Journal of Computers Vol. 33 No. 1, February 2022, pp. 87-100
doi: 10.53106/199115992022023301009

A Progressive Real-time Visualization Method for Earthquake Big Data

Weifeng Shan[1], Jianqiao Li[1], Yuntian Teng[2], Huiling Chen[3], Zhiyang Li[1], Maofa Wang[4*]

[1] School of Emergency Management, Institute of Disaster Prevention, Sanhe 065201, China
shanweifeng@cidp.edu.cn, m.lijianq@gmail.com, lizhiyang1024@gmail.com
[2] Institute of Geophysics, China Earthquake Administration, Beijing 100081, China
tengyt@cea-igp.ac.cn
[3] College of Computer Science and Artificial Intelligence, Wenzhou University, Wenzhou 325035, China
chenhuiling.jlu@gmail.com
[4] Guangxi Key Laboratory of Trusted Software, Guilin University of Electronic Technology, Guilin 541004, China
wangmaofa2008@guet.edu.cn
Received 20 March 2021; Revised 12 August 2021; Accepted 8 September 2021

Abstract. As the volume of seismic observation time-series data grows larger, web-based visualization schemes suffer from longer system response times. Although big data visualization schemes based on sampling and filtering can greatly reduce the data scale and shorten transmission time, what it gains in speed it loses in information. Progressive visualization has become an increasingly popular scheme because it can quickly "see" some results without having to wait for all the data, thus enabling users to grasp a data-change trend quickly and perceive the rules behind it. In this paper, a Cloudberry-based progressive real-time visualization schema for earthquake big data (PVSEBD) is proposed for the first time. It greatly shortens the transmission time of each data slice, improves the user interaction experience, and meets the long-term, large-scale visualization needs of earthquake consultation business. Because the correctness of average aggregation function (AVG) in progressive visualization is often not guaranteed, this paper proposes an innovative AVG translation rule solution based on the accumulability of the COUNT and SUM aggregation functions. The experimental results showed that PVSEBD automatically adjusts the amount of data returned each time according to size and has a shorter response time for each interaction compared with the solution based on the web-based visualization toolkit Portable Progressive Parallel Processing Pipeline (P5).

Keywords: earthquake big data, big data visualization, progressive visualization, aggregate function, Cloudberry

1 Introduction

China's Digital Earthquake Precursor Observation Network now has nearly 3000 sets of instruments, covering observation methods such as fluid, geomagnetism, geoelectricity, deformation and gravity. Most of these are second-sampling devices that acquire time series float data at a sampling rate that on some devices can reach 100 Hz. One of these devices generates 65.9 MB of data per day and 23.5 GB per year. With the implementation of the National Earthquake Intensity Rapid Reporting and Early Warning Project, the National Intensive Earthquake Observation Project and others, there will be several times the amount of today's seismic observation data to be processed and represented with appropriate visual solutions to make them useful [1]. Visual analysis of the data helps with earthquake preparation because displaying useful information is crucial for responding to changing states. To enhance feedback to users [2], it is necessary to develop a seismic big data visualization system that allows seismic experts to view changing trends in long-term observation data as they conduct earthquake consultation.

Visualization is the process of representing data, information and knowledge in the graphical form [3], and it is also a basic tool for observing and analyzing data [4]. With the increasing digitization and interconnection of the world and the exponential growth of data, more and more physical objects are integrated into the information network. Data visualization can promote people's understanding of massive data [5]. Data visualization is the process of processing data of any size or dimension to produce a set of lower dimensional or understandable data, so as to make it easier for people to operate and understand the data [6]. Data visualization also enables researchers to understand their own data and convey their opinions to others [7]. In a word, it provides experts, among others, with an intuitive and interactive means to explore and analyze data to identify patterns and infer correlations and causality [8]. Without visualization, it is difficult for users to perceive the value in big data quickly; however, this

* Corresponding Author

method has its own set of requirements. More than just presenting data, visualization participates in data analysis through human-computer interaction. As a multi-disciplinary research field, human-computer interaction is the basis of designing human-computer interaction system [9]. This interactive performance is the core of emerging visualization analysis methods because it is where the “data - information - knowledge” flow is realized.

