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

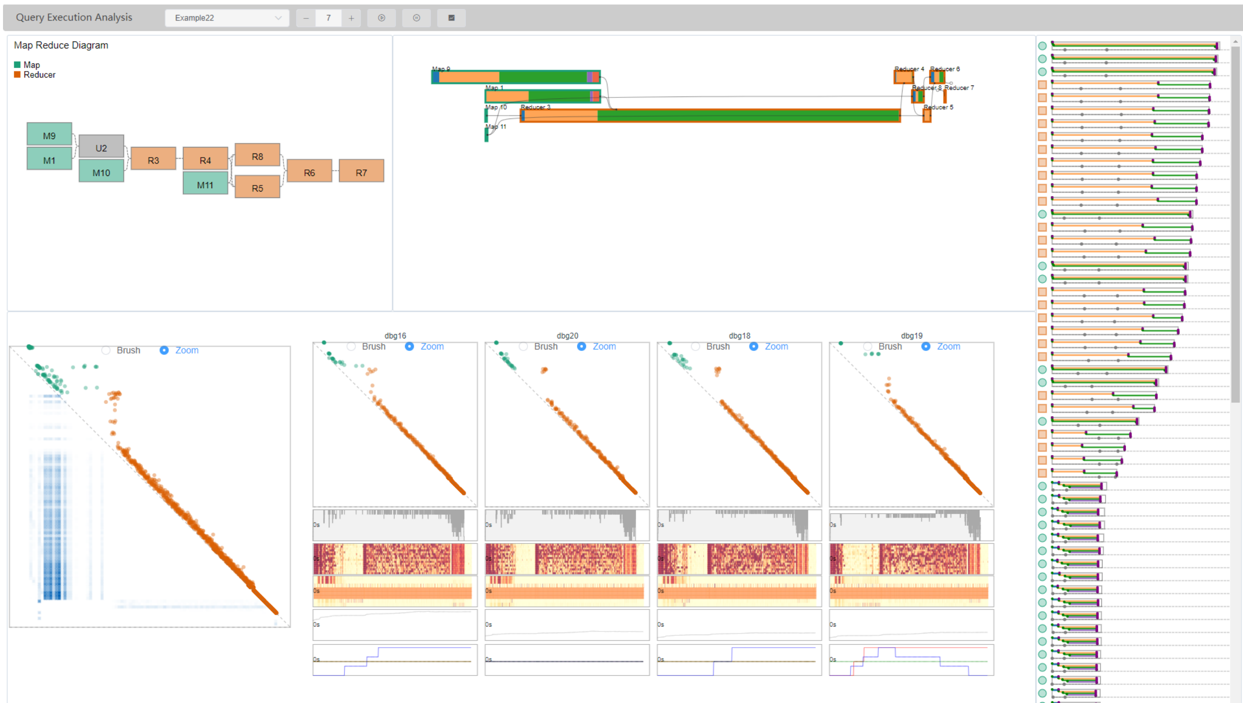

Fig. 1. In the Clouds: Vancouver from Cypress Mountain. Note that the teaser may not be wider than the abstract block.

Abstract— Understanding the query execution of distributed databases is crucial to many real-world practices such as detecting the query bottleneck and improving the system performance. Such analysis is always challenging due to the large volume of tasks executed in parallel and the complex task dependencies. Moreover, the unpredicted system behaviors also affect the query execution case by case. Existing techniques usually evaluate the distributed query by the statistics of system performance (e.g., CPU, memory, and disk I/O) or execution metrics(e.g., operator duration), which lost the fine-grained information at the atom task level. To tackle this problem, we propose QEVIS, a visual analytics system to interactively understand and diagnose the distributed query execution procedure from multiple levels. An optimized algorithm for the temporal directed acyclic graph(TDAG) layout is devised to show the overall query plan structure and execution process. A suite of novel visualization and interaction designs are integrated to estimate the correlation between the atom task trace and the system performance metrics. We illustrate the effectiveness of our QEVIS with three case studies from real-world applications and interviews with domain experts.

