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

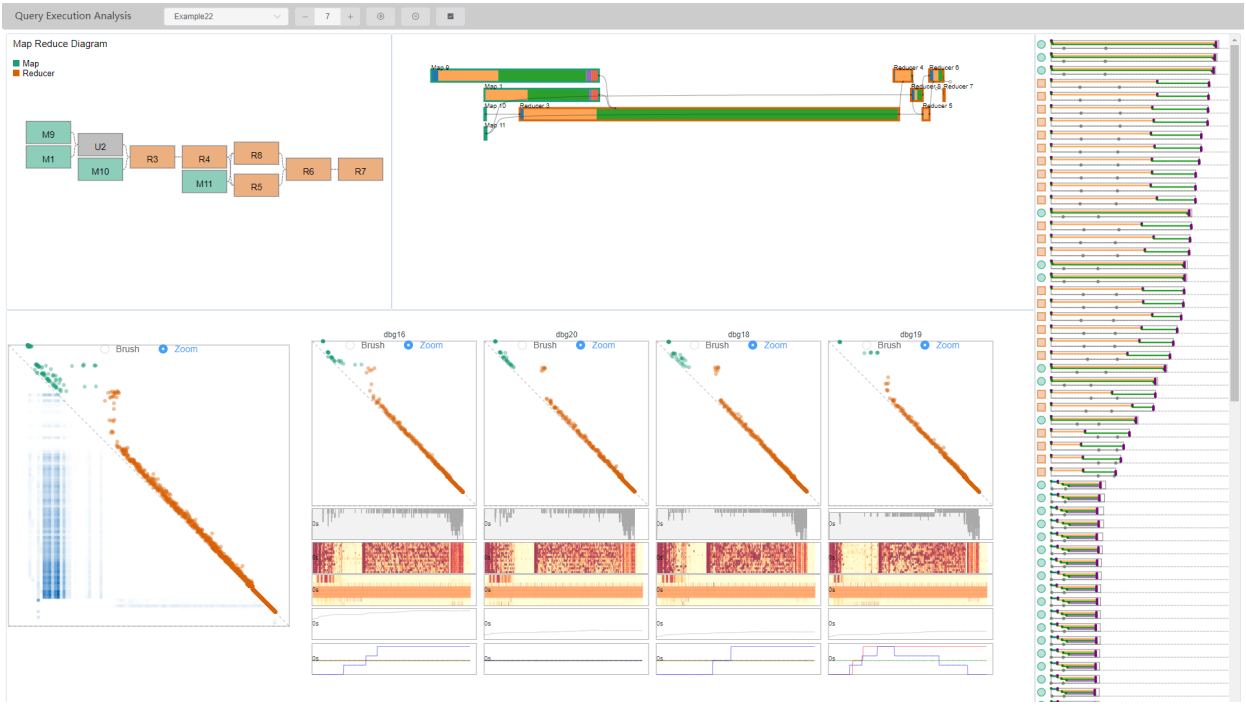

Fig. 1. In the Clouds: Vancouver from Cypress Mountain. Note that the teaser may not be wider than the abstract block.

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) 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

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

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. 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.

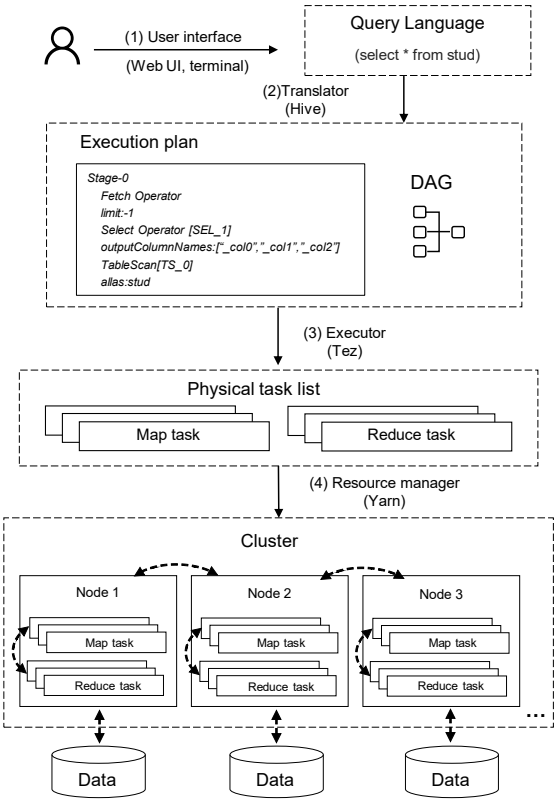

Fig. 2. The query processing workflow of Hive (Hadoop 2.0).

