DHive: Query Execution Performance Analysis via Dataflow in Apache Hive

Chaozu Zhang
Department of Computer Science and Engineering, Southern University of Science and Technology
zhangcz2021@mail.sustech.edu.cn

Qiaomu Shen
Research Institute of Trustworthy Autonomous Systems, Southern University of Science and Technology
shenqm@sustech.edu.cn

Bo Tang*
Department of Computer Science and Engineering, Southern University of Science and Technology
tangb3@sustech.edu.cn

ABSTRACT

Nowadays, Apache Hive has been widely used for large-scale data analysis applications in many organizations. Various visual analytical tools are developed to help Hive users quickly analyze the query execution process and identify the performance bottleneck of executed queries. However, existing tools mostly focus on showing the time usage of query sub-components (jobs and operators) but fail to provide enough evidence to analyze the root reasons for the slow execution progress. To tackle this problem, we develop a visual analytical system DHive to visualize and analyze the query execution progress via dataflow analysis. DHive shows the dataflow during query execution at multiple levels: query level, job level and task level, which enable users to identify the key jobs/tasks and explain their time usage by linking them to the auxiliary information such as the system configuration and hardware status. We demonstrate the effectiveness of DHive by two cases in a production cluster. DHive is open-source at https://github.com/DBGroup-SUSTech/DHive.git.

PVLDB Reference Format:
Chaozu Zhang, Qiaomu Shen, Bo Tang. DHive: Query Execution Performance Analysis via Dataflow in Apache Hive. PVLDB, 16(12): 3998 - 4001, 2023.
doi:10.14778/3611540.3611605

1 INTRODUCTION

Large-scale data analytical system such as Apache Hive has been widely used in both industry and academia. With the rapid increase of data scale, debugging and optimizing the queries become the daily work of engineers and users. The questions such as "where does time go?" and "what is the performance bottleneck?" are frequently asked by them. However, answering these questions is difficult even for the most experienced engineers due to the inherent complexity of the query execution in Apache Hive. It is essential for users to understand the query execution process before taking action to improve execution performance in many cases. We briefly introduce two of them as follows:

Execution performance understanding: The performance of query execution is unpredictable because it is affected by many factors, such as the data layout, resource contention, and query execution strategy. For example, processing the data from remote machines incurs expensive data movement overhead. Thus, understanding the overhead of data movement overhead in the query execution time is vital to improve the query execution performance by exploiting the data locality.

Query execution comparison: Comparison is crucial for performance analysis. For example, comparing the query execution progresses of the same benchmark query on different versions of Hive will assist the developers to verify the effectiveness of the proposed optimizations. Hence, it will help them determine which optimization techniques should be taken into account in the next Hive release version.

However, there are two major challenges to analyze the time consumption in Apache Hive query execution. **(C1)** *Complex execution process.* A query execution can be affected by the query's inner logic, optimization strategies and executing environment, which requires users to analyze the query execution from multiple aspects. **(C2)** *Massive tasks.* For large-scale input data, many atomic tasks (i.e., the basic execution unit to process a sub-set of input data) are spawned from Map/Reduce jobs. For example, there are in total of 9743 tasks from Query 54 in TPC-DS, with 100GB input data in our cluster. Identifying and explaining the key tasks during query execution is obviously difficult.

In this demo, we present DHive, a visual analytical system for post-analysis of Hive query execution to tackle these challenges. We handle challenge C1 by visualizing the execution progress at multiple levels of detail: query level, job level, and task level. And we trace the processing details by dataflow visualization, which records the information about data size and the data processing speed. Auxiliary information is also jointly shown to help users understand the execution time usage. DHive addresses the challenge **C2** by providing a set of coordinated visualizations and rich interactions, which enables users to efficiently narrow down to the jobs/tasks of interest. Specifically, DHive includes the following components:

- Query logic view: An interactive timeline-based visualization to show the time usage and the relationship of jobs.
- Dataflow overview: A river-based visualization showing the dataflow of the whole query and its jobs.
- Processing details view: A timeline-based visualization tracing the execution of individual tasks, visualizing the speed of data input and computing, to help users explore and reason the low-level problems during the query execution.