Index Terms—Radiosity, global illumination, constant time

1 Introduction

The distributed database system is becoming increasingly pervasive due to the explosive growth of data in science, industrial, life, etc. Meanwhile, many tools such as Hadoop [5], Flink [2], Myria and Vertical are developed to facilitate the query execution procedure including optimizing the query, translating traditional query language to logic execution plan, generating and dispatching tasks to clusters to perform the data acquisition in parallel.

To maximally leverage the distributed systems, it is crucial for users to understand and evaluate how the query runs across the clusters. The frequently asked questions include ”Where does the time go?”, ”What is the bottleneck of my query?”, ”Can we improve the performance of the specific query?”. Many research work devoted to evaluating and improving the performance of data analytic frameworks, but most of them try to reveal the performance by making high-level statistics about the correlated metrics collected from the execution logs or experiment conducted on benchmarks, which cannot be used for the understanding of the special case and provide the details answer for these questions case by case.

Understanding query logic and execution in the distributed environment is challenging. Three stages are involved for human-beings to understand a database query comprehensively: 1) understand the query logic, 2) understand the execution plan structure, and 3) understand the plan execution procedure. Understand the query logic itself is long studied direction, especially when the queries are becoming increasing

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

deep and nested. With the pervasive of distributed database system, visual explaining of distributed execution plan structure is studied. These methods transform the abstract execution plan with thousands of lines of description to intuitive diagrams such as tree or graph and design interactions allowing users to interactively explore the nested structures. In our paper, we mainly focus on the stage 3) i.e. understand the execution procedure of query execution.There are three challenges to facilitate the fine-grained analysis of query executions.

Large number of atom tasks: When issuing the execution plan on the distributed database system, large number of atomic tasks will be generated and executed on the nodes in the cluster in parallel. The duration of tasks may be significant different from each other even executed on the same server. Identifying the unnature long cases and analyzing their performance from the large amount of tasks is challenges, since it is difficult to define clear ground truth to fits all cases.

Complex dependencies among the tasks: The processing of a task always depends on several prerequisite tasks which provide the necessary input data. Analyzing the complex many to many dependencies and identifying the trace of interested are difficult in the large task set.

Unpredicted behavior of distributed system: also increases he difficulty to understand the model execution procedure. For instance, the developer find that the same query execution plan run today may be different from that of yesterday. In general, four aspects are considered to affect the performance of clusters: CPU usage, memory usage, network IO and disk status. Existing work studies try to reveal how these metrics related to the system performance or quantify the impact and significance of these features. These studies are conducted based on the observed performance data from the experiment or logs collected from the production environment. One work inspired us is QVA which tries to linkage the resources status to query performance and resource usages. However, these work fail to provide the fine-grained execution traces for users to inspect the reasons of model behavior.

In this work, we develop a visual analytics system called QEVIS (Figure 1) for database users to understand and diagnose query behavior on the distributed database. The design of QEVIS follows the visualization Mantra "overview first, zoom and filter, details on demand" [21]. The system consists of three coordinated views and a suite of interactions to support the interactive query execution analysis at a multi-level of details. To demonstrate the query execution overview, we design an optimized algorithm to layout the temporal directed acyclic graph(TDAG), which takes both the temporal information and the topology information into consideration. To explore tasks set and their data dependencies, we implement the machine view integrating the task distribution components and machine performance components for users to analyze the interaction between tasks and distributed system. At last, the task view is designed to show detailed information about the individual tasks. The system can run with two modes: 1) analysis mode: the system directly shows the final results for exploration; 2) simulation mode: the system replays the execution process with a given simulation rate.

The contribution of this works can be summarized as follows:

- The design and implementation of QEVIS, a visual analytic system for understanding and analyzing the distributed query execution process.
- Well-established visualization front-end including the algorithm to layout for visualizing the TDAG, novel visualization design to demonstrate the large number of tasks and their complex dependencies.
- Case studies on the analysis of query process performed on the Hadoop platform.

2 Related work

2.1 Query Analysis

Understanding the query behavior and evaluating database performance has been studied for decades since the database management systems(DBMSs) have been found. Both database and visualization communities have proposed methods to analyze the performance and diagnose the queries automatically or manually. We give a brief introduction about the related works from the two following aspects: *query logic structure* and *query execution structure*, and refer the interested readers to [7] for the systematic overview about the database query debug and performance analysis.

