QEVIS: Understanding and Diagnosing the Fine-grained Execution Process of Hive Query via Visualization

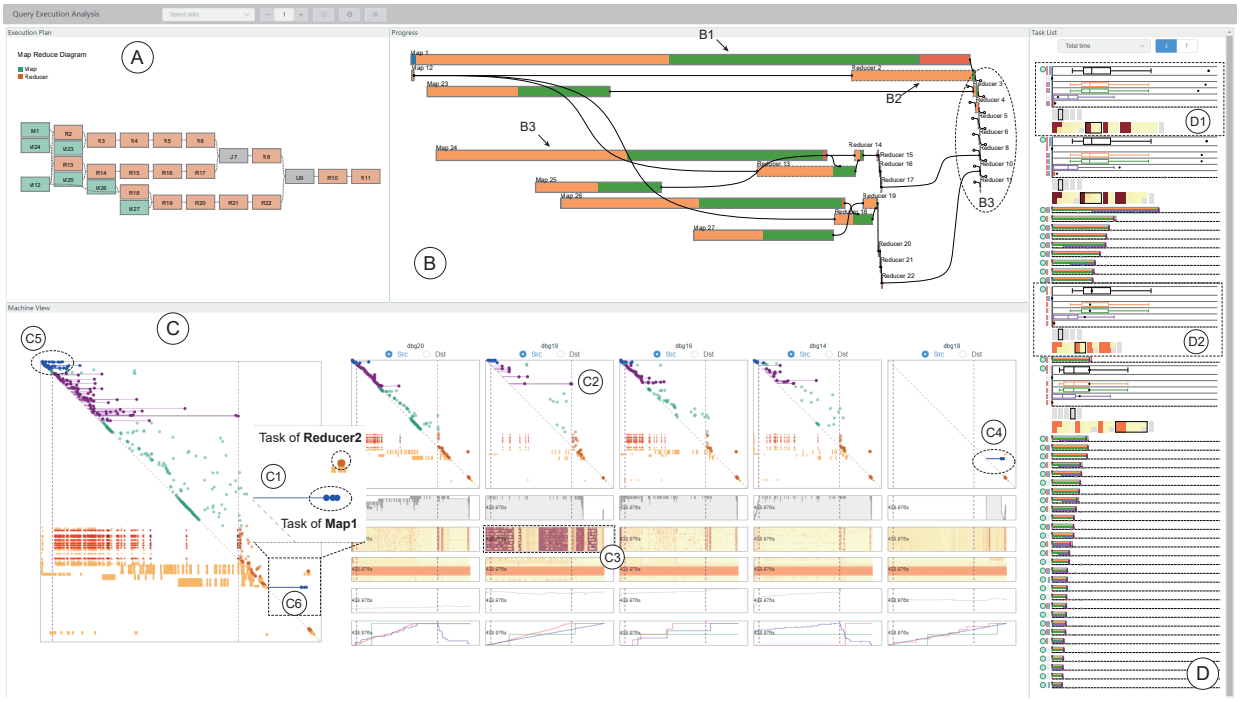

Fig. 1. QEVIS facilitates the interactive exploration of query execution analysis from three levels. (A) Execution plan present the vertex logic; (B) Progress view presents the overall execution progress and topology of the logic vertices; (C) System view, consists of task distribution components and system performance components, allows users to explore the task dependencies and reason the task patterns; (D) Task View provides the detailed information of tasks, including: abnormal steps, dependencies and the data size.

Abstract— Understanding the query execution process of distributed databases is crucial to many real-world practices such as detecting query bottlenecks and improving system performance. However, analyzing distributed query execution is challenging due to a large number of parallel tasks and the complex dependencies among these tasks. Moreover, system statuses such as network and memory can also affect query execution in complicated ways. Existing techniques typically collect *aggregate statistics* of system performance (e.g., CPU, memory, and disk I/O) or execution progress (e.g., operator duration), and thus cannot track *fine-grained query execution process* at the atomic task level, which is key to reason the query behavior. To tackle this problem, we propose QEVIS, a visual analytics system that supports understanding and diagnosing distributed query execution from multiple views in an interactive manner. We tailor-make an algorithm for temporal directed acyclic graph (TDAG) layout to visualize the overall structure and execution process of the query plan. A suite of novel visualization and interaction designs are introduced to analyze the tasks, data dependency, and relation between atom task execution and system performance metrics. We illustrate the effectiveness of our QEVIS with three real-world case studies (i.e., query optimization, system configuration and xxx) and interviews with domain experts.

Index Terms—Radiosity, global illumination, constant time

1 Introduction

Distributed database systems are becoming increasingly pervasive due to the need to handle large-scale data in various domains such as finance, e-commerce and science. Many such systems have been developed, including Hadoop [9], Flink [3] and Myria [17], which support specifying queries using high-level languages such as SQL. A query is typically executed by first translating its high-level specification into an optimized logical execution plan, and then spawning parallel tasks across the machines in the cluster to execute the logical plan. For trouble shooting and performance optimization, users of distributed database systems need to analyze and understand how queries are actually executed in a cluster. Frequently asked questions include "Where does the query execution time go?", "What is the bottleneck of my query?", "How can I improve the performance of a specific query?".

Many works have tried to understand query execution in databases [8, 13, 22, 24] and they can be classified into three categories depending on their focuses: (i) query logic, (ii) query execution plan and (iii)

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

actual query execution process. Understanding the query logic has been well-studied and there are established methods for complex queries with deep and nested structures [8, 24]. Works on execution plan understanding typically translate an execution plan specification (can contains up to thousands of lines) into intuitive diagrams (such as tree or graph) for easy comprehension and allow users to explore the execution plans interactively [27, 33]. However, there are relatively fewer works on understanding the actual query execution process, which is more challenging than understanding query logic and execution plan for the following three reasons.

A larger number of atom tasks: In distributed database systems, a query usually invokes many parallel *atom tasks* (e.g., xxx) across the machines in cluster, which are the minimum execution units. These tasks may exhibit significant differences in their computation complexity, data access pattern, memory requirement, network transfer and consequently execution time. Therefore, long running tasks are not necessarily anomalies and it is challenging to pinpoint suspicious tasks from a large number of tasks for human inspection.

Complex dependencies among the tasks: There are data dependencies among the atom tasks of a query as an atom task may take the outputs of one or more tasks as inputs. These dependencies can be very complex considering the large number of atom tasks, different types of interactions (e.g., synchronous or asynchronous) among the tasks and the fact that the tasks are executed distributedly. As a result, an abnormal task may not be caused by its own problems and it is difficult to trace back the task dependencies to identify the root causes.

Complicated system influences: System statuses such as network bandwidth, memory capacity and CPU load directly determine the execution efficiency of atom tasks. Different tasks have different resource requirements, e.g., some are network-bound, some are computation-bound while some require large memory, and thus resource availability may influence different tasks in different ways. In addition, tasks on the same machine may contend for resources and there may be load imbalance among the machines. These complexities make it difficult to correlate task execution with system performance and identify the system causes of execution problems.

Existing works on understanding the query execution process are inadequate as they focus on the high-level statistics []. For example, xxx [] and xxx []. However, we observe that for many real-world use cases (e.g., query performance optimization and system problem identification), it is necessary to track the fine-grained query execution process at the atom task level and correlate atom task execution with both data dependencies in the query plan and statuses of the underlying system.