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 Visualize 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 the query process at the task level. Tez task is the atomic level of executions because ny failure operation in the task

will lead to the re-run of whole task. Understand the execution of single task can be helpful to identify the bottleneck of the whole query process. However, visualize the tasks is challenge. First, to visualize the tasks in traditional way(Gantt chart) need a very large rendering space. Multiple features such as the size of input/output data and the operators should be visualized for understanding the task. Moreover, the many to many relationship among the tasks also makes it difficult to design clear and **scaleble** visualization.

R3 Provide the visual insight to reason the behaviour and pattern of a specific task. To solely visualize the tasks themselves are not enough to explain the specific pattern of tasks. Many performance of hardware resource such as the network status, hard disk waiting list is also related to the patterns. Such kinds of information should be vitalized effectively to assist the exploration of query executions.

R4 Support interactive exploration. Other than the visualization designs, a flexible interaction should be implemented for users to navigate to any time range, vertex, task group or single task of interest. The linkage among the correlated visual elements are also should be considered om the design to coordinate the information.

4.2 Task Analysis

Guided by the aforementioned requirements, we discussed with the domain experts about the visualization form and distilled the following visualization tasks:

T1 Visualize the execution process and query plan structure effectively. To guarantee the continuity of exploration(R1), the process and plan structure should be integrated into one visualization view. Several criteria should be considered such as the minimize usage of canvas, minimize the cross of links and provide clear topology structure.

T2 Effectively visualize the information of tasks. To facilitate the fine grained exploration of query execution(**R1**, **R2**), the information about the tasks should be visualized, including: the size of data processed by the task; the data-flow among the tasks; the temporal information of task(start the time, end time, duration, etc) and the corresponding sub-process. Moreover, the abnormal(tasks taking longer time) tasks and the specific execution trace should be easily observed.

T3 Visualize the machine status. Display the machine status such as network status, disk IO pending list, CPU usage and Memory Usage will be useful to investigate the characters of task, and reasoning the patterns of the query execution(**R3**). These performance metrics should be well linked with the specific patterns of tasks.

T4 Interaction and linkage. System should provide the flexible interactions allowing users to switch the focus among the different point of interest, such as a specific time range, a vertex or a group of tasks(**R2**, **R3**, **R4**). For example, user may select a vertex and explore if the tasks in this vertex are CPU-bound or I/O-bound. This requires the visualization to show the related tasks when choosing a vertex and highlight the corresponding CPU usage and disk information simultaneously.

4.3 System Architecture

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

The **Backend Manager** module performs the log processing and data fusion. Two categories of data are collected and fused: the execution data and the system performance data. When collecting the

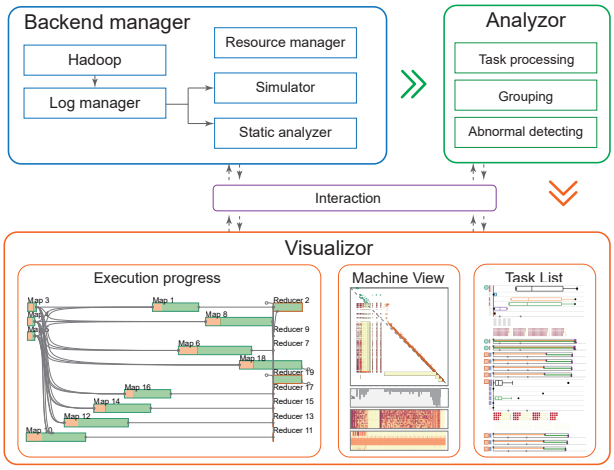

Fig. 3. QEVIS system consists of three major components: backend manager, analyzer and visualizer.