Different user requirements and application scenarios produce different visual presentation methods of big data processing [2]. There are many general applications for visualization software (Tableau, power Bi, Excel and MATLAB), web-based visualization systems (B/S structure, through front-end development technologies such as HTML, CSS and JavaScript) and data visualization component libraries (including D3.js, HightCharts, ECharts, Antv), but for seismic visualization these applications need a lot of redevelopment since there is no comprehensive analysis or summary for seismic big data. To address this problem, web-based big data visualization schemes are currently popular, and Web3D [10, 11] is a lightweight solution that processes large-scale 3D model data to improve real-time visualization for on-line transmission and browser display. Another web-based system, Cartolabe [12] explores large topic-based text corpora. However, neither of these is suitable for time-series data scenarios given the increase in data volume and algorithm complexity and the longer wait times. This is a particular problem for seismic experts who usually need to view analyze and process long-term time series data from multiple observation instruments at the same time. Let us suppose that an earthquake expert wants to view original seismic observation data from three observation instruments from 2010 to 2019. Assuming that the observation instruments were sampled in minutes, the data generated by each device would be about 10 bytes per minute, and the data to be transmitted would be about 150 MB. If all the data were transmitted at one time over the web and the client received and visualized the data, the wait time would be very long. If the query results were transmitted one year at a time, they would be visualized and rendered 10 times, and each transmission would be about 15 MB, a greatly reduced scale of data transmission, and users would quickly see part of the visualization results.

Usually, sampling and filtering are used to reduce the scale of data transmission and improve visualization speed. Two common methods are random and hierarchical sampling [13], which are usually applied in approximate query processing [14]. For example, the ScalaR [15] system used a simple random sampling technique, and Chaudhuri et al. used an optimized, layered Microsoft SQL Server database systems sampling technique [16]. For filtering technology, Gao et al. [17] proposed a distributed DMF filtering algorithm for large network systems; Axelsson [18] presented an adaptive triangulated irregular network (TIN) filtering algorithm; and Hui et al. [19] combined multi-level interpolation and traditional morphological algorithms to propose an improved filtering algorithm. Although the sampling and filtering schemes greatly reduced the scale of data transmission and shortened the time of data transmission, it loses information behind the high-frequency data. Since earthquakes are the result of long-term crustal movements, seismologists usually need original long-time seismic observation data to find seismic variation rules. Therefore, the visualization method based on sampling and filtering is not suitable. In view of the shortcomings of the two sampling and filtering visualization methods as well as the characteristics of seismic data, this paper proposes progressive visualization, a scheme that has become more and more popular in the visualization community. Compared with traditional schemes, progressive visualization helps users interact better with a large amount of data, and gives priority to speed to enhance interactive analysis. Progressive visualization uses online aggregation to decompose long-time calculations into smaller and faster blocks, and return some approximate results without having to wait for all results [20]. Although it has many advantages, our research has found potential risks. For example, in most of the current progressive visualization schemes, the calculation of aggregation function needs to merge data fragments. Because existing computer systems can easily produce wrong intermediate results, users will draw wrong visualization conclusions. In this paper, we make a series of improvements to produce a correct calculation of the aggregate function.

The main contributions of this paper are as follows:

First, this paper puts forward a web-based, progressive, real-time visualization scheme for earthquake big data based on a Cloudberry middleware system for the first time.

Second, this paper addresses the common problem of incorrect AVG aggregation function calculation results, a problem that appear in the Cloudberry progressive scheme, which is based on the DRUM mechanism. Therefore, this paper proposes an AVG translation rule solution based on the cumulative principle of COUNT and SUM aggregate function.

Finally, this paper draws a comparison between PVSEBD and the web-based visualization toolkit P5 and analyzes their respective advantages and disadvantages. The experimental results showed that PVSEBD automatically adjusted the size of each data return according to its size and displayed the results quickly. The response time of each interaction was significantly less than that of the other two systems.