In this work, we develop a visual analytics system called QEVIS (see a demo in Figure 1) to facilitate distributed database users in understanding and diagnosing the fine-grained query execution process. QEVIS is designed following the visualization Mantra "overview first, zoom and filter, details on demand" [31]. QEVIS provides three coordinated views and a suite of interactions to support the interactive analysis of query execution with different levels of details. In the *progress view*, we design a tailored algorithm to layout the temporal directed acyclic graph (TDAG) to show the overall query execution progress, which takes both the temporal information and the topology information into consideration. In the *task view*, we show the detailed information about individual tasks and their dependencies, which allows to easily identify abnormal tasks. In the *machine view*, we integrate task execution with system statics, which enables users to correlate task execution with system statuses. *How do the three views interact?* QEVIS can run in two modes: (i) *monitor mode*, which shows the query execution process in real-time; (ii) *analysis mode*, which replays the execution process at a user specified rate. *In which cases are the two modes are useful?* Although we focus our discussions on queries in Hive, we believe the designs of QEVIS can generalize to other distributed data processing systems, such as parameter server-based machine learning systems [], vertex-centric graph processing systems [] and streaming processing systems [3].

Case studies; summary of domain experts interview; short para

Our contributions in this works can be summarized as follows:

- We identify the requirements and functionalities of a visualization system for understanding the fine-grained query execution process in distributed database systems through extensive discussions with domain experts.
- We design the QEVIS system, which can visualize the overall execution progress of distributed queries, show the task dependencies and allow effective identification of abnormal tasks, and support correlating task execution with system statuses.
- We conduct comprehensive case studies using QEVIS to analyze Hive queries, which shows that QEVIS is a valuable tool that makes complex tasks such as abnormal detection, bottleneck identification and reason easy.

2 Related work

2.1 Visual Analysis for Database Queries

Both the database and visualization communities have proposed methods to analyze query performance and diagnose performance problems manually or automatically. We briefly review related works on *understanding query logic* and *analyzing query execution plan*, and refer the interested readers to [11] for a systematic survey on database query debugging and performance analysis.

Understand query logic. Query specifications in high-level languages such as SQL can be difficult to read as they may have deep and nested structures. Many research works have tried to provide an intuitive understanding of the query logic [1, 4, 8, 12, 19, 24]. The most common way is to leverage visualization techniques to assist query understanding. For example, GraphSQL [4] and Visual SQL [19] propose visual query languages, which support query visualization and interactive query tuning. QueryVis [24] uses a node-link diagram to show the relation among the operators and proves that node-link diagram can expresses queries without ambiguity. Besides visualization, Gawade et al. [22] proposed a method to translate a query into its descriptions in natural language.

Analyze query execution plan. As introduced before, (distributed) database systems first generate a logical execution plan for a query and then execute physical tasks according to the plan. Understanding how the logical plan is executed can enable users to reason query performance. Many industrial softwares (e.g., Tez UI, *xxx* and ***xxx***) visualize the query execution process at the operator level [10]. They use tree or directed acyclic graph (DAG) to show the relation among the operators and adopt the Gantt chart to visualize the execution progress. For example, VQA [33] displays the logical query plan as a tree with nodes indicating operators and edges indicating dataflow. Bar charts are inserted in the nodes to show the execution statistics of the operators (e.g., execution time, memory allocated). Perfopticon [27] provides separate views for the logical query plan, overall execution progress, and system performance statistics. It also allows users to observe the execution statistics of the operators on different workers and can help identify performance problems such as slow workers and data skew.

Existing works analyze query execution at the *operator level* at best while our QEVIS analyzes query execution at the more fine-grained task level. As task is the minimum (i.e., atom) execution unit in Hive, tracking task execution allows to accurately identify the causes of errors and performance problems. For instance, data skew can be observed by profiling the size of data processed by the tasks, query bottleneck can be analyzed by checking the tasks that block the data dependencies, and network or memory problems can be identified by measuring the resource consumption of the tasks. As an operator corresponds to many tasks across the machines, it is difficult to gain such fine-grained insights on the operator level. However, the large number of tasks and their complex dependencies make task-level execution visualization more challenging than operator-level visualization.

2.2 Visualization for Sequence Data

Query execution in distributed databases can be described by a sequence of events, e.g., the start, execution and finish of tasks on the machines,

and thus we review related works on sequence data visualization in this part. As a special type of time-series data, event sequence is defined as a series of discrete events recorded in the time order of their occurrences [16]. Sequence data is ubiquitous in many fields such as health care [25, 35], social media [23, 38], and education [5, 6, 15, 18, 28]. Sequence visualization aims to show the information of the events such as the event type, start time, end time and duration. For specific visualization applications, various tasks are proposed, such as visual summarization, prediction & recommendation, anomaly detection and comparison. Existing sequence visualization techniques can be classified into five categories according to their form of visual representations, i.e., *sankey-based visualization*, *hierarchy-based visualizations*, *chart-based visualizations*, *timeline-based visualizations* and *matrix-based visualizations* [16]. We briefly introduce them below and refer the readers to related surveys [16, 32] for more detailed discussions on time-series and event sequence visualization.

Sankey-based, hierarchy-based and matrix-based visualizations display an event sequence after mapping it to special structures such as graph or tree. For example, LifeFlow [36] utilizes the tree structure to summarize an event sequence, in which a node represents a group of events. Outflow [35] models the progression paths of an event sequence as a DAG with a node indicating a cluster of states, and visualizes the DAG as a Sankey diagram [30]. These methods provide high-level summarization for an event sequences and cannot show the detailed information of each event. Matrixwave [39] utilizes a sequence of matrices to visualize the dependencies among the events. However, it does not show the detailed temporal information of the events and cannot scale to long event sequence.

Chart-based visualizations uses barchart, linechart or scatter plot to visualize the trend or distribution of the events, which provides assisting views to support interactive explorations. For instance, barchart and linechart are used to show the distribution of the attributes in the event sequence over time [2, 14]. Scatter plot has been used to provide a coarse overview of an event sequence or some sequence groups by projecting them to the 2D canvas using dimension reduction techniques or two chosen attributes [14, 26, 37]. In addition to the distribution of the events, scatter plot has also been used to identify outliers [].

Timeline-based visualizations are recognized as the most intuitive way to demonstrate events in their time order. Specifically, the Gantt chart directly shows the temporal information of the events, including start time, end time and duration. Many industrial tools such as Tez UI, xxx and xxx use the Gantt chart to demonstrate the execution progress []. For example, Lifeline [29] uses the Gantt chart to display event sequences and the individual events in the sequences, and each sequence takes a single row. LiveGantt [20] proposes an algorithm to improve scalability for visualizing scheduling events. However, these methods cannot be directly used for understanding query execution as the dependencies among the events are ignored. In addition, the Gantt chart suffers from poor scalability when applied to a large number of events and makes it difficult to identify abnormal events without proper alignment.

3 Background

In this part, we introduce the query processing workflow of Hive (Hadoop-based) and related terminologies to facilitate further discussion. The query processing procedures of other distributed database systems are similar.

As illustrated in Figure 2, query processing in Hive takes four steps. First, the user issues a query via some interface such as web-based UI or SQL terminal as shown in Figure 2-(1). Then, Hive uses a cost-based optimizer to optimize the query execution plan (e.g., select the best methods to conduct the scan operators, determine a good order for the join operators and choose the index to use) and obtains a logical execution plan as shown in Figure 2-(2). The logical exaction plan can contain hundreds of lines that describe the query execution process and will be executed by the computation engine (e.g., Tez). We can model the logical exaction plan as a Directed Acyclic Graph (DAG), in which each **vertex** corresponds to a sequence of logical operators (such as filter, aggregate and etc.) that conducts some sub-steps of the query.