execution data from Hadoop system, the Log Manager cleans and preprocesses the log files, extracts the important metrics and saves them as the local files. Log analyzer directly takes these files as input and processes them as structured data for future processing. Monitor module segments the data at the specific time range and output them with a given rate. The simulator simulates the execution progress, allowing users to adjust the running speed and explore the dynamic query process. Moreover, Resource Manager collect the system performance metrics and output 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 the vertex of the execution plan. The dataflow dependencies are recorded in this step. Moreover, Analyzor also performs the anomaly detection for the tasks in a group to enable the in-depth analysis.

The **Visualizor** module integrates coordinated views to support interactive exploration of query execution results and reasoning about the query behavior at multiple levels. The Execution Overview demonstrates the execution process at the vertex level. An algorithm for the temporal DAG is proposed to visualize the structure and procedure simultaneously. The task group view consists of two components: 1) task overview visualizes the temporal information and data dependencies of tasks executed on the same machine; 2) metrics component shows the corresponding machine performance metrics. The task list view provides more detailed information at the operator level, enabling the users to understand and compare the time usage of tasks.

5 Preprocess and Data Analysis

5.1 Data Collection

As shown by figure 3, our analysis pipeline starts from the plan, log, and system performance data collection.

Query plan parsing: As shown in section 3, in Hive, an execution plan is described as a text file involving hundreds to thousands of lines of operations. We parse this file first and generate a directed acyclic graph with the node as the logic vertex and the edge as the data dependencies.

Execution log and system performance processing: Since the Hadoop execution logs are collected with the tedious system log, we locate the records of interest by detecting the keywords from the predefined keyword set and then parse these logs and extract the information. Several important information is saved to the local file, including the task id, the logic vertices corresponding to the tasks, the temporal information(start time, duration) of tasks, the temporal information of steps in a task, etc.

5.2 Data Modeling

The data we collected from backend can be modeled as a temporal graph with two levels: the logic-level and execution-level, which is shown as the figure 4.

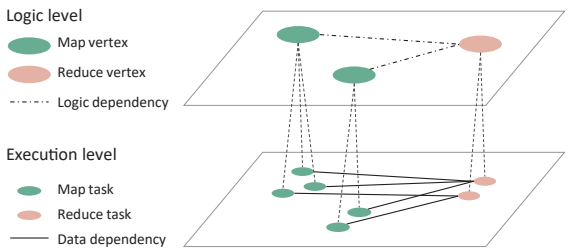

Fig. 4. Two-level temporal graph.

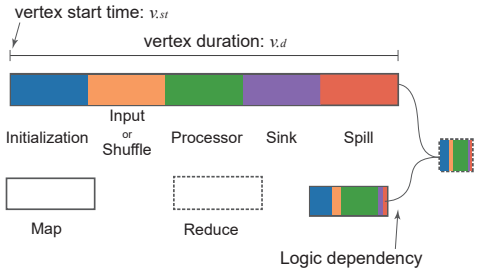

Fig. 5. Visual encoding of logic vertex and dependency

As discussed in section 3, the logic level graph indicates the DAG extracted from the execution plan, denoted by $\mathbb{G}_L = (\mathbb{V}_L, \mathbb{D}_L)$, where $\mathbb{V}_L$ is the logic vertex set and the $\mathbb{D}_L$ is logic dependency set between the vertices. The execution-level graph denotes the $\mathbb{G}_E = (\mathbb{T}_E, \mathbb{D}_E)$, where $\mathbb{T}_E$ denotes the set of tasks executed by the the physical machines and $\mathbb{D}_E$ indicates the data dependency set between tasks. If a task $t \in \mathbb{T}_E$ is an physical instance of vertex $v \in \mathbb{V}_L$, we describe this relationship as the form $t \to v$. We also adopt the this description for the relationship between logic dependency $d_L \in \mathbb{D}_L$ and data dependency $d_E \in \mathbb{D}_E$ such as $d_e \to d_L$. Moreover, we use $P(v) = \{t | \forall t \to v\}$ to indicate all tasks which are the physical instances of $\mathbb{V}_L$. A map task t has five steps indicating as an array: $S =< s_{init}, s_{input}, s_{proc}, s_{sink}, s_{spill} >$ and a reduce task have five steps $S =< s_{init}, s_{shuffle}, s_{proc}, s_{sink}, s_{spill} >$. Each step can be modeled by a pair of attributes $< st, d >$ which denotes the start time and durzation. Notice that the different steps of the same task may have overlap in time.