2 Related Research

The common schemes of big data visualization are based on sampling and filtering strategies and progressive visualization schemes.

2.1 Visualization Scheme based on Sampling Strategy

Sampling [21, 22] technology is usually used in statistical analysis to extract a small number of samples from a large amount of data for analysis. Literature [23] considered the common special case of point datasets and proposed an efficient random sampling algorithm. In random sampling, each data point has the same probability of being selected, so the resulting sample may not be representative and may miss important data points [24]. To make improvements, Literature [25] used uniform sampling and non-uniform sampling; moreover, it improved sampling algorithms by analyzing the way users perceive density differences [26, 27]. Literature [28] presented a sampling-based visualization solution for search and large-scale network applications depending on which a Web-based ALVIN system was implemented. These sampling-based visualization schemes all have a common problem: the data sampled each time is random, and the hidden information behind the data may be lost due to its unrepresentativeness. Consequently, some prior knowledge and complex pre-processing operations are usually required [29].

2.2 Visualization Scheme based on Filtering Strategy

In this strategy, qualified data are filtered from the data set according to the filtering conditions. Literature [30] offered a visual information seeking (VIS) solution based on rapid filtering, which displayed the search results gradually. To solve the problem of data continuity aimed at different types of spatio-temporal characteristics, Literature [31] proposed a filtering algorithm. Literature [32] presented another filtering method which only needs a few parameters, which may appeal to users who do not have much experience using the related technology. However, these filtering methods usually only return a specific subset of data that does not convey enough information. Often, these schemes require many complex pre-calculation algorithms, which makes them unsuitable for earthquake prediction.

2.3 Progressive Visualization Scheme

In progressive visualization, part of the calculation result is gradually generated and visualized during the execution of the visualization. These partial results can be continuously merged into the visualization interface until the complete result is formed over time. Progressive visualization allows users to see the intermediate results running for a long time without waiting for all the results to be calculated [33]. Cook and Thomas [34] defined visual analytics as the science of interactive visual interfaces and extended the concept of progressive data analysis to visual analytics. Glueck et al. [35] introduced a Splash framework that gradually transfers data between the server and the client, thereby speeding up the system's response speed, improving user interaction, and showing the user benefits of gradually loading data on a slower network. Rinzivillo [36] proposed a method of progressive clustering, which obtains a relatively small subset by calculating the distance function to generate the calculation results quickly. Fekete [37] developed a Python toolkit ProgressVis to implement progressive visual analysis. Wong et al. [38] presented a powerful visual data-mining [39] system that supports progressive visualization based on SPAM-like algorithms. Stolper et al. [40] built a prototype system called Progressive Insights, and improved the SPAM algorithm [38, 41] to improve the progressiveness and interactivity. With the popularization of multi-core CPU and GPU hardware, Li et al. [42] proposed a Web-based visualization toolkit Portable Progressive Parallel Processing Pipeline (P5), which can make full use of GPU parallel computing resources and perform progressive data analysis and Visualization. Luo et al. [43] proposed VisClean, which can progressively visualize data with improved quality through interactive. Jia et al. [44] proposed a set of progressive, interactive, real-time query middleware, for large-scale data, Cloudberry. It uses a mechanism called DRUM, which divides a query request on large-scale data into N irregular micro-queries according to a specific database field. It focuses on how to deliver the results of mini-queries smoothly by following an "expected rhythm" so that the user sees regular updates of the incremental results. The above visualization schemes mainly focus on which progressive strategy is adopted to realize interactive visualization. They rarely consider the influence of the progressive schemes on the correctness of the AVG aggregation function results.