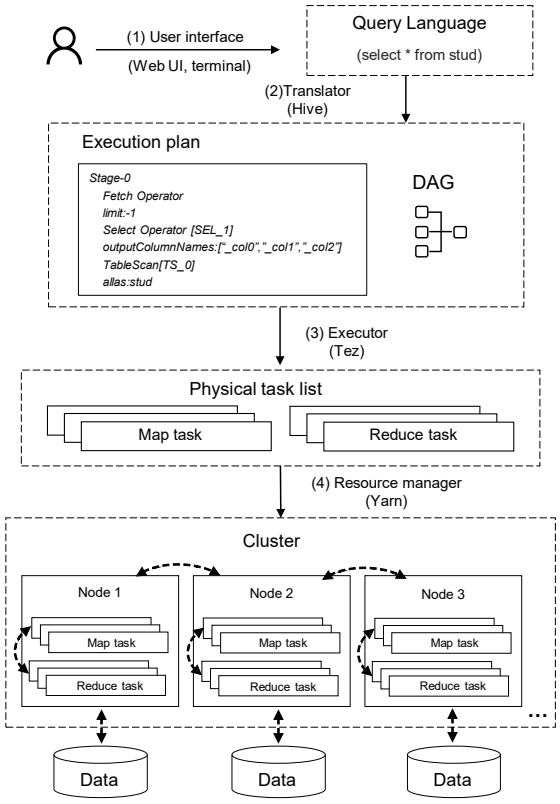

Fig. 2. The query processing workflow of Hive (Hadoop 2.0).

For Tez DAG, a vertex can be either a **map** vertex or a **reduce** vertex depending on it conducts map tasks or reduce tasks. The **edges** in a DAG are directed and model the data movement between vertices. For an edge, we call the source vertex **producer** vertex and the destination vertex **consumer** vertex. Note that a producer vertex may connect to multiple consumer vertices and a consumer vertex can take inputs from multiple producer vertices.

According to the logical DAG, Tez generates the physical execution plan by spawning a set of atomic **tasks** for each vertex (as shown in Figure 2-(3)). These tasks are then dispatched to the physical machines by Yarn as shown in Figure 2-(4), the resource manager of Hadoop2.0. Each task takes a piece of data as input and executes all the operators in its corresponding DAG vertex. To track the fine-grained execution process of the tasks, we decompose each task into five steps. For a map task, the steps are *Initialize*, *Input*, *Process*, *Sink*, *Spill*. For a reduce task, the second step is *Shuffle* instead of *Input*. Note that the data read by a task could come from its local file or remote producer tasks.

4 Design Goals and System Architecture

In this part, we identify the design goals of QEVIS as a visualization system that supports fine-grained understanding of the query execution process and introduce its overall architecture.

4.1 Requirement Analysis

In the QEVIS project, we collaborated closely with three experts in distributed database systems, who are also co-authors of this paper. In the first month of the project, we brainstormed to collect the most frequently raised questions when analyzing the performance of distributed query processing. Based on these discussions and by reviewing existing literature, we identified the following design requirements for QEVIS.

R1 Understand the logical query plan and overall query execution progress. Before our collaboration, the domain experts used profiling software (Tez UI, etc) or visualization tools (Tableau, etc) to show the query plan as DAG and the query progress as

Gantt chart. Although the two views help understanding query processing, the domain experts complained that they need to switch their focuses back and forth as the two views are separated, which breaks the continuity of exploration.

R2 Understand query execution at the atom task level. Task is the minimum execution unit in Tez and visualizing the execution of individual tasks is crucial for identifying the bottleneck of query processing. However, it is challenging to visualize the tasks. First, there are a large number of tasks and thus using traditional Gantt chart will require a very large rendering canvas. Rich information, such as input/output data size, start/finish time and involved operators, should be visualized alongside the tasks to facilitate understanding. Moreover, the many-to-many dependencies among the tasks make it difficult to design clear and **scalable** visualization.

R3 Provide visual insights for reasoning query performance and correlating with system statuses. QEVIS should make it easy to identify performance problems and conduct query optimization. Specifically, it should visualize the tasks in intuitive ways to support pinpointing suspicious tasks that may be the performance bottleneck. As system statuses such as network bandwidth and CPU load are also important for task execution, QEVIS should provide visualizations to connect task performance with system statistics.

R4 Support interactive exploration. In addition to the visualization designs, QEVIS should provide flexible interaction mechanisms that allow users to navigate to any time range, DAG vertex, task group or individual task of interest. The linkage among correlated visual elements should also be considered in the design to coordinate information from different perspectives.

4.2 Task Analysis

Guided by the design requirements, we discussed with the domain experts about the visualizations QEVIS should provide and distilled the following visualization tasks:

T1 Jointly visualize the query plan and execution progress. To ensure the continuity of exploration (R1), the logical query plan and query execution progress should be integrated into one visualization view. The visualization should meet several important criteria such as minimize the usage of canvas, reduce the number of links and show the task dependencies clearly.

T2 Visualize comprehensive information about the tasks. To facilitate fine-grained exploration of query execution (**R1**, **R2**), rich information about the tasks should be visualized, including the size of the data processed by each task, the data-flow among the tasks, the temporal information of individual tasks (start time, finish time, duration and etc.) and the corresponding sub-process. Moreover, the abnormal tasks (which may be the query bottleneck) and related execution traces should be easily observed.

T3 Visualize the system statuses. To investigate the system reasons behind task performance (**R3**), QEVIS should display the execution statistics of the overall system and each individual machine such as the amount of send/receive data, disk IO pending list, CPU usage and memory occupancy. These system statistics should be well linked with specific execution patterns of tasks.

T4 Interaction and linkage. QEVIS should provide flexible interactions that allow users to switch their focuses between the different points of interest, such as a specific time range, a vertex or a group of tasks (**R2**, **R3**, **R4**). For example, a user may select a vertex in the logical plan and try to determine its tasks are CPU-bound or IO-bound. This requires the visualization to show the related tasks when choosing a vertex and highlight the corresponding CPU usage and disk information in a cohesive manner.

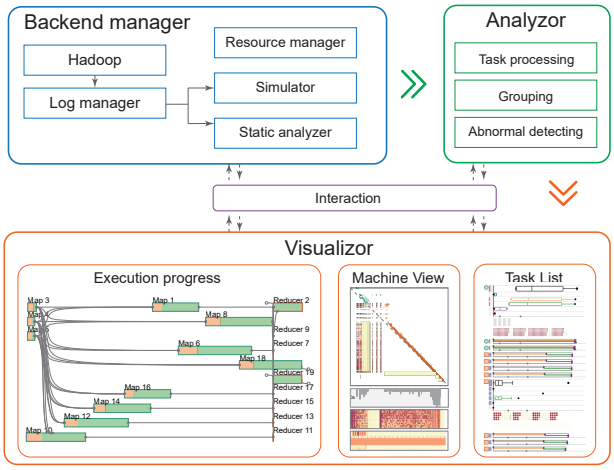

Fig. 3. QEVIS system consists of three major components, i.e., backend manager, analyzer and visualizer.

4.3 System Architecture

We build QEVIS on the top of Hadoop2.0 with **Hive** as the query optimizer and **Tez** as the executor. As shown in Figure 3, QEVIS consists of three modules: backend manager, analyzer, and visualizer.

