQL Anti-Patterns. PVLDB, 14(12): 2671-2674, 2021.
doi:10.14778/3476311.3476316

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/ChrisCN97/DeFiHap.

1 INTRODUCTION

Hive, a distributed and parallel processing framework based on Hadoop, is becoming increasingly popular for storing and accessing big data in recent years [9]. It enables developers to perform MapReduce tasks by writing HiveQL (a SQL-like Hive query language) programs.

HiveQL programming, however, is non-trivial since applications may suffer from anti-patterns [6]. An anti-pattern (AP) refers to a design pattern that is common in practice but is inefficient [4]. APs usually lead to HiveQL programs' poor performance, low maintainability, and/or crashes, as Figure 1 shows.

Though AP has been intensively studied [4], how to detect and fix HiveQL APs is still a challenging problem. First, little research has ever identified and classified APs in HiveQL programs systematically. Thus it is necessary to investigate HiveQL APs and their fix solutions. Second, HiveQL is similar to SQL [3, 7]. Thus HiveQL

Proceedings of the VLDB Endowment, Vol. 14, No. 12 ISSN 2150-8097.
doi:10.14778/3476311.3476316

```
SELECT SmallTable.city, BigTable.name
FROM BigTable JOIN SmallTable On BigTable.id = SmallTable.id
```

Figure 1: A HiveQL query which causes low performance due to a "Large Table on the Left" anti-pattern.

APs can be revealed through parsing HiveQL statements using a SQL parser followed by detecting SQL APs. Even though it is promising, core HiveQL APs are highly related to the MapReduce framework and the unique metadata of Hive. Existing solutions to detecting SQL APs are not suitable for detecting HiveQL APs.

Furthermore, tools are needed for facilitating the development of big data and distributed applications against APs, since scientists who are experts in other domains may not be familiar with these APs [1]. To the best of our knowledge, there exists little research on APs in HiveQL (and other SQL-like languages) queries on distributed clusters.

To tackle these challenges, we conduct an empirical study on HiveQL programming, in which 38 HiveQL APs are revealed. We then develop DeFiHap, the first tool for automatically **de**tecting and **fi**xing **H**iveQL **AP**s. DeFiHap employs a rule-based method to detect APs and generate fix suggestions, leveraging not only the HiveQL statements but also Hive metadata and configurations. In addition, it optimizes the performance of join queries by recommending the number of reducers via a multi-layer perceptron.

The experimental results show that DeFiHap is effective. DeFiHap detects 25 HiveQL APs and generates fix suggestions for 17 of them. In addition, for AP detection, DeFiHap achieves a precision of 100% and a recall of 96.88%; for AP fixing, it achieves a precision of 92.11%.

2 AN EMPIRICAL STUDY

We conduct an empirical study to identify real-world HiveQL APs by following three steps:

1) Data Collection. We search on StackExchange [2] using keywords such as "Hive SQL anti-pattern", "Hive SQL optimization", "Hive SQL bug", etc. Each retrieved post contains one or more keywords in its title, description, answers, and/or comments. We filter out the posts having less than 3 votes or without accepted answers. After this step, we get 458 high-quality posts in total.

2) Data Analysis. We manually analyze 458 posts by taking an open coding procedure [8]. All posts are labeled independently, and the conflicts are resolved at expert meetings. Posts not related to HiveQL APs are then marked as false positives and removed. We analyze the descriptions, answers, and comments of the 312 remaining posts and record their root causes and solutions.

3) APs Identification. Finally, we reveal 38 major HiveQL APs, which can be further categorized into "Statement Anti-pattern" (S-AP) and "Configuration Anti-pattern" (C-AP). Among them, 25 APs can be detected or fixed by DeFiHap, via a rule-based approach or