According to section 3, each task can be modeled as a sequence of attributes: $t :=< st, d, v, m, S_E >$, where st, d and v indicate the *start time*, *duration* and logic vertex of task t. m is the machine executes this task and S_T is the corresponding steps. We use $t.attr$ to indicate the $attr$ of t.

The vertex can be modeled as 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 e is $v.d = v.st + max(\{(v.st + v.d) | t \in P(v)\})$. 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 types, and with 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 the web-based visual analytics system to support the interactive exploration with four coordinated views. The Execution progress demonstrates the overview about how the query plan are execute(**T1**), the Task distribution view shows the task distribution and the data dependencies. Integrated with the machine performance metrics, this view is also used for reason the specific patterns of tasks(**T2** and **T3**). Task list provides the detailed information at the task level(**T2**). At last, the interaction and linkage are introduced to support the multi-level explorations(**T4**).

6.1 Query Progress View

Query Progress View is developed to overview the overall progress of query execution and the logic dependencies.

Algorithm 1 TDAGLayout(R, *canvas*)

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

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
```

6.1.1 Visual Encoding

The traditional method to visualize the progress data is the Gantt Chart [7] which maps the time interval on the horizontal axis and lists the progresses bar on the vertical axis. As shown by Figure 5, given a vertex v, the x-coordinate of the bar's left side indicates the start time $v.st$ and the duration of $v.d$ is encoded by the length of the rectangle. We use the stroke dash to present the type of vertex and use the color to encode the type of steps in a vertex. Moreover, the order of all steps is fixed as shown by Figure 5, and we use B-splines to show the dependencies.

6.1.2 Layout Method

Gantt Chart has been used in many progress visualization tools such as Tez UI and Tabula, in which each progress bar takes up the whole line, which is shown as Figure 6(A). Vertically, the progress bars are ordered by the start time of the vertex. In this visual design, since the layout doesn't consider the logic dependencies, it always has a very serious cross between the links and unclear topology structure.

To tackle this problem, we design an algorithm 1 to layout the TDAG. Notice that since the x-position and width of the bars have been fixed to encode the temporal information, only the vertical position can be changed to optimize the layout. With 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* as the vertex which has no parents. process is described as Algorithm 2, which use a in-order traversal to process the vertices in the graph. place is the function which tries to place the vertex in the canvas. At first, we use a naive which provides a vertex an exclusive row, which is shown as Figure 6(B). The new layout shows a clear topology structure because the brother vertices are visited closely. However, this method also results in a poorly using of the canvas because each vertex will take the whole row.

To provide a tight layout, we revise the method place which tries to place the vertex at the top position of the canvas, if the vertex is overlapped with the existing vertex, then move this vertex down the distance of the vertex height. If the place cannot find a place for the vertex in the canvas, the place will return *False* indicating the height of canvas is not enough, otherwise return *True*. Each parent vertex will be visited multiple times until it is successfully placed into the canvas(shown as line 10 to line 14). The result(Figure 6(C)) shows a