The **Backend Manager** module performs log processing and data fusion for query execution data and system performance data. When collecting execution data from the Hadoop system, the Log Manager cleans and pre-processes the log files, extracts important metrics and saves them as local files. Log Analyzer takes these files as input and processes them as structured data for future processing. Monitor segments the data at regular time ranges and output them at a given rate. The Simulator simulates the query execution process, and allows users to adjust the running speed and explore the execution process interactively. Moreover, Resource Manager collects system performance metrics and outputs them to the next processing stage.

The **Analyzer** fuses the execution data and system performance data by timestamps. The tasks will be grouped according to their corresponding vertices in the logical execution plan. The dataflow among the tasks (i.e., dependencies) are also recorded in this step. Moreover, Analyzer also performs the anomaly detection for the tasks to support in-depth analysis.

The **Visualizer** module integrates coordinated views to support interactive exploration of the query execution process and reasoning about query performance from multiple levels. The Execution Overview demonstrates the execution process at the vertex level. An algorithm for temporal DAG layout is proposed to visualize the structure of the logical execution plan and the query execution progress simultaneously. The task group view consists of two components: (i) task overview, which visualizes the temporal information and data dependencies of the tasks executed on the same machine, and (ii) system statuses, which shows the performance statistics of the corresponding machine. The task list view provides more detailed information at the operator level, enabling the users to analyze the execution statistics of individual tasks.

5 Data Collection and Preprocessing

5.1 Data Collection

As shown in Figure 3, our visualization pipeline starts from the plan, log, and system performance data collection.

Query plan parsing: As introduced in Section 3, in Hive, a logical execution plan is described as a text file that contains hundreds to thousands lines of descriptions. We parse this file and generate a DAG with the vertices being the operators and the edges modeling the data dependencies.

Execution log processing: As Hadoop execution logs are collected from tedious system log, we locate the records of interest by first matching keywords from a pre-defined keyword set and then parsing these logs to extract the required information. The information we save

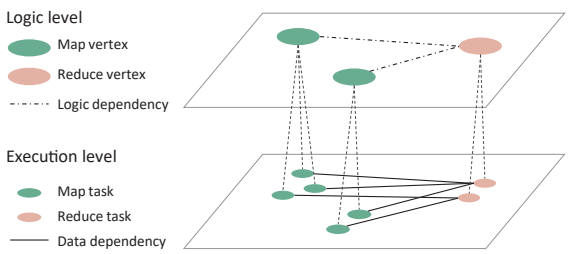

Fig. 4. Two-level execution graph.

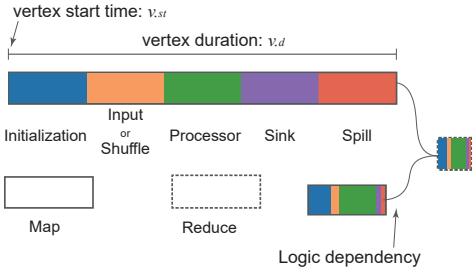

Fig. 5. Visual encoding of logic vertex and dependencies

to local files include task id, the logical vertex corresponding to each task, the temporal information (start time, duration) of individual tasks, and the temporal information of the sub-steps in a task.

5.2 Data Modeling

The data we collected from the backend manager can be modeled as a temporal graph with two levels, i.e., the logical level and the execution-level, which is shown in Figure 4.

The logical level graph is the DAG extracted from the execution plan, denoted by $\mathbb{G}_L = (\mathbb{V}_L, \mathbb{D}_L)$, where $\mathbb{V}_L$ is the logical vertex set and $\mathbb{D}_L$ is logical dependency set between the vertices. The execution level graph is denoted as $\mathbb{G}_E = (\mathbb{T}_E, \mathbb{D}_E)$, where $\mathbb{T}_E$ is the set of tasks executed by the physical machines and $\mathbb{D}_E$ indicates the data dependency set between tasks. If a task $t \in \mathbb{T}_E$ is a physical instance of vertex $v \in \mathbb{V}_L$, we describe this relation using $t \to v$. We also apply this description to the relation between logical dependency $d_L \in \mathbb{D}_L$ and psychical dependency $d_E \in \mathbb{D}_E$ and use $d_e \to d_L$. Moreover, we use set $P(v) = \{t | \forall t \to v\}$ to indicate all tasks that are physical instances of logical operator $\mathbb{V}_L$. A map task t has five steps and is recorded as an array $S =< s_{init}, s_{input}, s_{proc}, s_{sink}, s_{spill} >$ while a reduce task is recorded as $S =< s_{init}, s_{shuffle}, s_{proc}, s_{sink}, s_{spill} >$. Each step is associated with a pair of time $< st, d >$, which indicates its start time and duration. Note that different steps of the same task may overlap in time due to pipelining.

Each task is modeled as a sequence of attributes $t :=< st, d, v, m, S_E >$, where st, d and v indicate the *start time*, *duration* and logical vertex of task t. m is the machine that executes t and S_T is the corresponding steps. We use $t.attr$ to index an $attr$ of t. A logical vertex is modeled as a triplet $v :=< st, d, S_L >$. The start time of v is $v.st = min(\{t.st | t \in P(v)\})$, the duration of vertex v is $v.d = v.st + max(\{(t.st + t.d) | t \in P(v)\})$. The steps $S_L = \{d_{init}, d_{input}, d_{proc}, d_{sink}, d_{spill}\}$ or $\{d_{init}, d_{shuffle}, d_{proc}, d_{sink}, d_{spill}\}$ according to the type of the vertex (i.e., map or reduce). For a given v, $d.attr = sum(\{s.attr | s \in S_L\})$ where $attr \in \{init, input, shuffle, proc, sink, spill\}$.

6 Visualization Design

Following the data modeling, we present our web-based visual analytics system QEVIS, which supports interactive exploration of the query execution process with four coordinated views. The *execution progress* demonstrates an overview about how the query plan is executed (**T1**), the *task distribution view* shows the task distribution and their data dependencies. Integrated with the performance statistics of the machines, this view also help reason the system causes of task performance (**T2** and **T3**). *Task list* provides detailed information of individual tasks (**T2**). At last, interaction and linkage are introduced to support multi-level exploration (**T4**).

6.1 Query Progress View

Query progress view is developed to show the overall progress of query execution and the logical dependencies among the operators.

6.1.1 Visual Encoding

The Gantt Chart [7] is widely used to visualize progress data, which maps the time interval to the horizontal axis and lists the progress bars on the vertical axis. As shown in Figure 5, for a vertex v in the logical execution plan, the leftmost x-coordinate of its bar indicates the start time $v.st$ and its duration $v.d$ is encoded by the length of the bar. We use stroke and dash to represent the type of vertex (i.e., map or reduce) and adopt different colors to encode the sub-steps of a vertex. Note that the order of the sub-steps is fixed as shown by Figure 5. We use B-splines to show the dependencies among the tasks.

Algorithm 1 TDAGLayout (*R*, *canvas*)

```
1: for root in R do
2:     process(root, canvas)
3: end for
```

6.1.2 Layout Method

The Gantt chart has been used in many progress visualization tools such as Tez UI and Tabula, in which each progress bar takes up an entire line as shown in Figure 6(A). Vertically, the progress bars are ordered by the start time of the corresponding vertices. In existing visual design, the layout doesn't consider the logical dependencies among the vertices, which results in many cross edges and fails to show a clear topology structure.