3 Cloudberry Progressive Visualization System

3.1 Cloudberry System Architecture

Cloudberry is an efficient, flexible, interactive real-time visual analysis middleware system [37]. Fig. 1 shows its three-tiered architecture [45]. It consists of two modules: Neo and Zion [46]. The Neo module is a web server based on the Play Framework and uses the Actor model to receive JSON visual query requests sent from the front-end. The Controllers component defines the server component interfaces corresponding to client request routing, such as register, logout, and query. The views component defines a set of web query and result display interfaces based on HTML and JavaScript; the database (DB) component is mainly responsible for connecting to the back-end database according to Cloudberry's configuration information and checking and creating a metadata table. The Zion module is the core component of Cloudberry middleware and also uses the Actor model. It consists of the following components:

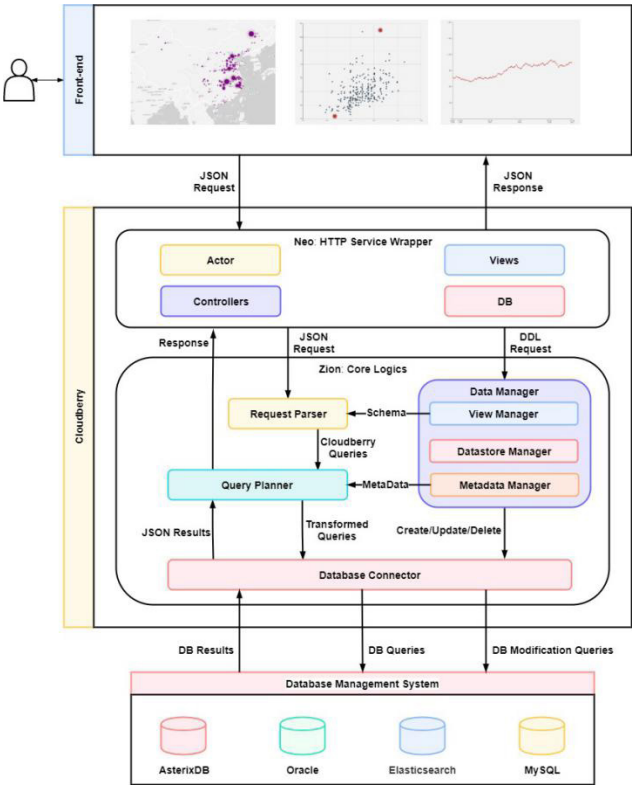

Fig. 1. Cloudberry three-tiered system architecture

(1) Request Parser——used to check incoming JSON format query request syntax, parse it into an internal query language, and forward the parsed result to the Query Planner.

(2) Query Planner——responsible for query rewriting depending on the given views information. If there is an appropriate view, the original query will be split into multiple queries to ask different datasets. After all the results come back, the query planner will merge results from all queries and return to the client. Not every query can find an appropriate view, especially at the beginning when the system is starting. In this case, instead of waiting for the entire query result, Cloudberry returns a series of partial results by streaming at a steady pace. It splits the query into a series of mini-queries, and the selectivity of each mini-query is adapted based on the query performance so that each one is guaranteed to finish within a short time.

(3) Data Manager——responsible for processing data-related modules: metadata, datastore and view managers. The metadata manager and datastore manager are mainly used to map data formats and register data sets. Cloudberry uses view technology to cache data and improve query response speed [47]. The view manager controls the life-cycle of the views and stores the metadata of the base datasets and their associated views [48].

The materialized views and the meta-dataset of Cloudberry are also stored in the backend database. In this way, we can rely on the database to do the computations both on views and base datasets, so the middleware is light-weight.

(4) Database Adapter——Cloudberry's adaption interface for different database management systems, such as MySQL, Oracle relational databases, and AsterixDB semi-structured databases. The latest version also provides an Elasticsearch interface to translate Cloudberry internal queries into SQL statements that conform to the target database syntax, encapsulate the query return results in JSON format and send them to the query planner.

3.2 Progressive Visualization Based on Drum Framework

Cloudberry implements progressive visualization based on the DRUM framework. The framework focuses on dividing an original query into N micro-queries according to the time dimension and ensures that these micro-queries are returned to the front-end in as quickly and incrementally as possible to improve the user interactive query experience [49].