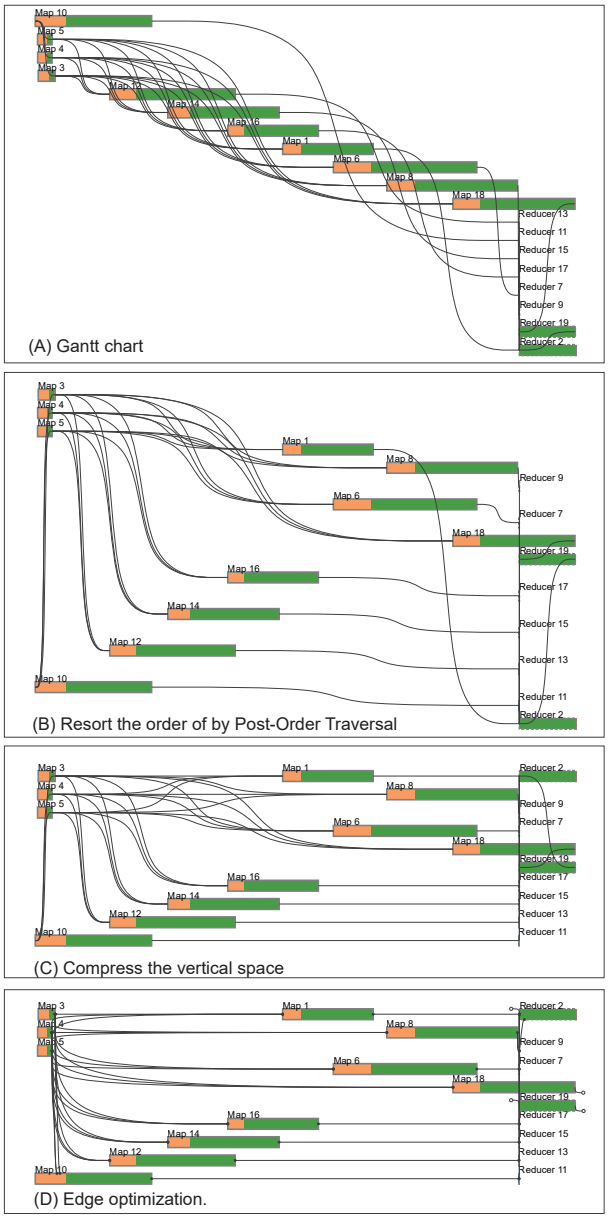

Fig. 6. Temporal DAG layout algorithms

layout which requires a smaller canvas than Figure 6(B).

The tight layout further results in an unclear structure of the dependencies because it compress the space to put the links. We further optimized it by changing the position of the control points of B-splines. With two different source vertex, the x-position of the control points are different, thus making the links from different source vertices are attracted to different position. The result is shown as Figure 6(D).

6.2 Pattern Explorer

Pattern Explorer is developed to provide efficient pattern discovery and reasoning at the task-level. This component consists of multiple coordinated views, including Distribution View and Performance View.

6.2.1 Distribution View

The Distribution View is designed to explore the temporal pattern and dependencies of tasks. **Visualizing the temporal information of tasks:** Before our collaboration, the domain experts use Gantt Chart Diagram to display the overall progress of tasks, shown in figure 7(C). However, Gantt Chart based method suffers significant scalability problem in our applications. Hundreds to thousands of tasks are associated with a vertex in our scenario, which requires a very large space to place all the

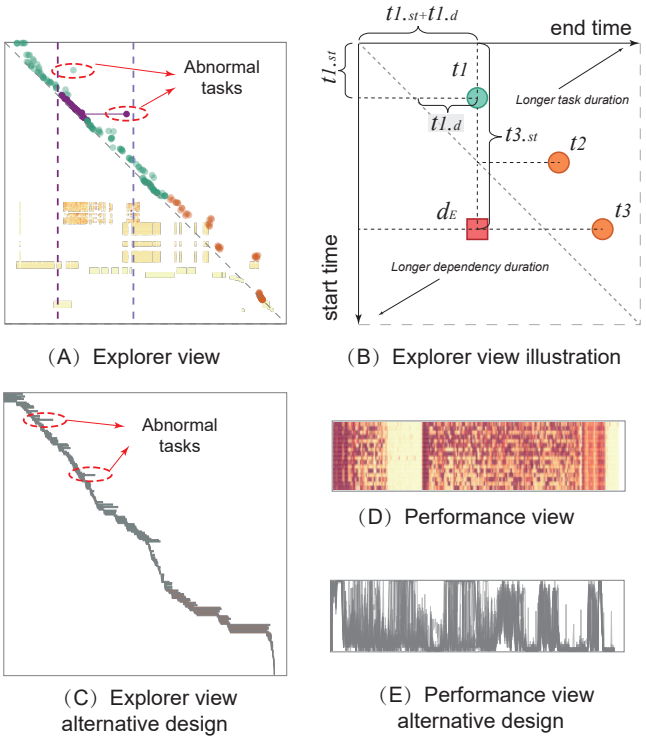

Fig. 7. Visual design for the pattern explorer.

horizontal bars clearly. Moreover, visualizing a large number of bars in this way is difficult for users to compare the absolute length of the horizontal bars due to the lack of alignment, which hinders the users' ability to discover the group-based patterns or identify the abnormal tasks.

To tackle this issue, we develop a scatter-based visual representation which is shown as Figure 7(A). Figure 7(B) illustrates the visualization design which uses a square shape of rendering canvas as the basis, the vertical axis(from top to bottom) indicates the start time, and the horizontal axis(from left to right) indicates the end time. The task(e.g., t_1, t_2, t_3) with the temporal information of start time and duration can be visualized as dots layouted on the canvas. This design has two benefits: 1) we simplify each horizontal bars as dots. Thus the all tasks can be presented as the point cloud in the canvas. This presentation form may results in the visual clutter caused by the gathering and overlap of dots, but can significantly highlight the clusters and outliers, which can help the users to find the task of interest; 2) we linearly map the time range to both horizontal and vertical axis. In this way, as shown by task t_1 in figure 7(B) the horizontal distance from point to the diagonal line of canvas represents the duration(e.g., $t_2.d$) of the task, which helps the users to compare the task duration. Generally speaking, in this view, if a task has a long duration, it tends to move to the top right corner.

We compare our design with Gantt Chart shown as 7(C), we scale the height of the bars of Gantt Chart to visualize the all tasks in the view with same size of our design. It is found that in our design, the outliers are more easily observed and the overall temporal distribution is more clear in our design than Gantt Chart.