To tackle this problem, we design Algorithm 1 to layout the TDAG. As the horizontal position and width of the bars are used to encode temporal information, only the vertical position can be adjusted to optimize the layout. For an edge $e = \{u, v\}$, we define u as the child of v and v as the parent of u. Moreover, we define the *root* of a TDAG as the vertex which has no parents. As described in Algorithm 2, process uses an in-order traversal to process the vertices in the TDAG, in which place is a function that tries to place a vertex in the canvas. At first, we follow the naive approach and place each vertex in an exclusive row. As shown in Figure 6(B), the layout shows a clearer topology structure compared with Figure 6(A) as the sibling vertices are placed close to each other. However, this method requires a large canvas because each vertex takes up a row, which will result in serious scalability problem when the TDAG contains many vertices.

To provide a tight layout, we revise the place method by trying to place each vertex at the top of the canvas. If a vertex overlaps with an existing vertex, then we move this vertex down by the width of a bar uh. If place cannot find a position for the vertex in the canvas, it returns $false$ indicating that the height of the canvas is not enough. Each parent vertex will be visited multiple times until it is successfully placed into the canvas (line 10 14 in Algorithm 1). As shown in Figure 6(C), this layout requires a smaller canvas than Figure 6(B). However, the tight layout does not show a clear dependency structure as it compress the space to put the links and creates many cross links. Therefore, we further optimize the placement method by changing the position of the control points of the B-splines. For two different source vertices, the x-positions of their control points are different, which enables the links from different source vertices to attach to different positions. The result is shown in Figure 6(D).

6.2 Pattern Explorer

Pattern explorer is designed to support tracking execution performance at the task level and reasoning the system causes of performance problems. As there is a large amount of information to visualize, pattern explorer is further divided into distribution view and performance view.

Algorithm 2 process(v, *canvas*)

```
1:  if v.visit = True then
2:      Do Nothing
3:  end if
4:  if v.children is empty then
5:      if place(v)= true then
6:          canvas.height ← canvas.height + uh
7:          place(v)
8:      end if
9:  else
10:     for c in v.children do
11:         process(c)
12:         if place(v)= true then
13:             v.visit ← True
14:         end if
15:     end for
16:     if v.visit = True then
17:         canvas.height ← canvas.height + uh
18:         place(v)
19:     end if
20: end if
```

6.2.1 Distribution View

The distribution view is designed to show the temporal information and dependencies of the tasks.

Visualizing the temporal information: Before our collaboration, the domain experts used the Gantt chart to display the overall progress of tasks, as shown in Figure 7(C). However, the Gantt chart suffers from severe scalability problem for our applications as hundreds to thousands tasks are associated with a logical vertex and thus a very large canvas is required to show all horizontal bars clearly. Moreover, visualizing many bars in the Gantt chart makes it difficult for users to compare the length of the bars because of the lack of alignment. This hinders user's ability to discover group-based patterns and identify abnormal tasks.

To solve this problem, we design a scatter plot-based visualization, which is shown in Figure 7(A). Figure 7(B) illustrates the details of this design: use square shaped canvas as the basis, the vertical coordinate (from top to bottom) indicates the start time of a task, and the horizontal coordinate (from left to right) indicates the end time of a task. A task is visualized as a dot in the canvas according to its start time and duration. This design has two benefits. First, the horizontal bars in the Gantt chart is simplified as dots and thus a larger number of tasks can be visualized as point cloud in the canvas. Point cloud may suffer from visual clutter due to the overlap of the dots but we argue that it enables easy identification of clusters and outliers, which helps users observe performance patterns and pinpoint suspicious tasks. Second, as the start time and end time of a task are mapped to the horizontal and vertical axis respectively, the horizontal distance between a task and the diagonal line of the canvas represents the duration of a task (see task t_1 in Figure 7(B) for an example). This design allows users to easily compare the duration of the tasks and if a task has long distance to the diagonal line, it is likely to be abnormal.

We compare our design with the Gantt chart in 7(C), in which the height of the bars are scaled such that the Gantt chart takes up the same canvas size as our design. The results show that it is easier to observe the overall temporal distribution and identify the outliers in our design.

Visualizing the task dependencies: The most common way to visualize data dependencies is using curves to connect related entities as in the progress view (Section 6.1). However, such design will result in serious visual clutter when dealing with a large number of tasks. To solve this problem, we also visualize task dependencies as dots on the canvas. As shown in Figure 7(B), the dependency d_E of task t_1 on t_3 is visualized as a dot (the red colored rectangle) in the left bottom part of canvas with its y-coordinate being the end time of the producer task and x-coordinate being the start time of the consumer task. We also use heatmap instead of point cloud to improve render efficiency for a large

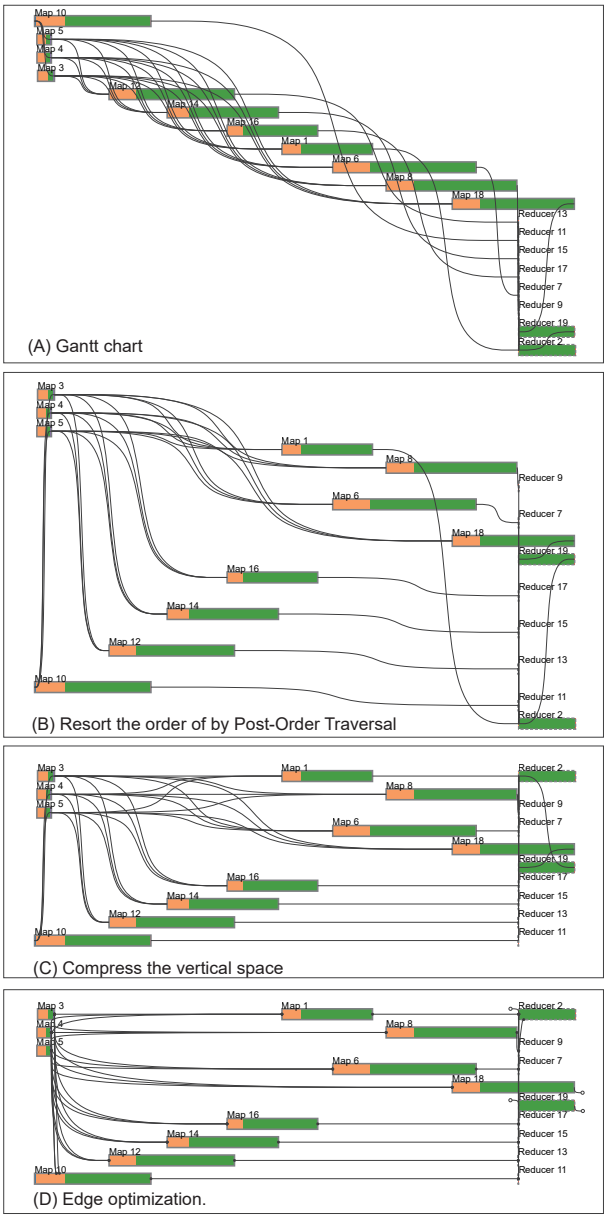

Fig. 6. Temporal DAG layout algorithms

number of points.

This design utilizes the left half of the canvas and shows the distribution of the dependencies according to their temporal information, allowing users to discover long dependencies among the tasks. To facilitate interactive exploration, we implement several interaction mechanisms in this view. For example, once a task is selected, its dependencies will be highlighted on the canvas. Moreover, users can select the time range of interest by brushing the timeline at the horizontal or vertical boundaries of the canvas, and entities in the selected time range will be zoomed in to allow users to focus on their interests. When users select a vertex, its corresponding tasks will be highlighted in purple color as shown in Figure **??**.