Table 1: List of HiveQL anti-patterns detected and fixed by DeFiHap

Category	Anti-Pattern Name	Description	Impact*	Detection	Fix
Statement AP (S-AP)	Large Table on the Left	Putting table with more records on the left of JOIN.	P	rule-based	rule-based
	Greedy Selection	Using SELECT * which could retrieve redundant result.	P, M	rule-based	rule-based
	Too Many JOINs	Using more than one JOIN operation.	P	rule-based	-
	Misusing HAVING	Using HAVING without GROUP BY.	P	rule-based	rule-based
	Misusing INTERVAL	Combining INTERVAL and DATE_SUB( ) for date query.	E	rule-based	rule-based
	SELECT Inconsistent with GROUP BY	Missing selected columns after GROUP BY.	E	rule-based	rule-based
	Calculation in Predicate	Calculating in predicates after ON or WHERE.	P	rule-based	-
	Calling Functions in Predicate	Calling functions in predicates after ON or WHERE.	p	rule-based	-
	No Group By	Using aggregation functions without GROUP BY.	E	rule-based	-
	Using ORDER BY	Using ORDER BY instead of SORT BY.	P	rule-based	-
	JOIN in Subquery	Using JOIN in the sub-query.	P	rule-based	-
	Creating Duplicate Table	Creating a table having the same column properties as another table in the database.	P, M	rule-based	-
	Querying without Partition	Querying on a partitioned table without using partition filter.	P	rule-based	rule-based
	Data Skew	Querying on a dataset with a non-uniform distribution.	P	rule-based	-
Configuration AP (C-AP)	Inappropriate Number of Reducers	Setting too many or too few reducers for a JOIN operation.	P	tuning	tuning
	Disabled Column Pruner	Not enabling the column pruner configuration item.	P	rule-based	rule-based
	Disabled Partition Pruner	Not enabling the partition pruner configuration item.	P	rule-based	rule-based
	Disabled Output Compression	Not enabling the output compression configuration item.	P	rule-based	rule-based
	Disabled Parallelization	Not enabling the parallelization configuration item.	P	rule-based	rule-based
	Disabled Cost based Optimizer	Not enabling the cost based optimizer (CBO) configuration item.	P	rule-based	rule-based
	ExecutorService Rejection	Task is rejected by executorService.	E	rule-based	rule-based
	Inserting without Dynamic Partition	Inserting the partition table without setting dynamic partition.	E	rule-based	rule-based
	Disabling Partial Aggregation	Not enabling the function of partial aggregation on the map side.	P	rule-based	rule-based
	Disabling Map Join	Not enabling Map Join when small tables join large tables.	P	rule-based	rule-based
	Disabling Small File Merging	Not enabling the function of automatically merging small files.	P	rule-based	rule-based

* **"P" is for poor performance, "M" for low maintainability, and "E" for program error.**

performance tuning techniques, as Table 1 shows. The complete list and their examples are publicly available.

3 SYSTEM DESIGN

We develop DeFiHap for automatically detecting and fixing APs in HiveQL applications. Application developers can leverage DeFiHap to create efficient, maintainable, and accurate queries in big data applications. The architecture of DeFiHap is shown in Figure 2. It detects HiveQL APs; after that, it generates fixes by combining rule-based and performance tuning techniques.

3.1 Detecting HiveQL Anti-Patterns

Instead of employing machine learning methods which require intensive training [5], DeFiHap employs a rule-based detection method through analyzing HiveQL statements, data, Hive configurations, and other metadata.

AST Parsing. DeFiHap parses HiveQL statements into abstract syntax trees (ASTs), aiming at extracting their key elements and structure, and facilitating follow-up AP detection and fix activities. Moreover, the tree-structures allow rules to be recursively represented and improve the extensibility of the tool. We employ Antlr4[1] to implement the AST parser since Antlr4 is a powerful parser generator and is also used by Hive for parsing statements.

Rule-based AP detection. DeFiHap's AP checker implements a static analysis of ASTs, HiveQL metadata, and the data in the database, aiming at searching for the latent *statement APs*. While traversing an AST, it triggers detection rules on nodes. The detection rules range from simple pattern-matching functions in regular expressions to complex functions leveraging Hive data and metadata. Similarly, the AP checker conducts rule-based checks on each Hive configuration obtained via a "set" command, and reports the detected *configuration APs*.

Example: Let the HiveQL statement in Figure 1 be chosen as an example. In this example, the table with more records named BigTable is placed on the left side of "JOIN", which leads to a "Large Table on the Left" AP. DeFiHap builds and traverses the AST of this statement. On the JOIN node, the detection rule is triggered: DeFiHap obtains the two table names and queries the Hive metadata tables (TBLS, PARTITIONS, PARTITION_PARAMS, TABLE_PARAMS). Since the records of the left table are more than those of the right table, DeFiHap reports a "Large Table on the Left" AP.