Visualizing the task dependencies: The direct way to visualize the data dependencies is to use the curves such as the Progress View(section 6.1). However, such design will result in a serious visual clutter when dealing with the large data size. To solve this problem, we also transform the dependencies as a dot on the render canvas. As shown by Figure 7(B), the dependency d_E from task t_1 to t_3 is visualized as a dot(red color rectangle) in the left bottom part of canvas, which presents the end time or producer task and the start time of consumer task. To further deal with the scalability problem, we use heatmap instead of point cloud to improve the render efficiencies.

This design utilizes the left half part of canvas and showing the

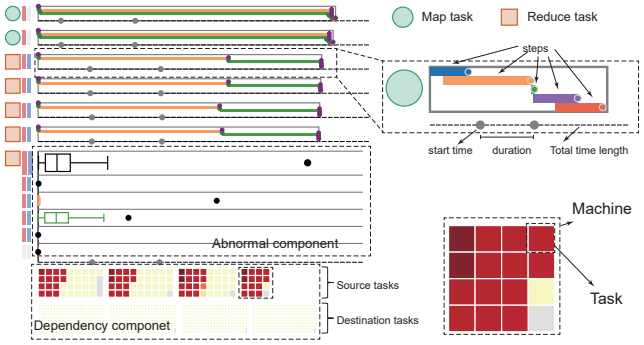

Fig. 8. Visual design for task list.

distribution of dependencies according to their temporal information, allowing the users to discover the long dependency duration clusters.

To facility the interactive explorations, several interactions are implemented in this view. For example, once a task is selected, the corresponding dependencies will be highlighted on the canvas. Moreover, users can select the time range of interest by brush the a timeline at the horizontal or vertical boundaries of canvas,then the time range of this view will be updated to allow users to switch their focus. Moreover, when the users select a vertex, the associated tasks will be highlighted in purple color, which is shown as Figure **??**.

Visualizing system performance metrics: As introduced in section 4, several metrics affecting the query performance are collected. These metrics can be modeled as multi-dimensional time-series data. For example, a machine may have 12 CPUs, thus the usage of the CPU can be modeled as a temporal sequence with 12 dimensions. We use the heatmap-based visualization to demonstrate temporal trend of CPUs(shown as Figure 7(D)). We use the color range from yellow to red to encode the CPU usage from 0 to 100%. Another visualization form for the multi-dimensional time-series data is linechart, which can provide more accurate visualization than color encoding when the dimension number is small. But when we use linechart to show the CPU usage(shown as Figure 7(E)), the lines results in a serious visual clutter which cannot deliver useful information at all. For all the machine metrics, CPU usage and Disk information both contains more than 10 dimensions, which we use heatmap-based visualization. For other metrics such Memory and Network, the number of dimensions is less than five and we use the linechart as the visualization methods. To support the correlation discovery between the patterns in task distribution and system performance metrics, we set the same time scale for both the task distribution view and performance metric view.

6.3 Task List View

Task list view is developed to enable the detailed exploration of the individual tasks. All the tasks will be listed from the top to the bottom according to the duration, and only the top one hundred tasks are shown by default.There are two visual forms of a task: glyph form and extension form. By default, the task glyph are placed row by row. When the user select the task of interest, the glyph will be extended as the extension form(shown as **??**).

6.3.1 Task glyph