The specific approach of the DRUM framework is to divide the original query Q into a series of non-equidistant mini-queries Q_1, Q_2, Q_3, ..., Q_n in the time dimension, and according to the response performance of the previous micro-queries to predict and adjust the time range of each micro-query to ensure that each micro-query can be completed in a short time and return the query results as uniformly as possible. Micro-query greatly reduces the scale of each data processing and speeds up the response speed.

Fig. 2 shows the execution process of the DRUM framework [46]. For each query request from the front-end, it will be submitted to the micro-query generator. The generator uses the estimation [50, 51] information provided by the evaluation module to select the query range of the next micro-query Q_i, and then generates the query Q_i and sends it to the database [49]. After receiving the micro-query results, Cloudberry aggregates them along with previous fragments into new query results that replace the original ones. Finally, the middleware server sends these aggregated results to the front-end, the web browser receives these results and it merges them into an interactive visualization interface. At the same time, the evaluation module collects the running statistics of this micro-query and evaluates the time range of the next one.

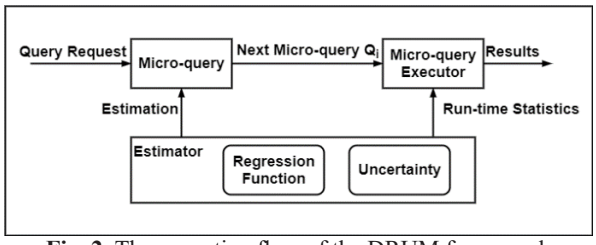

Fig. 2. The execution flow of the DRUM framework

Compared with existing progressive visualization schemes such as P5, the progressive visualization scheme based on DRUM has obvious advantages. DRUM dynamically adjusts the query conditions based on evaluating each one's response time, so although the number of data records returned each time may be different the response time changes little and remains within the user's acceptable range. However, P5 and other solutions require users to set the batch size manually. With the increase of data volume, the system response time will also improve significantly.

4 Accuracy Analysis of AVG Aggregate Function Results in Progressive Visualization

Aggregation is one of the most commonly used methods for querying and analyzing time-series data, which record the most original state change information, whereas query and analysis are usually processed based on original values. Common aggregate functions include SUM, MAX, AVG and COUNT. In progressive visualization, data are usually divided according to unequal intervals. The correctness of the results of an AVG aggregate function calculation is a common problem that may not have been considered in existing schemes. Therefore, it is important to ensure the correctness of aggregation function in progressive visualization. We used a time-series dataset to illustrate this defect in a formal way. For example, in an earthquake consultation business, experts often need to aggregate observation data by day, month and year. For example, they may choose to view the monthly

average value of earthquake observation data from 1 January 2019 to 31 July 2020 as in Fig. 3.

```
Q: SELECT to_char(t.OBSDATE, 'yyyy-mm') MONTH, avg(OBSVALUE) as AVG
    FROM EARTHQUAKEDATA t
    WHERE t.OBSDATE >= to_date('2019-01-01', 'yyyy-mm-dd')
      AND t.OBSDATE <= to_date('2020-07-31', 'yyyy-mm-dd')
    GROUP BY to_char(t.OBSDATE, 'yyyy-mm')
    ORDER BY MONTH;
```

Fig. 3. Query example

The STARTDATE field here is formatted as "yyyy–mm–dd", and then grouped by month. The average value of the observation data is calculated and finally sorted by month.