3.2 Fixing Statement Anti-Patterns

DeFiHap uses an S-AP fix engine to fix the detected statement APs through rewriting a given HiveQL statement. Instead of generating the target statement from scratch, the S-AP fix engine constructs statement templates in order to revise a small part of the elements via a sequence editor. Specifically, the S-AP fix engine first constructs a template for the statement under revision. It then traverses the AST and uses the fix rules to edit elements in the template.

DeFiHap defines four editing operations for each element: *DEL*, *KEEP*, *SWAP*, and *ADD*. The S-AP fix engine processes the original statement element-by-element. It copies relevant elements (tagged

[1]https://github.com/antlr/antlr4

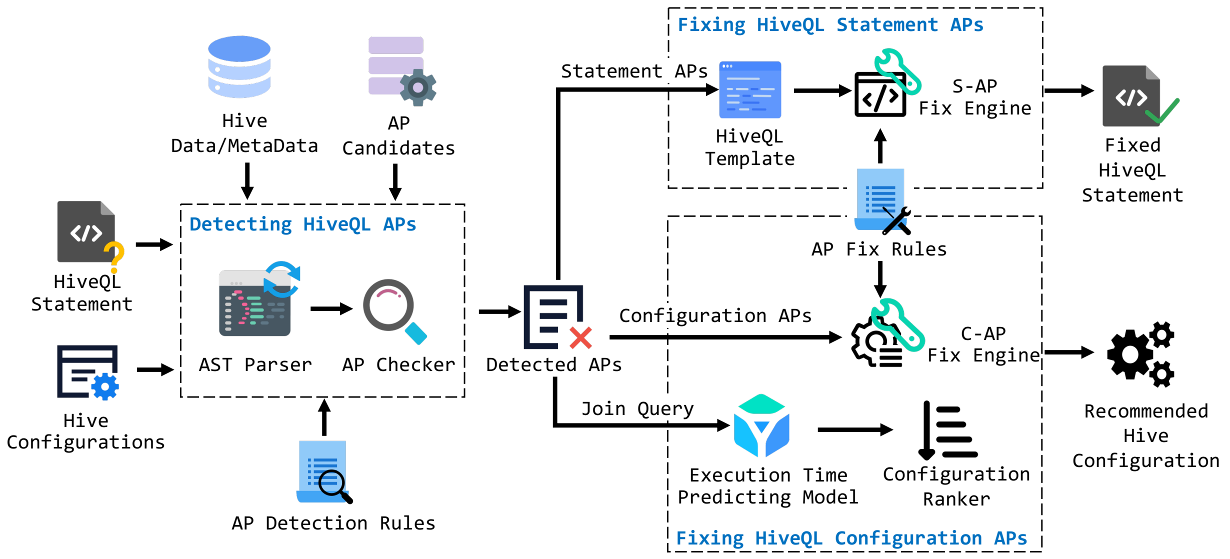
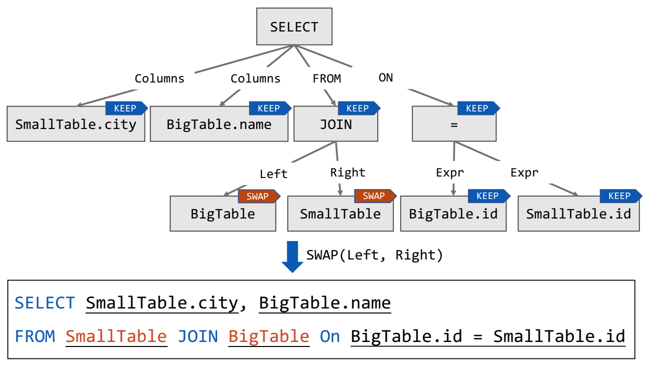


Figure 2: An overview of DeFiHap.

as *KEEP*), removes the irrelevant elements (tagged as *DEL*), and exchanges two elements (tagged as *SWAP*). It can also *ADD* elements, *i.e.*, inserting a span. We implement the template and the S-AP fix engine using StringTemplate4[2], a Java template library.

Example: Given the HiveQL statement in Figure 1, DeFiHap identifies the fault elements in the AST, *i.e.*, "BigTable" and "SmallTable". During traversing the AST, DeFiHap invokes a fix rule to swap these two table names and fills the elements in the slots of the template. Thus DeFiHap generates the fixed HiveQL statement, as Figure 3 shows.



```
SELECT SmallTable.city, BigTable.name
FROM SmallTable JOIN BigTable On BigTable.id = SmallTable.id
```

Figure 3: A fixed HiveQL statement generated via a template-based rewriting technique.