*Dr. Bo Tang is the corresponding author.
This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit https://creativecommons.org/licenses/by-nc-nd/4.0/ to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.
Proceedings of the VLDB Endowment, Vol. 16, No. 12 ISSN 2150-8097.
doi:10.14778/3611540.3611605

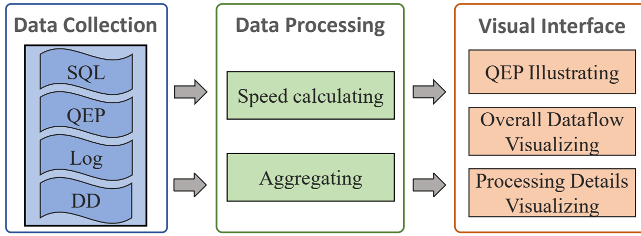

Figure 1: System architecture of DHive

2 RELATED WORK

Several performance analysis tools have been developed for big data systems [1, 3–6, 9]. However, using these automated techniques in isolation, such as [9], are limited to localizing problems at the level of granularity desired by users [3], the weak interpretability and extra labeling overhead are also coming with recently developed learning-based methods, e.g., iSQUAD [5].

Visualization can be used to migrate these limitations, however, current interactive tools such as [1, 4, 6, 8] only focus on showing the whole time usage of operators or Map/Reduce jobs and do not provide auxiliary information such as system profiling, which makes the reasoning of abnormal query execution very difficult. Perfopticon [6] and QEVIS [8] are the most relevant tools to DHive, they both directly illustrate the time usage distribution or parallelism of different operators for each Map/Reduce job. However, it is not sufficient to locate the root reason of abnormal time usage without any details about the execution process. For example, with only the time usage distribution of operators, it is difficult to reason if the abnormal time usage is caused by lower input speeding (e.g., I/O block), lower computing speed (e.g., CPU competition), or both. In DHive, we depict these details with dataflow by visualizing the data size and data processing speed over time for each task or job, which presents the much more fine-grained query process.

3 SYSTEM ARCHITECTURE

Figure 1 depicts the system architecture of DHive. It consists of three layers: the data collection layer, the data processing layer, and the visual interface layer.

3.1 Data Collection

DHive collects the data from multiple resources, including SQL statement, query execution plan (QEP), execution log from Apache Hive, and the data distribution (DD) in the initial and intermediate execution progress. Our objective is to trace the processing details of dataflow of a query execution. However, we find that the original log from Hive cannot provide enough details, where only the start/end time of each stage of tasks is contained as the most-grained information. Hence we add several new logging points to Hive to collect the temporal information of each task with a given timestamp such as total data size allocated, unprocessed data size, and output data size during the execution process, the specific code files with new logging points can be found at https://github.com/DBGroup-SUSTech/DHive.git.

3.2 Data Processing

The task is the basic execution unit spawned from Map/Reduce job for query processing. We first extract data from the log file to model a given task t associated with a set of attributes: $\langle st, et, j, ds, c, F_t \rangle$, where st, et, j, ds and c indicate the start time, end time, its corresponding job, the total data size the task processed and the related container (i.e., the allocated resource unit on a cluster node in Hive) executing this task. $F_t = \left\langle f_t^0, f_t^1 ... f_t^n \right\rangle$ is a tuple list that records the temporal features of task t. $f_t^i = (ts^i, u^i, o^i, d_{in}^i, d_{proc}^i, s_{in}^i, s_{proc}^i)$ indicate the feature at timestamp ts^i. u^i and o^i indicate the size of unprocessed data and output data of task t. d_{in}^i and d_{proc}^i indicate the total input time and computation time from ts^0 to ts^i. s_{in}^i and s_{proc}^i depict the input speed and computation speed of task t during the time range between ts^{i-1} and ts^i.$s_{in}^i = \frac{u^{i-1} - u^i}{d_{in}^i - d_{in}^{i-1}}$ and $s_{proc}^i = \frac{u^{i-1} - u^i}{d_{proc}^i - d_{proc}^{i-1}}$. Specifically, $s_{in}^0 = s_{proc}^0 = 0$.