In a relational database, it is no problem to aggregate the monthly mean, and it is easy to implement. However, since Cloudberry usually divides a query into multiple micro-queries with different intervals, the correctness of the final result of AVG aggregation function cannot be guaranteed if there is only the resultant information after AVG aggregation when merging the query results. Unlike COUNT and SUM, AVG is not cumulative. In the above example, when calculating the monthly average value of the earthquake observation data from 1 January 2019 to 31 July 2020, Cloudberry can divide the total query Q into N min-queries Q_1, Q_2, Q_3, ..., Q_n according to time conditions. As shown in Fig. 4, we used micro-query Q_i ($1 \le i \le n$) to represent the ith query, AVG'(Q_i) to represent the average value of the ith query in Cloudberry, AVG_{comb} (Q_i) to represents the average value of the ith query after merging and AVG(Q_i) to indicate the correct results. In addition, SUM(Q_i) represents the sum of the data of the ith query, and COUNT(Q_i) represents the total number of time series points of the ith query. In Cloudberry progressive visualization scheme, the data are segmented at unequal intervals, and the size of each Q_i interval is different. This way, the number of returned results, COUNT(Q_i), is not fixed.

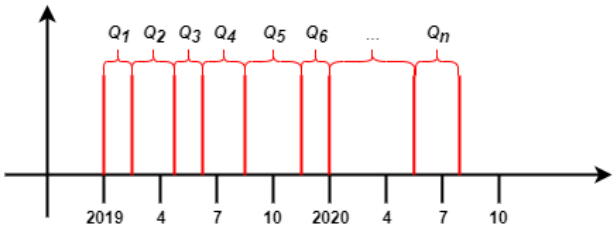

Fig. 4. Query slices

After executing micro-query Q_1, the average observational data returned by Cloudberry to the front-end should be the average of the query results: AVG'(Q_1). After executing sub-query Q_2, the average value of the second query is AVG'(Q_2); after each data transmission, the calculation results should be combined. According to Cloudberry's progressive strategy, the results of Q_1, Q_2 need to be combined at this time. Before modifying Cloudberry's internal translation rules, Cloudberry calculates the AVG aggregation function according to the following formula

$$AVG_{comb}(Q_2) = \frac{AVG'(Q_1) + AVG'(Q_2)}{2}. \quad \textbf{(1)}$$

AVG can be expressed by SUM and COUNT, so equation (1) also can be written as

$$\frac{\frac{SUM(Q_1)}{COUNT(Q_1)} + \frac{SUM(Q_2)}{COUNT(Q_2)}}{2}. \quad \textbf{(2)}$$

However, in general, the result of Q_2 should be

$$AVG(Q_2) = \frac{SUM(Q_1) + SUM(Q_2)}{COUNT(Q_1) + COUNT(Q_2)}. \quad \textbf{(3)}$$

When COUNT(Q_1) = COUNT(Q_2) = m (m>0), equation (2) can be written as

$$AVG_{comb}(Q_2) = \frac{SUM(Q_1) + SUM(Q_2)}{2m}. \quad \textbf{(4)}$$

At this time, equation (3) and equation (4) are equivalent. However, because data are segmented at unequal intervals, so COUNT (Q_i) (1≤i≤n) may not be equal. Therefore, it is necessary to ensure the accuracy of the AVG aggregate function after segmentation and merging under the non-equidistant progressive segmentation strategy. Furthermore, in addition to Cloudberry, other progressive visualization schemes may also have incorrect calculations of the AVG aggregation function.

The correct result should be the sum of all the data in the current month divided by the total number of days according to the following formula

$$AVG(Q_i) = \frac{SUM(Q_1) + SUM(Q_2) + ... + SUM(Q_i)}{COUNT(Q_1) + COUNT(Q_2) + ... + COUNT(Q_i)}. \quad \textbf{(5)}$$

4.1 Solution of AVG Aggregate Function in Cloudberry

Since Cloudberry originally designed the hypothetical application scenario to provide efficient interactive real-time analysis services for large-scale data, it mainly focused on operations such as data statistics and accumulation. However, it has good scalability, which supports its improvement and expansion. According to the previous analysis, Cloudberry translates COUNT, SUM, AVG and other aggregation operations sent from the front-end into the corresponding aggregation functions of the target database. To ensure the correctness of the AVG aggregate functions, we proposed a change to the Cloudberry translation rules. The specific implementation process is shown in Fig. 5.

(1) After Request Parser receives the query request sent from the front-end, if the query contains AVG(*f*) aggregation function, it will be translated into two functions with special internal naming COUNT(*f*) and SUM(*f*), and then sent to Query Planner.