Visualizing system performance statistics: As introduced in Section 4, several system performance statistics (e.g., CPU usage, network bandwidth) that affect query execution are collected. These metrics can be modeled as multi-dimensional time-series data. For example, for a machine with 12 CPUs, the CPU usage can be described by a temporal sequence with 12 dimensions. We use heatmap-based visualization to demonstrate the temporal usage of CPUs (as shown in Figure 7(D)). A color encoding from yellow to red is adopted to represent a CPU usage from 0 to 100%. An alternative is to visualize multi-dimensional time-

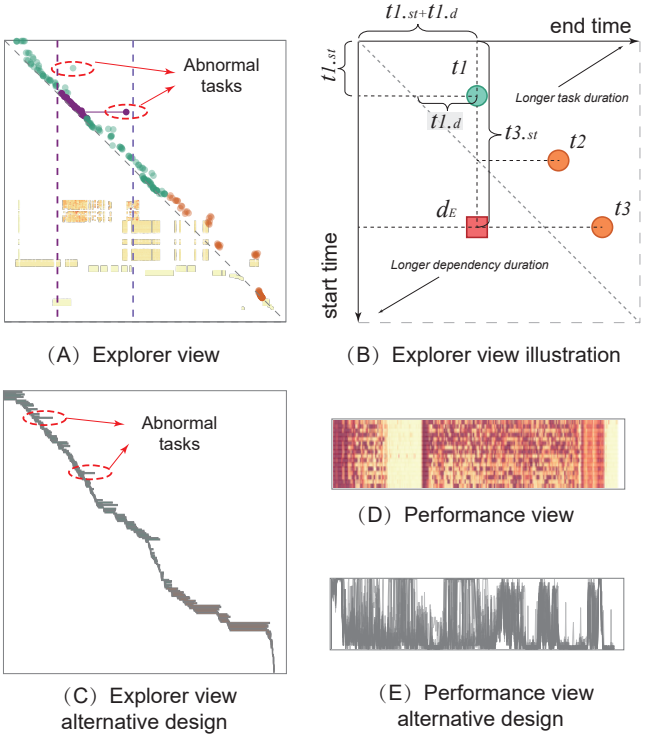

Fig. 7. Visual design for the pattern explorer.

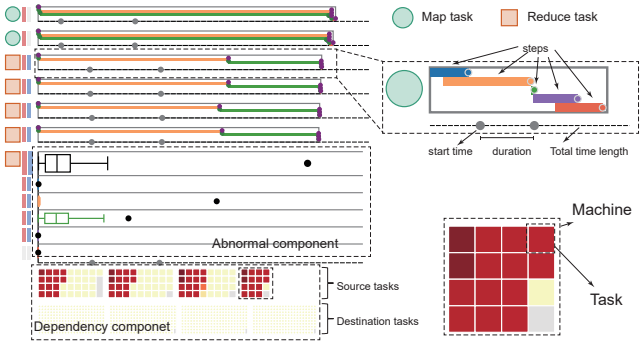

Fig. 8. Visual design for task list.

series data as line chart, which is more accurate than color encoding when the dimension is low. However, we observed that using line chart to show CPU usage (Figure 7(E)) results in a serious visual clutter due to the large number of lines. Similar phenomenon is observed for disk usage, which also contains more than 10 dimensions. Therefore, we use heatmap-based visualization for both CPU usage and disk usage. For other metrics such memory and network, we use line chart as the visualization method as their dimension is less than five. To support correlating task execution with system statuses, we set the same time scale for both the task distribution view and the performance metric view.

6.3 Task View

Task View is designed to enable fine-grained exploration of individual tasks. All tasks are listed from the top to the bottom according to their duration, and only the 150 tasks with the longest duration are shown by default. User can also sort the tasks by There are two visual forms for a task, i.e., glyph form and extension form. By default, the task glyphs are placed row by row. When the user selects a task of interest, its glyph will be extended into the extension form (as shown in Figure **??**).

6.3.1 Task Glyph

As shown in Figure 8, a green or orange rectangle is placed at the left side of each row to indicate the type of the task. To the right, a rectangle is used to represent a task and its length is proportional to the duration of the task (normalized by the longest task). In a rectangle, the five sub-steps of a task are shown line by line. We use both color and the y-coordinate to encode sub-step types. As the duration of a sub-step can be very close to 0, we place a circle at the left and right sides of the rectangle of a sub-step. Thus, an zero-length sub-step will be marked by the overlap of the two shapes.

6.3.2 Extension View

When the user clicks a task, an extension view is displayed below the task glyph as shown in Figure 8(**XX**). The extension view contains two components, i.e., abnormal view and dependency component.

In the abnormal view, we conduct anomaly detection for the tasks. As suggested by the domain experts, a task is likely to be abnormal if its duration is significantly longer than that other tasks that correspond to the same vertex in the logical DAG and run on the same machine. Based on this suggestion, we use the Tukey Fence method [34] to decide if a task is abnormal. For a set of real numbers S and a number $l \in S$, the anomaly score of l calculated as

$$AB(l,S) = \begin{cases} true, & \text{if } l > S.q_3 + 1.5 \times (S.q_3 - S.q_1) \\ false, & \text{otherwise} \end{cases}, \quad (1)$$

where $S.q_i$ is the i^{th} quartile of set S. Given a task and its corresponding logical vertex, we calculate the anomaly score for both the task a whole and its 5 sub-steps. With the anomaly score, we use six rows in the abnormal component. For a task t and its vertex v, the top five rows visualize the duration distribution of the five sub-steps in v using box plot: the left and right vertical lines indicate the minimum and maximum duration of the corresponding sub-steps, and the left and right sides of the rectangle indicate the 1^{st} and 3^{st} quartile. We also place a dot over the box plot to show the duration of task t. The last row use the same visual as for task duration.

The dependency view consists two rows representing the producer tasks and consumer tasks that are related to the selected task, respectively. As the number of consumer tasks and producer tasks can reach thousands, we use the pixel bar chart [21] due to scalability issues. As shown in Figure 7, we vertically divide the dependency view into segments to represent different machines. The consumer tasks or producer tasks are visualized as pixels in the boxes corresponding to machines that execute this task. The position of a dependent task is decided by the data size transmitted between it and the selected task. The dependent tasks are placed from left to right in descending order of their data transfer and color encoding (from dark red to light yellow) is also used to indicate the data size.

6.4 Interactions

To facilitate interactive exploration, QEVIS supports two types of cross-view interactions, i.e., cross-level interaction and temporal linkage.

Cross-level and cross-view interaction: QEVIS supports linking visual elements related to the same task. For example, if the user hovers the mouse on one task in the task view, the dot that corresponds to the task and its data dependencies of on the machine view will be highlighted by changing the color to purple. Moreover, the progress bar of the logical vertex associated with the task will be highlighted by changing the stroke width.

Temporal linkage: In both machine view and progress view, the user can select the time range to narrow down the focus of interest. If time range is selected in one view, other views will adjust their time ranges accordingly.

7 Evaluation

7.1 Case study

We collaborated with the domain experts to demonstrate the effectiveness of QEVIS by analyzing several real world cases which are slower than our exception.