3.3 Fixing Configuration Anti-Patterns

Given a join query, DeFiHap does not only detect and fix its statement APs, but also tune the Hive configuration to optimize the query's performance on the Hadoop MapReduce framework.

We observe that the performance of a join query heavily depends on the number of reducers. As Figure 4 shows, the execution time can be improved by up to 57.8% by setting the "mapred.reduce.tasks" parameter of a join query properly. Intuitively, performance tuning can be conducted – DeFiHap employs a machine learning based approach to search for and recommend reducer settings for different join queries.

[2]https://github.com/antlr/stringtemplate4

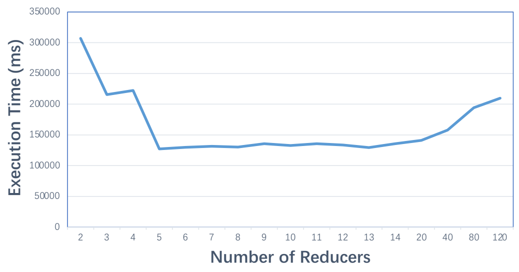
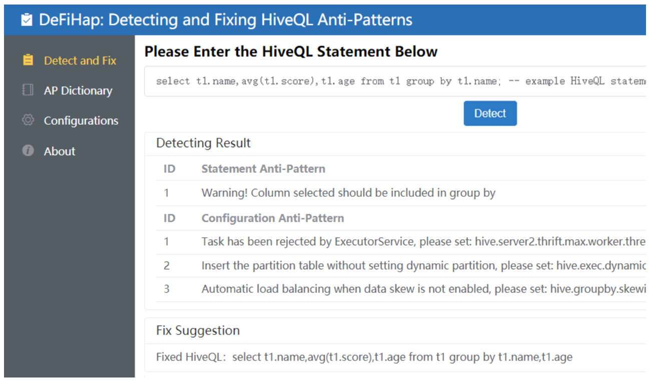


Figure 4: A comparison of the execution time of a join query with different reducer numbers.

Model selection and training. We first conduct experiments to identify the parameters that affect the reduce process, including the numbers of records in each table, different join key values in each table, and reducers. Next, we train multiple machine learning models (multi-layer perceptron (MLP), support vector regression (SVR), and decision tree (DT)) with these parameters to predict the execution time of join queries. We obtain the training data by executing join queries with different table sizes and reducers and collecting the execution time from the Hive log in the cluster. Indeed, the testing results indicate that MLP is the most effective model in this study.

Configuration recommendation. DeFiHap employs the trained MLP model, and recommends the reducer number w.r.t. a given join query by following three steps: (1) obtain the join table names and join key from the AST, and query the number of table records and join key values from Hive and its metadata; (2) predict the query execution time using the MLP with the parameter values



Please Enter the HiveQL Statement Below

```
select t1.name,avg(t1.score),t1.age from t1 group by t1.name; -- example HiveQL statem
```

Detecting Result

ID	Statement Anti-Pattern
1	Warning! Column selected should be included in group by

ID	Configuration Anti-Pattern
1	Task has been rejected by ExecutorService, please set: hive.server2.thrift.max.worker.thre
2	Insert the partition table without setting dynamic partition, please set: hive.exec.dynamic
3	Automatic load balancing when data skew is not enabled, please set: hive.groupby.skewi

Fix Suggestion

Fixed HiveQL: select t1.name,avg(t1.score),t1.age from t1 group by t1.name,t1.age

Figure 5: A snapshot of DeFiHap.

and reducer numbers; (3) rank the reducer numbers in ascending order of the predicted execution time and recommend the most efficient one.

Besides, for simple configuration APs, such as "Disabled Column Pruner", DeFiHap's C-AP fix engine suggests to enable the column pruner with a general configuration recommendation. Rather than using a machine learning model to fix some APs (such as the "Inappropriate Number of Reducers" AP), DeFiHap allows fixes of these APs to be manually collected and applied.

4 DEMONSTRATION AND EVALUATION

We have implemented DeFiHap as a web system. The snapshot of DeFiHap UI is shown in Figure 5. Using a web interface, the user first enters a HiveQL statement for diagnosis. As section 3 explains, DeFiHap parses the statement into an AST and checks it by employing all of the AP detection rules. If statement or/and configuration APs are detected, DeFiHap fixes the faulty statement or/and suggests desired configurations. In addition, DeFiHap recommends the reducer number for the given join query by taking a machine learning based approach.