QEP can be modeled as a directed acyclic graph (DAG) with the nodes as jobs and edges as the dependency relations. We denote QEP as $\mathbb{G} = (\mathbb{N}, \mathbb{E})$, where $\mathbb{N}$ is the jobs set and $\mathbb{E}$ is the edge set. We further calculate the aggregated temporal information of the whole query and jobs. For instance, job $j = \left\langle st, et, ds, F_j \right\rangle$ where st, et and ds are start time, end time and total data processed by job j. $F_j = \left\langle f_j^0, f_j^1 ... f_j^n \right\rangle$ where $f_j = (ts^i, u^i, o^i)$, u^i and o^i are the total unprocessed data and output data of job j at timestamp ts^i.

3.3 Visual Interface

Figure 2 illustrates the user interface of DHive, which is built upon the data collection and processing layers. DHive consists of three coordinated views which enable users to explore the query execution at multi-levels: query level, job level, and task level. We elaborate on the design of the user interface in section 4.

4 VISUALIZATION DESIGN

In this section, we discuss the visual interface of DHive. To facilitate the visualization design, we first formulate four goals informed by the observation of common analysis practices and review results from state-of-the-art visualization tools. The final four goals are presented as follows:

- G1: Illustrate the logical plan of each query.
- G2: Visualize the overall execution process of the query, i.e., show the duration and processing progress of each job.
- G3: Trace the execution of individual tasks during the overall query execution progress.
- G4: Support query execution process comparison.

Guided by these goals, we design four coordinated views in DHive shown as Figure 2: (i) *query selection view*; (ii) *query logic view* (G1-2); (iii) *dataflow overview* (G2) and (iv) *processing details view* (G3). These four views profile the query execution at multi-grained, where the *query logic view* and the *dataflow overview* illustrate the logic content and execution speed (bytes/sec) of query at jobs level, respectively, and *processing details view* profiles each task execution process. Meanwhile, DHive automatically links related elements in these four views and supports execution comparison between different queries by sharing the same time axis (G4).

Figure 2: The visual interface of DHive (best viewed in color)

Query selection view: *Query selection view*, shown in Figure 2(A), lists all the queries to be analyzed. After selecting a query for analysis, the general execution information will be displayed on the right panel, such as the total time usage and total data size.

Query logic view: As shown in Figure 2(B), the *query logic view* displays the QEP as a directed acyclic graph (DAG) enhanced with a timeline. Each node in this DAG represents a Map/Reduce job, The left and right positions of each node are aligned with the start and end time of the corresponding job. The nested rectangles in each node illustrate its time distribution for input, computation, and output, encoded with red, green, and pink colors, respectively. The job dependencies are visualized as the edges with grey color. We have implemented a greedy algorithm in [4] to lay out the DAG with time information for a better visual result.

Dataflow overview: *Dataflow overview* visualizes the overall data processing procedure of query and jobs shown in Figure 2(C1, C2). We design a river-flow-based visualization [2] to present the data evolution over time. Figure 3 displays the design details: during the execution of each task, the data is recorded as (i) prepared: the status of data before task processing; (ii) unprocessed: unprocessed data during the task processing; and (iii) output: the output data. Figure 3(A) illustrates task dataflows consisting of these three statuses encoded by red, green and blue color. Meanwhile, the height of the flow represents the data size at each timestamp. The job and query dataflows are generated by stacking of task dataflows along the time axis, as shown in Figure 3(B) and Figure 3(C).

Figure 2(C1) shows an example of dataflow visualization of a whole query. We use glyphs to represent filter (σ) and join (⧓ for *map join* and ⋈ for *merge join*) operators, which are overlaid at the timestamp they occur. Figure 2(C2) shows dataflows of individual jobs, we implement a novel greedy algorithm to place all the job dataflows without overlap aligned with time, and the job dataflows with dependencies are placed at a closer vertical distance.