Analyzing query structure. Queries such as SQL sentences can be hard to read since they always have a deep and nested structure. Many research works have been conducted to help the database users to understand the quires quickly. The most common method is to utilize visualization techniques to show the logic structure of operations[all]. For example, extended from previous work, QueryVis uses the node-link diagrams to show the relationship between operators; the unambiguity is also proved in this paper. Other than visualization, Gawade et al. proposed a method to translate a query to Natural Language.

Analyzing query execution. After a query is issued by the database users, the query will be optimized and executed on the database platforms. Especially for the distributed database system, the query will be translated into the logic execution plans, which are used to generating the physical tasks. Understanding the execution plans is important for users to expect the query performance. Many existing industrial softwares are developed to visualize the query execution process [6]. These softwares always utilize the Gantt chart to show the progress and use tree or directed acyclic graph(DAG) to show the relationship among the operators. VQA [23] displays the logic of the query plan as a tree with the nodes indicating operators and the edge indicating the dataflow. Barcharts are inserted into the node to show the metric of the operator(e.g., execution time, memory allocated). Another work highly related to our is Perfopticon [17], which designs a visual analytics system consisting of coordinated views to visualize the query plan, the overall query execution data flow and execution trace. The system allows users to observe the execution trace of fragments or operators on different workers and can help the database users successfully identify the query problems such as data skew, incorrect hashing, etc.

Our work is highly motivated by Perfopticon and performs the finer-grained level of exploration: the task level, which can identify the abnormal query behavior more precisely. Compare with Perforpticon, our system also consider the physical system performance metrics in our visual design to help users to reason the query execution.

2.2 Visualization for Sequence Data

The query execution records in the distributed database system can be represented by even sequence data. As a special type of time-series data, event sequence record a series of discrete events in the time order of occurence [10]. Nowadays, researchers have studied the sequence data visualization techniques as applied to various applications such as health care [15, 25], social media [14, 28], and education [3, 4, 9, 11, 18]. For the detailed taxonomy about the time-series and event sequence visualization, we refer the readers to read the surveys [10, 22].

Sequence visualization is designed to reveal the information of events such as the event type, start time, end time and duration. Moreover, for the complex application requirement, various visualization tasks are proposed, such as visual summarization, prediction & recommendation, anomaly analysis and comparison. Existing visualization techniques can be classified into five categories according to the form of visual representations, i.e., *sankey-based visualization*, *hierarchy-based visualizations*, *chart-based visualizations*, *timeline-based visualizations* and *matrix-based visualizations* [10].

Hierarchy-based visualizations [8], ***sankey-based visualizations*** and ***matrix-based visualizatios*** are always designed for displaying the sequence or sequence collection after modeling the them as special structures such as graph or tree. For example, LifeFlow [26] utilizes the tree structure with a node presenting a group of events to summarize the sequences. Outflow [25] models the progression paths of sequences as directed acyclic graph with a node indicating a cluster of states, and then visualize the graph as Sankey diagram [20]. These methods always provide the highly abstract summarization for sequences and cannot directly reveal the pattern of specific individual sequence. Ma-

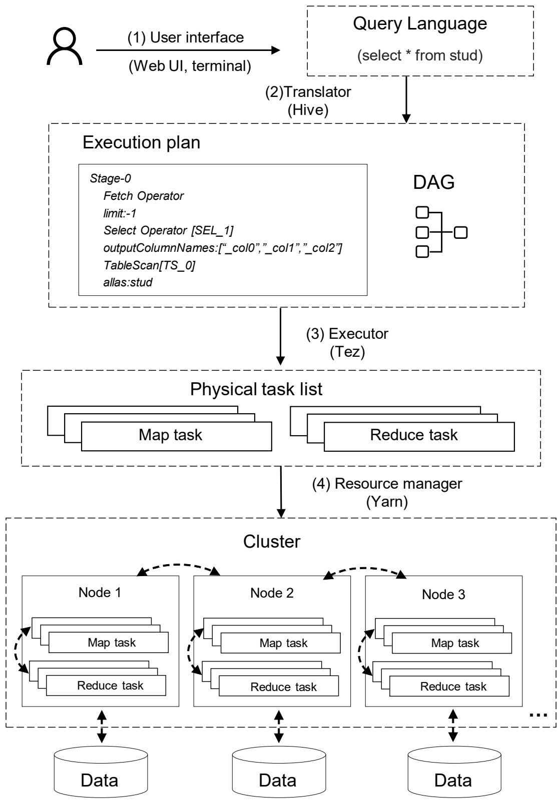

Fig. 2. Overview of distributed query pipeline(Hadoop 2.0).

trixwave [29] utilizes a sequence of matrixs to show the connections between specific events, which can provide details connecting information of sequence. However, Matrixwave loss the detailed temporal information and cannot be used for very large sequence collections.