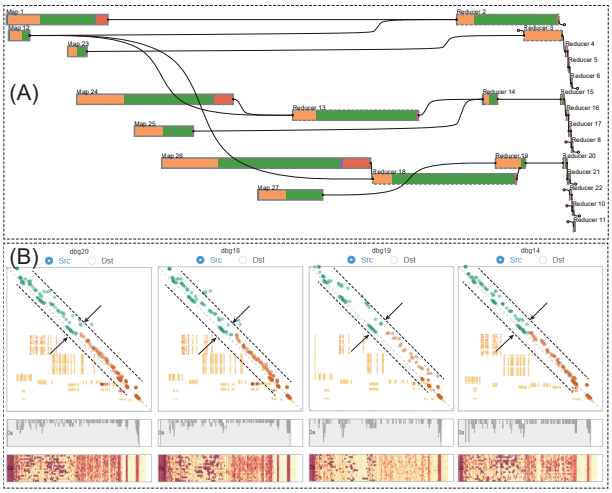

Fig. 9. Re-execute the query after removing a unhealthy node and stop the CPU-bound programs.

7.1.1 Identify the hardware bottleneck

The first case is executed on a cluster with 5 nodes and run around 623 seconds. The execution plan is shown as the Figure 1(A).

When exploring the execution with QEVIS, we find that in the progress view, it is obvious Map1(Figure 1(B1)) significantly run longer than the duration of other vertices. On the other side, the Reducer2, a subsequent vertex of Map1, finished in a very short time after Map1 is finished. Reducer is followed by a sequence of short reduce vertices(shown as Figure 1(B3)) which are quickly executed after that. From this pattern, we guess the bottleneck should be related Map1, several abnormal tasks may lead to the long execution time of Map1. We hover the mouse on it to highlight the associated tasks as blue color in the Distribution View. In the summary distribution view, we can see there are two task groups which are distributed far away from each other (shown as Figure 1(C5, C6)). By checking the blue-colored tasks on the machine views through cross-view linking, we find the tasks at the left top corner are dispatched on the machine dbg14, dbg16, dbg19 and dbg20. But the tasks on the right bottom corner are only executed on machine dbg18. Moreover, it seems that dbg18 only executes very few tasks in this case. We further check the cluster and notice that something wrong with the network of dbg18, thus the tasks assigned to it are delayed and executed very late, leading to the overall long duration of the query execution. In the exploration of summary view, we can find the these tasks of Map1 provides the data to several tasks of Reducer2, shown as the Figure 1(C1), which also verifies our assumption.

In addition to Map1, we notice another vertex Map24 also have a very long duration. When we click to mark the tasks of Map24 (these tasks are colored as purple), we find that none of them are executed on dbg18. So we guess dbg18 is not the only reason results in the slow query. At the same time, there is one task run with a very long time (shown as Figure 1(C2)) on the machine dbg19. Moreover, the dbg19 executes far fewer tasks than the machine dbg14, dbg16 and dbg20, but these tasks tends to have longer duration. By exploring the performance view, we find the CPU usage view of dbg19 has large piece of red color region, indicating the the CPU is more busy than the other machines(shown as Figure 1(C3)). We further explore the system log and find there are several computing-bounded programs are executed on dbg19 at the same time which takes the computing resource thus making these query tasks very slow.

To address this problem, we directly remove the node dbg18, stop the other programs run on server dbg19 and re-execute the same query. The query is finished with 200s. In the progress view, we find the duration of both vertex Map1 and Map24 are greatly reduced shown as Figure 9(A). We also find that in the distribution view, all the tasks are distributed close to the diagonal line indicating that they are all executed within the small time range Figure 9(B). Moreover, the CPU performance view shows that all the CPUs on different server have the balanced performance.

7.1.2 Diagnose the task failure

The second case runs on the cluster with four nodes. As show by the summary view, we notice that there are two tasks failed during the execution. The expert wants to know how why these tasks are failed. By observing the performance view, we notice during the execution of the failed tasks, the all the machine run the maximum number of tasks but the usage of their CPU, memory and network are not fully used, which indicating these tasks take the quota of the works but do nothing. but cannot finished until two tasks are abandoned. ***

7.1.3 Understand the data skew

7.2 Expert interview

8 Conclusion

In this paper, we propose QEVIS, an interactive visual system for Hive query execution analysis. QEVIS consists of three linked views which transfer the multi-level progress data into different visual forms according to the analyzing requirements, including: 1) a space-efficient algorithm to layout the TDAG, which displays the progress and the clear topology structure simultaneously; 2) a novel visual encoding method to visualize the complex task dependencies and task distribution, system performance is also integrated with it to allow users to reason the query patterns; 3) a task view for individual task analysis. Three case studies about the query execution understanding, resource bottleneck detection and query debugging as well as two user interviews show our system can be helpful in real-world practices.

References

[1] A. Abouzied, J. Hellerstein, and A. Silberschatz. Dataplay: interactive tweaking and example-driven correction of graphical database queries. In *Proceedings of the 25th annual ACM symposium on User interface software and technology*, pp. 207–218, 2012.

[2] B. C. Cappers and J. J. van Wijk. Exploring multivariate event sequences using rules, aggregations, and selections. *IEEE transactions on visualization and computer graphics*, 24(1):532–541, 2017.

[3] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas. Apache flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 36(4), 2015.

[4] C. Cerullo and M. Porta. A system for database visual querying and query visualization: Complementing text and graphics to increase expressiveness. In *18th International Workshop on Database and Expert Systems Applications (DEXA 2007)*, pp. 109–113. IEEE, 2007.

[5] Q. Chen, Y. Chen, D. Liu, C. Shi, Y. Wu, and H. Qu. Peakvizor: Visual analytics of peaks in video clickstreams from massive open online courses. *IEEE transactions on visualization and computer graphics*, 22(10):2315–2330, 2015.

[6] Q. Chen, X. Yue, X. Plantaz, Y. Chen, C. Shi, T.-C. Pong, and H. Qu. Viseq: Visual analytics of learning sequence in massive open online courses. *IEEE transactions on visualization and computer graphics*, 26(3):1622–1636, 2018.

[7] W. Clark. *The Gantt chart: A working tool of management*. Ronald Press Company, 1922.

[8] J. Danaparamita and W. Gatterbauer. Queryviz: helping users understand sql queries and their patterns. In *Proceedings of the 14th International Conference on Extending Database Technology*, pp. 558–561, 2011.

[9] A. S. Foundation. The apache hadoop project.

[10] A. S. Foundation. Tez ui.

[11] S. Gathani, P. Lim, and L. Battle. Debugging database queries: A survey of tools, techniques, and users. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pp. 1–16, 2020.

[12] W. Gatterbauer. Databases will visualize queries too. *Proceedings of the VLDB Endowment*, 4(12):1498–1501, 2011.

[13] M. Gawade and M. Kersten. Stethoscope: a platform for interactive visual analysis of query execution plans. *Proceedings of the VLDB Endowment*, 5(12):1926–1929, 2012.

[14] D. Gotz, J. Zhang, W. Wang, J. Shrestha, and D. Borland. Visual analysis of high-dimensional event sequence data via dynamic hierarchical aggregation. *IEEE transactions on visualization and computer graphics*, 26(1):440–450, 2019.