Processing details view: *Processing details view* (Figure 2(D)) visualizes the data processing details for tasks. There are three panels

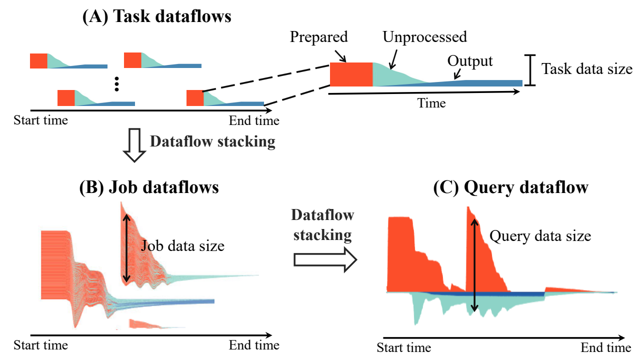

Figure 3: Example of dataflow stacking process

in this view: (1) data parallelism (Figure 2(D1)), (2) task execution details (Figure 2(D2)), and (3) counter statistics (Figure 2(D4)).

▹*Data parallelism.* As shown in Figure 2(D1), the data parallelism for the whole query execution is shown as the grey area chart, a nested area chart will be colored with yellow to depict the data parallelism of a selected job. For instance, Figure 2(D1) illustrates the data parallelism during "M1" execution process.

▹*Task execution details.* Figure 2(D2) visualizes the execution details for each task. In this view, each row represents a container and displays the tasks executed by that container in sequential order based on their execution time. Each task is visualized as a rectangle, and the task processing details are presented by the input and computation speed encoded with red and blue, and the gradient color from white to dark red or dark blue present the speed size of input and computation from minimum to maximum, respectively. Meanwhile, for some Map tasks, the stages of input and computation may be overlapped. Hence the top half of the rectangle is used to encode the input speed for these tasks, while the bottom half is used for computation speed encoding. Moreover, whenever a job is

selected from other views (e.g., "M1" is chosen in Figure 2(B)), its corresponding task dataflows will be highlighted in Figure 2(D2). The essential processing details for each task will also be shown by task selection, for example, Figure 2(D3) presents the information from the four tasks executed by container *08-28*.

▹*Counter statistics.* The statistical information from Hive counters is shown in Figure 2(D4), which provides different measure metrics of the execution of the tasks. For the tasks spawned from one job, similar behaviors should be exhibited [7]. Hence we sort the metrics according to their degree of dispersion of the value in the selected job. The higher score means the more discrete numerical distributing and the corresponding metric will be presented first with darker green encoded in Figure 2(D4). After selecting one metric, a scatter chart in the right panel shows the exact distribution of these values, where each task is presented as a scatter plot and is placed according to its start time and metric value.

5 DEMONSTRATION OVERVIEW

DHive is implemented using Python Flask and Vue. Our demonstration will use the queries in TPC-DS with 100GB input data, and all these queries are running on a production cluster with 9 machines.

The goal of our demonstration is to show VLDB attendees the effectiveness of DHive and we also allow the audience to experience the interactive features of DHive through a web application. In the following, we will describe the two demo scenarios we aim to show in detail.

Single query performance analysis: The first scenario is to understand the query execution process of TPC-DS Query 29, which takes 76 seconds. After selecting this query in Figure 2(A), we first start exploration from the *query logic view* in Figure 2(B). Obviously, the time durations of Map jobs M1 and M8 are significantly larger than other jobs. Then, by checking the *dataflow overview* in Figure 2(C2), we find both M1 and M8 incur long tails, which means a slow data processing speed during their executions.

To diagnose this problem, we click the M1 node in the *query logic view*, and the corresponding tasks are highlighted automatically in the *processing details view*, as illustrated in Figure 2(D2). Apparently, several tasks of M1 take significantly larger execution time than other tasks. For example, for the container *08-28* (i.e., the container with green border), the last highlighted task executed by it has a longer execution time than the first three highlighted tasks. To investigate the reason for time usage difference, we then click these four tasks and analyze them in Figure 2(D3). We find these four tasks are all processed at dbg08, and the major difference is the source of input data and input speed, where the slowest task (i.e., the last task at the bottom) fetches the data from dbg09, instead of dbg08. Thus, it incurs expensive overhead to move the data from dbg09 to dbg08 via inter-connects. In this scenario, the performance bottleneck is the overhead of data movement during the execution progress, bounded by the network bandwidth.