Chart-based visualizations uses barchart, linechart or scatter plot to visualize the trend or distribution of events, which always works as assisting views to support the interactive explorations. For instance, barchart and linechart are always used to show the distribution of the attributes of sequences or temporal trends [1, 8]. Scatter plot can be used to show the overview of coarse-level overview of the sequence or sequence groups by projecting them to the 2D canvas through dimension reduction algorithms or two specific attributes [8, 16, 27]. Other than the distribution of sequences, the scatter plot can also reveal the outliers.

Timeline-based visualizations are known as the most intuitive ways which demonstrate the events in time order. Gantt chart is a direct way to show the temporal information of event sequences, including the start time, end time and duration. Many industrial tools such as the Tez UI uses the Gantt diagram to clearly demonstrate the progress of the operations. Lifeline [19] use Gantt diagram to display the sequence as well as the events and each sequence takes a single row. LiveGantt [12] proposes an algorithm to visualize the scheduling events with better scalability. However, these methods cannot directly be used in our application since the dependencies of these sequences are ignored. Moreover, the Gantt chart also suffers the series scalability problem when applied it to a large sequence dataset, and the abnormal sequence will be hidden without alignment.

3 BACKGROUND

The architecture and terms are introduced in this section to serve the as basis for the further discussions. The figure 2 demonstrates how a query is processed by a distributed query system. We use Hive based Hadoop architecture in this paper, and the analysis pipeline can be easily extended to other systems.

When user issues a query (shown as Figure 2(1)) through the interface such as a web-based user interface or SQL terminal. Hive uses a cost-based optimizer to optimize the query such as determining the best methods for scan operators, join orders and aggregate operation, and then translate it as the logic execution plan shown as Figure 2(2). The logic exaction plan may contains hundreds of lines of description, which describes the execution process as a Directed Acyclic Graph(DAG) that can be processed by the executor(e.g., Tez).

A DAG is a collection of vertices and edges. Logically, a **vertex** consists of a sequence of logical operators(such as filter, aggregate, etc) which describes the execution of a part of the query. Further more, there are two types of vertices in the Tez DAG, **map** vertex and **reduce** vertex. In the DAG, the **edges** define the data movement between the adjacent vertices. We name of source vertex as the **producer** vertex and the target vertex as the **consumer** vertex. Noticed that a producer vertex may connect to multiple consumer vertices and a consumer vertex could be connected by multiple producer vertices.

With a given vertex, Tez further creates a set of atomic **tasks**(shown as Figure 2(3)). Then these tasks will be dispatched on the physical machines by Yarn, the resource manage tool of Hadoop2.0. A task takes a piece of data as input and execute all operators defined in the corresponding vertex. To ease the analysis of tasks, we define the sub-processes of a task as five steps. For a map task, the steps are *Initialization*, *Input*, *Processor*, *Sink*, *Spill*. For a reduce task, the second step is *Shuffle* instead. Moreover, the data read by a task could come from the local file or producer tasks.

4 SYSTEM DESIGN

4.1 Requirement Analysis

During one year of collaboration, we have closely collaborated with three experts in distributed database, who are also the co-authors of this paper.

In the first month of the collaboration, we have held brainstorming to collect the most frequent raised questions when analyzing the distributed query system performance. Based the discussions with domain experts and review of existing literature, we have formulated the following design requirements.