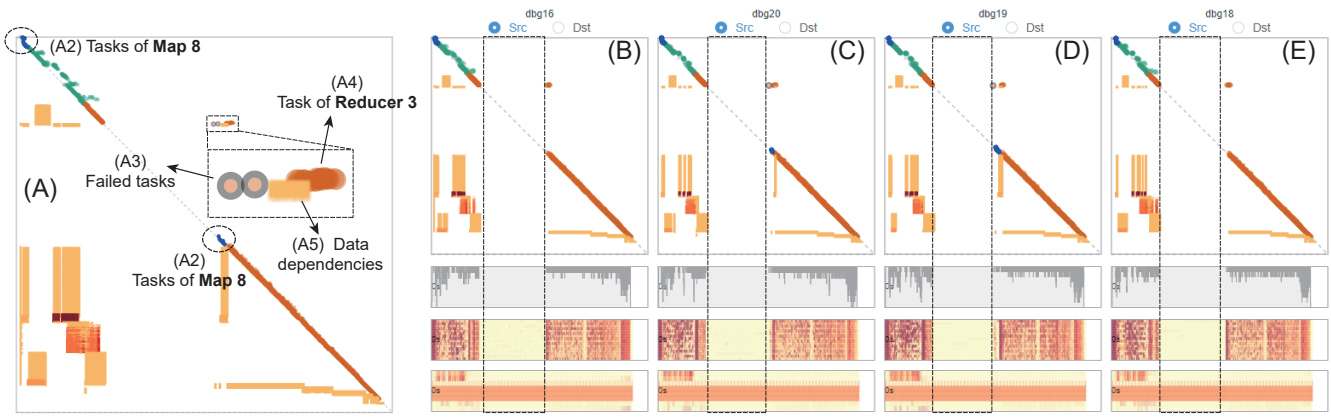

Fig. 10. Analyze the task failure.

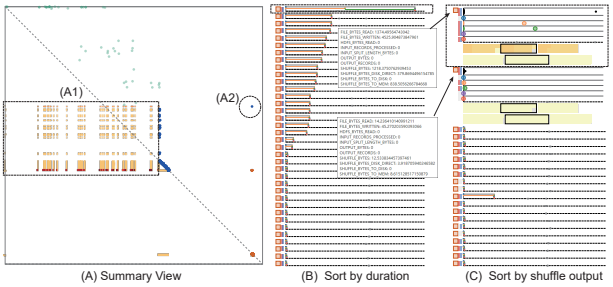

Fig. 11. Detect the data skew.

[15] M. C. Goulden, E. Gronda, Y. Yang, Z. Zhang, J. Tao, C. Wang, X. Duan, G. A. Ambrose, K. Abbott, and P. Miller. Ccvis: Visual analytics of student online learning behaviors using course clickstream data. *Electronic Imaging*, 2019(1):681–1, 2019.

[16] Y. Guo, S. Guo, Z. Jin, S. Kaul, D. Gotz, and N. Cao. Survey on visual analysis of event sequence data. *arXiv preprint arXiv:2006.14291*, 2020.

[17] D. Halperin, V. Teixeira de Almeida, L. L. Choo, S. Chu, P. Koutris, D. Moritz, J. Ortiz, V. Ruamviboonsuk, J. Wang, A. Whitaker, et al. Demonstration of the myria big data management service. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pp. 881–884, 2014.

[18] H. He, B. Dong, Q. Zheng, and G. Li. Vuc: Visualizing daily video utilization to promote student engagement in online distance education. In *Proceedings of the ACM Conference on Global Computing Education*, pp. 99–105, 2019.

[19] H. Jaakkola and B. Thalheim. Visual sql–high-quality er-based query treatment. In *International Conference on Conceptual Modeling*, pp. 129–139. Springer, 2003.

[20] J. Jo, J. Huh, J. Park, B. Kim, and J. Seo. Livegantt: Interactively visualizing a large manufacturing schedule. *IEEE transactions on visualization and computer graphics*, 20(12):2329–2338, 2014.

[21] D. A. Keim, M. C. Hao, U. Dayal, and M. Hsu. Pixel bar charts: a visualization technique for very large multi-attribute data sets. *Information Visualization*, 1(1):20–34, 2002.

[22] G. Koutrika, A. Simitsis, and Y. E. Ioannidis. Explaining structured queries in natural language. In *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, pp. 333–344. IEEE, 2010.

[23] P.-M. Law, Z. Liu, S. Malik, and R. C. Basole. Maqui: Interweaving queries and pattern mining for recursive event sequence exploration. *IEEE transactions on visualization and computer graphics*, 25(1):396–406, 2018.

[24] A. Leventidis, J. Zhang, C. Dunne, W. Gatterbauer, H. Jagadish, and M. Riedewald. Queryvis: Logic-based diagrams help users understand complicated sql queries faster. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pp. 2303–2318, 2020.

[25] S. Malik, F. Du, M. Monroe, E. Onukwugha, C. Plaisant, and B. Shneiderman. Cohort comparison of event sequences with balanced integration of visual analytics and statistics. In *Proceedings of the 20th International Conference on Intelligent User Interfaces*, pp. 38–49, 2015.

[26] S. Malik, B. Shneiderman, F. Du, C. Plaisant, and M. Bjarnadottir. High-volume hypothesis testing: Systematic exploration of event sequence comparisons. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 6(1):1–23, 2016.

[27] D. Moritz, D. Halperin, B. Howe, and J. Heer. Perfopticon: Visual query analysis for distributed databases. In *Computer Graphics Forum*, vol. 34, pp. 71–80. Wiley Online Library, 2015.

[28] X. Mu, K. Xu, Q. Chen, F. Du, Y. Wang, and H. Qu. Moocad: Visual analysis of anomalous learning activities in massive open online courses. In *EuroVis (Short Papers)*, pp. 91–95, 2019.

[29] C. Plaisant, B. Milash, A. Rose, S. Widoff, and B. Shneiderman. Lifelines: visualizing personal histories. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 221–227, 1996.

[30] P. Riehmann, M. Hanfler, and B. Froehlich. Interactive sankey diagrams. In *IEEE Symposium on Information Visualization, 2005. INFOVIS 2005.*, pp. 233–240. IEEE, 2005.

[31] B. Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *The craft of information visualization*, pp. 364–371. Elsevier, 2003.

[32] S. F. Silva and T. Catarci. Visualization of linear time-oriented data: a survey. In *Proceedings of the first international conference on web information systems engineering*, vol. 1, pp. 310–319. IEEE, 2000.

[33] A. Simitsis, K. Wilkinson, J. Blais, and J. Walsh. Vqa: vertica query analyzer. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pp. 701–704, 2014.

[34] J. W. Tukey et al. *Exploratory data analysis*, vol. 2. Reading, Mass., 1977.

[35] K. Wongsuphasawat and D. Gotz. Outflow: Visualizing patient flow by symptoms and outcome. In *IEEE VisWeek Workshop on Visual Analytics in Healthcare, Providence, Rhode Island, USA*, pp. 25–28. American Medical Informatics Association, 2011.

[36] K. Wongsuphasawat, J. A. Guerra Gómez, C. Plaisant, T. D. Wang, M. Taieb-Maimon, and B. Shneiderman. Lifeflow: visualizing an overview of event sequences. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pp. 1747–1756, 2011.

[37] J. Wu, Z. Guo, Z. Wang, Q. Xu, and Y. Wu. Visual analytics of multivariate event sequence data in racquet sports. In *2020 IEEE Conference on Visual Analytics Science and Technology (VAST)*, pp. 36–47. IEEE, 2020.

[38] J. Zhao, N. Cao, Z. Wen, Y. Song, Y.-R. Lin, and C. Collins. # fluxflow: Visual analysis of anomalous information spreading on social media. *IEEE transactions on visualization and computer graphics*, 20(12):1773–1782, 2014.

[39] J. Zhao, Z. Liu, M. Dontcheva, A. Hertzmann, and A. Wilson. Matrixwave: Visual comparison of event sequence data. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pp. 259–268, 2015.