Execution performance comparison: In the second case, we show how to analyze the reason for the performance improvement of TPC-DS Query 29 on Hive via the help of DHive. Specifically, the previous running time for Query 29 is 45s, while the last execution cost is 76s with the same input data. After selecting both queries in the *query selection view*, as shown in Figure 4(A1) and Figure 4(B1),

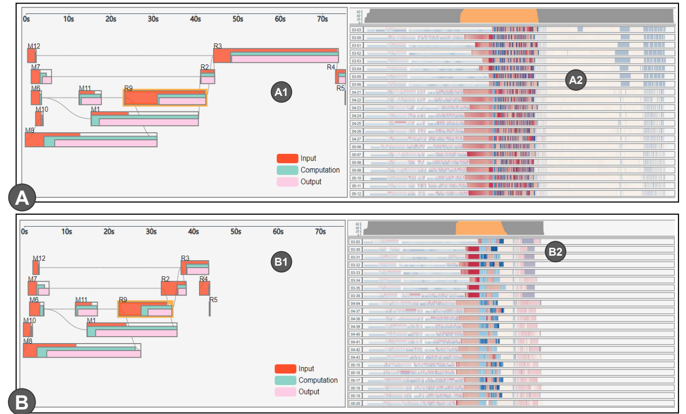

Figure 4: Execution comparison: (A) the previous slower execution progress and (B) the last faster execution progress.

these two execution progresses are visualized with a shared time axis. Obviously, the time costs of Reducer jobs R9, R2 and R3 in the last execution are smaller than those in the previous one.

To identify the reasons for this improvement, we click R9 node and highlight its corresponding tasks in Figure 4(A2) and Figure 4(B2). Compared with the visualization result in Figure 4(A2), fewer tasks are executed by each container in Figure 4(B2), while the computation time for each task (i.e., the longer blue segment) is slightly longer. Interestingly, the total shuffling time is reduced, shown as the thin red area shown in Figure 4(B2), which means that in the last execution, each task of R9 incurs a slightly longer computation time, but the data shuffling cost of R9 is significantly reduced. Thus, we can conclude that in the last execution, the number of tasks of job R9 is less than the previous one, but the processed data of each task is larger than the previous one. Motivated by the above observation, we found the value of *hive.exec.reducers.max* parameter in Apache Hive is changed from 1008 to 150, which is the root reason for the query performance improvement.

ACKNOWLEDGMENTS

This work was partially supported by Shenzhen Fundamental Research Program (Grant No. 20220815112848002) and the Guangdong Provincial Key Laboratory (Grant No. 2020B121201001).

REFERENCES

[1] 2023. Tez UI. https://tez.apache.org/tez-ui.html
[2] Weiwei Cui and et al. 2011. Textflow: Towards better understanding of evolving topics in text. *TVCG* 17, 12 (2011), 2412–2421.
[3] Elmer Garduno and et al. 2012. Theia: Visual signatures for problem diagnosis in large hadoop clusters. In *LISA*. 33–42.
[4] Haotian Liu and et al. 2022. GHive: A Demonstration of GPU-Accelerated Query Processing in Apache Hive. In *Proceedings of the 2022 ACM SIGMOD*. 2417–2420.
[5] Minghua Ma and et al. 2020. Diagnosing root causes of intermittent slow queries in cloud databases. *Proceedings of the VLDB Endowment* 13, 8 (2020), 1176–1189.
[6] Dominik Moritz and et al. 2015. Perfopticon: Visual query analysis for distributed databases. In *Computer Graphics Forum*, Vol. 34. Wiley Online Library, 71–80.
[7] Chris Muelder and et al. 2016. Visual analysis of cloud computing performance using behavioral lines. *TVCG* 22, 6 (2016), 1694–1704.
[8] Qiaomu Shen and et al. 2023. QEVIS: Multi-grained Visualization of Distributed Query Execution. *IEEE VIS* (2023).
[9] Dong Young Yoon and et.al. 2016. Dbsherlock: A performance diagnostic tool for transactional databases. In *Proceedings of the 2016 ACM SIGMOD*. 1599–1614.