R1 Understand the general query execution progress and query plan structure. Before our collaboration, the domain experts have used profiling software (Tez UI, etc) or visualization tools(Tableau, etc) to show the query progress as Gantt chart and query plan structure as directed graph. However, these two visualizations are always displayed in separated views which require users to switch their focus thus break 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

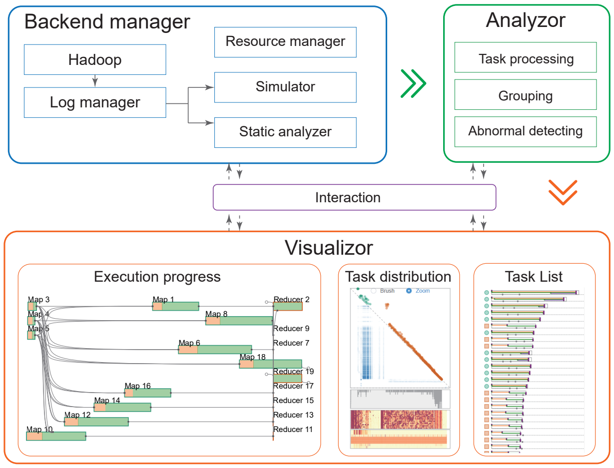

Fig. 3. QEVIS system consists of three major components: backend manager, analyzer and visualizer.

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 Design

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

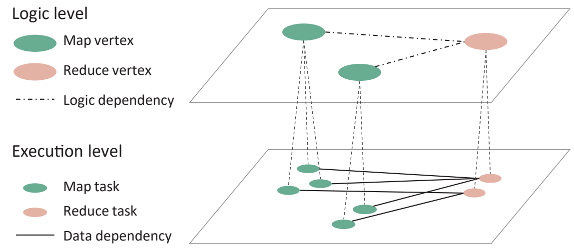

Fig. 4. Two-level temporal graph.

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.

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

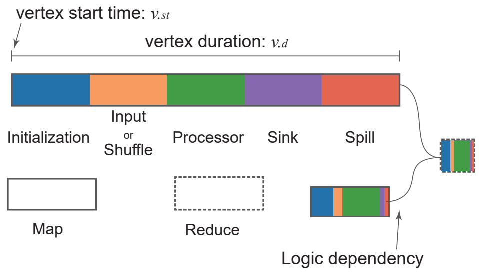

Fig. 5. Visual encoding of logic vertex and dependency

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.

6.1.1 Visual Encoding

The commonly used method to visualize the progress data is Gantt Chart. As shown by Figure 5, the x-axis indicates the timestamp. The rectangles with the same height indicate the temporal information of logic vertices. Given a vertex v, the position of the 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.

6.1.2 Layout Method

In Gantt Diagram has been deployed to many progress visualization tools such as Tez UI and Tabula, each progress bar takes up the whole line with horizontal position and length encode start time and durations which is shown as Figure 6(A). Vertically, the progress bars are ordered by the start time of the vertex. To present the logic dependencies between vertices, we use curves instead of straight lines to avoid the intersection between the lines and progress bars as much as possible. In this visual design, since the layout doesn't consider the logic dependencies, it always results in a very serious cross between the links and unclear topology structure.