(2) When the progressive client interface receives the results of Query Planner's execution, according to the naming correspondence between COUNT(*f*) and SUM(*f*), the results of current and last SUM and COUNT are merged into the result of AVG aggregation function with equation (5).

(3) Finally, the progressive client interface sends this result to the front-end.

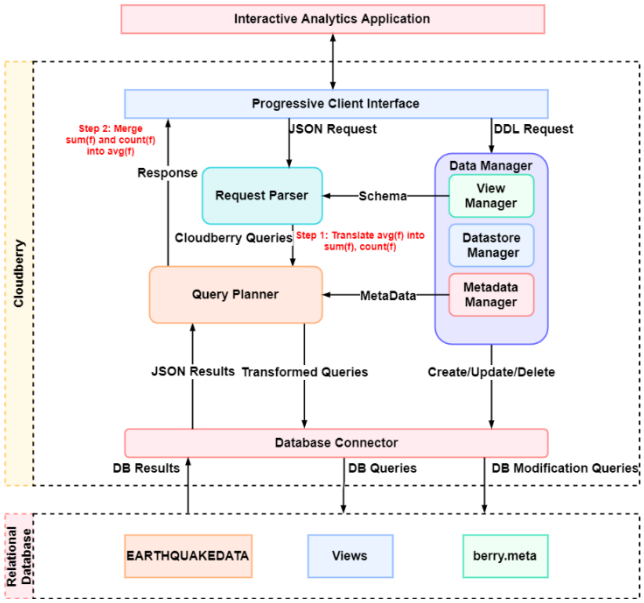

Fig. 5. Solution of AVG in Cloudberry

It can be seen that, except for the Progressive Client Interface and Request Parser components, the other components of Cloudberry are not aware of the existence of the AVG aggregation functions, and the overall improvement plan does not affect Cloudberry's overall structure and logic.

Solving the problem of the correctness of aggregate function is significant because existing progressive visualization schemes such as P5 ignore the calculation rules of the aggregate function under non-equal interval segmentation, and often cannot guarantee the correctness of the aggregate function. Cloudberry, though, can be improved because it is scalable, and by changing Cloudberry's translation rules, the correctness of the aggregation function is guaranteed.

5 Progressive Visualization of Earthquake Big Data

The seismic observation data have the characteristics of a large data scale and long-time span. Cloudberry builds a progressive visualization of seismic big data based on the web, which can quickly produce partial calculation results and produce complete and accurate results over time, thus avoiding big data under the environment of traditional "waiting for the calculation to be completed" visualization [31]. In this paper, we proposed a Progressive Visualization Schema for Earthquake Big Data based on Coludberry, which is named PVSEBD.

To use PVSEBD, developers must register the dataset that the front-end needs so that it can access PVSEBD in advance to inform it of the dataset's basic structure. In addition, it is necessary to encapsulate the query request into the JSON format to achieve progressive visualization.

5.1 Register Earthquake Data Set

Since the existing seismic observation data was stored in a relational database, this article is based on a relational database simulation to create a new data table named EARTHQUAKEDATA, which includes DEVID, OBSDATE and OBSVALUE fields that represent seismic observation instruments, observation time and observation values. To realize the progressive visualization of seismic observation data, the data table information needs to be registered with PVSEBD. The format is shown in Fig. 6.