DeFiHap can be used in two main scenarios.

Scenario 1: *Detecting and fixing a HiveQL statement AP.* When a user enters a HiveQL statement suffering from some APs, DeFiHap invokes the detection and fix rules sequentially. A list of the detected APs and their fix suggestions are returned. For further understanding these APs, users can learn their descriptions, impacts, and code examples in the "AP dictionary" panel.

Scenario 2: *Hive configuration tuning.* On receiving an input join query, DeFiHap automatically tunes the reducer number to optimize its performance on the Hadoop MapReduce framework. DeFiHap recommends the number of reducers, and the user then decides whether the original configuration needs to be adjusted.

Evaluation. We evaluate DeFiHap on a variety of real-world HiveQL statements to quantify its capabilities of processing HiveQL APs. Three widely-used metrics are selected: precision, recall, and F1.

We collect 110 non-duplicated HiveQL statements from Stack Overflow. Among them, 67% are faulty statements and their APs are manually labeled. These statements correspond to all statement APs and their user fixes are extracted from the post descriptions or comments. The dataset also contains 14 different join queries for evaluating the technique of recommending reducer settings. We build Hive tables and load synthetic data for each statement.

DeFiHap runs on Hive 2.3.4 with default configurations in a 3-node Hadoop cluster, and the Hive metadata is stored in MySQL. The MLP model for recommending the reducer number is trained on 3640 Hive logs of 91 different join queries with different reducer numbers.

Result analysis. We run DeFiHap against 110 HiveQL statements diversify the Hive configurations for covering all configuration APs. We manually check the detection results and compare the suggestions with user fixes.

Table 2: Results

Function	Precision	Recall	F1
Detecting HiveQL APs	100%	96.88%	98.42%
Fixing HiveQL APs	92.11%	-	-
Recommending the reducer number	78.57%	-	-

As Table 2 shows, DeFiHap achieves promising results – its AP detection reaches a precision of 100%, a recall of 96.88%, and an F1 score of 98.42%; its AP fix reaches a precision of 92.11%; its recommendation of reducer numbers reaches a precision of 78.57%. The main reason for its effectiveness is that DeFiHap applies goal-driven detecting and fixing strategy on ASTs of HiveQL statements, which restricts the scope of a rule to a limited number of elements of the objective statement. This design mitigates the mutual impacts of different rules.

However, a few statements still cannot be fixed correctly. After a manual analysis, we find that the template-based rewrite technique taken by DeFiHap is less effective in fixing APs in complex statements. Besides, though the MLP model can correctly reflect the trends of execution time as reducer number changes, for some join statements whose execution time fluctuates repeatedly in a small range, MLP cannot predict the extreme point well. It can be one of our future work in employing other machine learning models to enhance the capabilities of DeFiHap.

ACKNOWLEDGMENTS

This work was supported by the National Natural Science Foundation of China (Grant No. 62032004). Beijun Shen is the corresponding author for this paper.

REFERENCES

[1] 2018. https://www.stitchdata.com/resources/the-state-of-data-science/.
[2] 2020. https://data.stackexchange.com/stackoverflow/query/new.
[3] P. Dintyala, A. Narechania, and J. Arulraj. 2020. SQLCheck: automated detection and diagnosis of SQL anti-patterns. In *SIGMOD*. 2331–2345.
[4] M. H. Dodani. 2006. Patterns of Anti-Patterns. *J. Object Technol.* 5, 6 (2006), 29–33.
[5] Shizhe Fu and Beijun Shen. 2015. Code bad smell detection through evolutionary data mining. In *International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE Computer Society, 41–49.
[6] A. Holmes. 2012. Big Data Patterns. In *Hadoop in practice*. Manning, 194–249.
[7] C. Nagy and A. Cleve. 2017. A static code smell detector for SQL queries embedded in Java code. In *SCAM*. 147–152.
[8] C. B. Seaman. 1999. Qualitative methods in empirical studies of software engineering. *IEEE Transactions on Software Engineering* 25, 4 (1999), 557–572.
[9] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Antony, H. Liu, and R. Murthy. 2010. Hive-a petabyte scale data warehouse using hadoop. In *IEEE ICDE*. 996–1005.