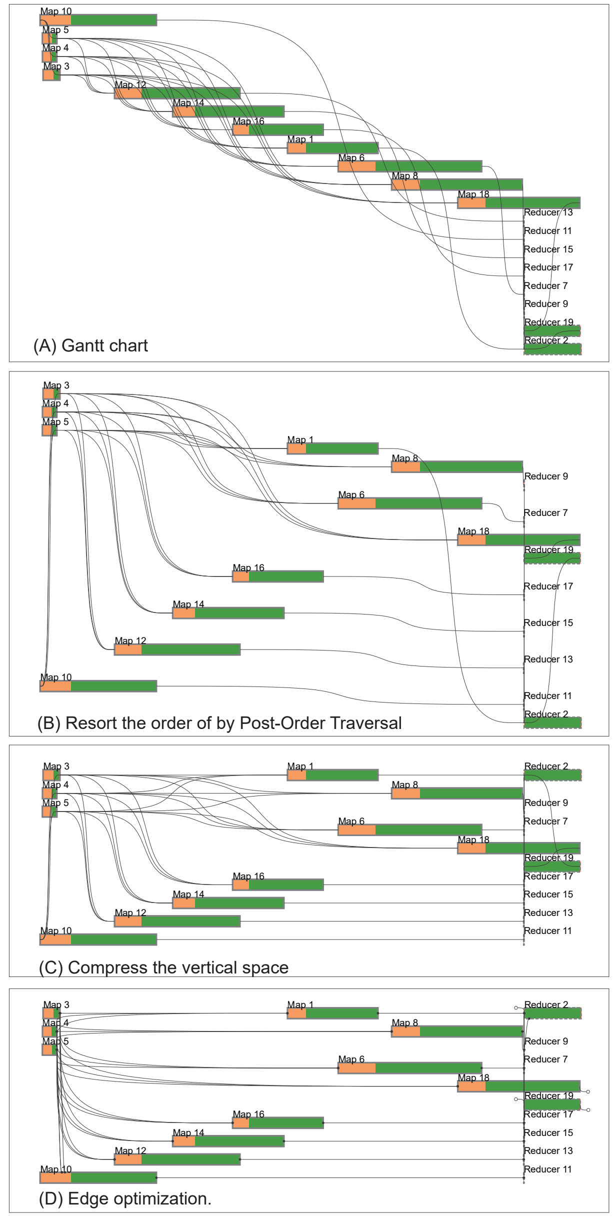

Fig. 6. Temporal DAG layout algorithms

To tackle this problem, we propose a three-step progress to generate the layout and visual form: Graph Reorder **Reorder the vertices:** First, we reorder the

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 Diagram-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 horizontal bars clearly. Moreover, visualizing a large number of bars in this way is difficult for users to compare the absolute length of the

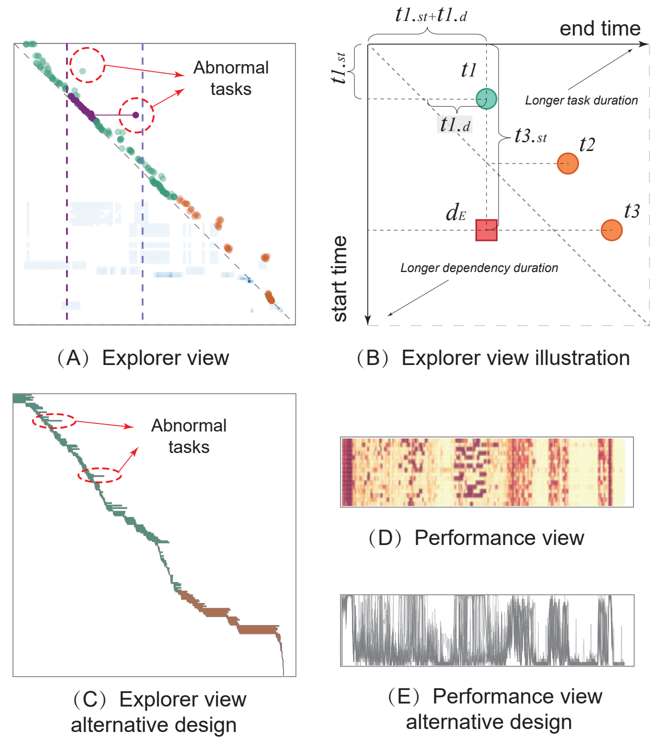

Fig. 7. Visual design for the pattern explorer.

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 Query 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 distribution of dependencies according to their temporal information,

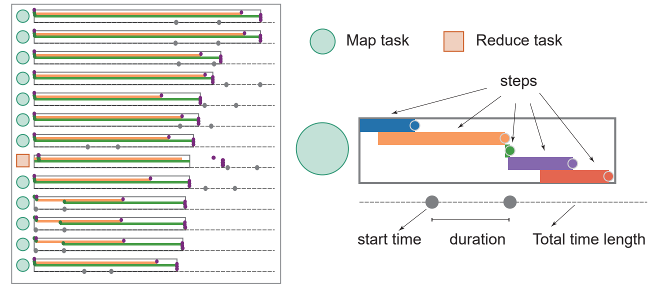

Fig. 8. Visual design for task list.

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 as individual points, which is shown as **??**. 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 [24] 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 [13] to visualize data amount of dependencies. For example, as shown in Figure 7(XX), 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 execute 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] 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.

[2] 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.

[3] 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.

[4] 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.

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

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

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

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

[21] 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.

[22] 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.

[23] 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.

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

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.