As shown by Figure 8, the green circle or orange rectangle is placed at the left side of a row, indicating the type of the task. On the right side, a rectangle is shown with the length to encode the relative task duration to the maximum duration for all tasks. In the rectangle, the five steps are shown line by line. We use both the color and the y-position to encode the step type. Moreover, the length of a step can be very close to 0, which makes the current visualization unobservable. To tackle this problem, we place a circle and a triangle at the left and right sides of the rectangle of a step, respectively. Thus, the zero-length step will be marked by the overlap of the two shapes.

6.3.2 Extension View

When the user clicks the task, an extension view is displayed below to the task glyph, shown as Figure 8(**XX**). There are two sub-components in this view: abnormal component and dependency component.

We first conduct the abnormal detection for the tasks. As suggested by the domain experts, a task is abnormal when its duration is significantly longer than that of other tasks associated with the same vertex and executed on the same machine. Based on these suggestions, we use Tukey Fence [34] to decide if a task is abnormal. With a given set of real integer S and $l \in S$, the anomaly is calculated as follows:

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

Where $S.q_i$ is the i^{th} quartile of set S. With the given task and the vertex, we conduct the abnormal detection for the duration of the task and each step.

After calculation, we design the abnormal component with six rows. With a given vertex v and the task t. The top fives row visualize the distribution of the five steps in v as boxplot: the left and right vertical lines indicate the minimum and maximum duration of corresponding steps, and the left and right sides of the rectangle indicate the 1^{st} and 3^{st} quartiles. We also place a dot over the boxplot to show the duration of task t. The last row is implemented to use the same visual form which is used for the presentation of task duration.

The dependency components consists two rows representing the producer tasks and the consumer tasks respectively. With a given task, the number of its consumer tasks and producer tasks may reach thousands of tasks. Considering the scalability issues, we use the pixel barchart [21] to visualize data amount of dependencies. For example, as shown in Figure 7, we vertically divide the dependency component into segments to present the machines. The consumer tasks or producer tasks are visualized as the pixels in machine box executing this task, the position of task is decided by the data size transmit between the tasks, which is layouted from left to right and from top to bottom. The color is also used to encode the data size.

6.4 Interactions

To facilitate the interactive exploration, out system supports the cross-view interactions. In summary, there are two categories of interactions are implemented in the system: cross-level interactions and temporal linkage.

Cross-level and -view interactions:: Our system supports linking among visual elements related to the same task. For example, when hovering the mouse on the task of Task View, the corresponding dots of this task and the data dependencies at the machine view will be highlighted by changing the color to purple. Moreover, the progress bar of vertex associated with the task will also be highlighted by change the stroke width.

Temporal linkage: In both machine view and progress view allow, users can select the time range to narrow down to the pattern of interest. When select time range at any view, the time focus of other views will be updated.

7 Evaluation

7.1 Case study

7.2 Expert interview

8 Conclusion

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.

[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.