The above content is a JSON file, as defined by the Data Definition Language (DDL) specification, which is sent to the Metadata Manager. PVSEBD adds a record of this dataset to the metadata table. The JSON file mainly includes the following parts [52]:

- "dataset", the name of dataset (table) in the database, and
- "schema", the data structure information of dataset, which comprises the following five items:
- "typename", dataset type name;
- "dimension", field name;

- “measurement”, apply SUM, COUNT, AVG, MAX and other aggregate function columns;
- “primaryKey”; and
- “timeField”, time column, used for time slicing.

```
{
  “dataset”:”EARTHQUAKEDATA”,
  “schema”: {
    “typeName”:”EARTHQUAKEDATA”,
      “dimension”: [
        {“name”:”DEVID”,”isOptional”: false,”datatype”:”String”},
        {“name”:”OBSDATE”,”isOptional”: false,”datatype”:”Time”}
      ],
      “measurement”: [
        {“name”:”OBSVALUE”,”isOptional”: false,”datatype”:”Number”}
      ],
      “primaryKey”: [“DEVID”, “OBSDATE”],
      “timeField”:”OBSDATE”
  }
}
```

Fig. 6. Registration dataset

Each row in dimension and measurement is a database field description, including name, and isOptional and datatype attributes. The name represents the field name; isOptional represents the column that can be empty in a traditional relational database or may be missed in semi-structured database; and datatype indicates the supported data type of the declared field: Boolean, Number, Point, Time, String, Text, Bag, Hierarchy.

5.2 Web Visualization Interface

According to the needs of seismic data visualization, the web front-end interface contains sampling interval drop-down boxes (Second, Minute, Hour, Day, Month, Year), aggregate function drop-down boxes (AVG, MAX, MIN, COUNT, SUM), and device check boxes, Limit text box, Start Date and End Date parameters, as shown in Fig. 7.

When the user clicks the Search button, the above content will be packaged as a query request in JSON format, as shown in Fig. 8, where

Fig. 7. Progressive visualization of earthquake big data

“dataset” indicates the dataset to be queried (EARTHQUAKEDATA, registered in the previous step).

“filter” is a query filter that is equivalent to the “where” condition in SQL. There are two conditions: one is the device ID selected by the user and assigned to the DEVID field as an array. The relationship is set to “matches”, similar to the “in operation” in SQL. The other condition is the OBSDATE field, the value of which is the start and end times selected by the user on the interface. The operation is in Range, similar to the “between ... and” operation in SQL.

“group” means grouping. The field name of the group after “by” is the same as in SQL language. The OBSDATE field is grouped by day and renamed as dd. Aggregate indicates what aggregation operation is performed on the grouped data. This case is set to average (AVG) the OBSVALUE field, and the column is renamed “v”.

```
{
  “dataset”: “EARTHQUAKEDATA”,
  “filter”: [
    {
      “field”: “DEVID”,
      “relation”: “matches”,
      “values”: [“1”, 2”, “3”]
    },
    {
      “field”: “OBSDATE”,
      “relation”: “inRange”,
      “values”: [“2000-01-01T00:00:00.0Z”, “2190-02-17T00:00:00.0Z”]
    }
  ],
  “group”: {
    “by”: [
      {
        “field”: “OBSDATE”,
        “as”: “dd”,
        “apply”: {
          “name”: “interval”,
          “args”: {
            “unit”: “day”
          }
        }
      }
    ],
    “aggregate”: [{
      “field”: “OBSVALUE”,
      “apply”: {
        “name”: “avg”
      },
      “as”: “v”
    }]
  },
  “select”: {
    “order”: [“dd”],
    “limit”: 3000,
    “offset”: 0
  },
  “option”: {
  “sliceMillis”: 500
  }
}
```

Fig. 8. Cloudberry query request

“order”, under “select”, is used to indicate the sort field of the returned data. At this time, it is set as the grouping field “dd”, and the limit is set at the maximum number of records returned at a time. Here is the value in the limit text box on the interface.

“option” is used to indicate progressive visualization parameters. It has an “option” property, indicating that it accepts the slicing of results and that the expected updating rate is 500 milliseconds.

6 Experiments

To evaluate the performance of the PVSEBD, we designed a number of experiments. All the tests in this article were performed in Google Chrome 83.0.4103.106 (64-bit) on a computer equipped with a six-core 2.20GHz Intel Core i7-8750H CPU and a Nvidia GeForce GTX 1060 (6GB) graphics card.

Since the earthquake data are time series, 100 million earthquake simulation observation records are generated, and each record contains the instrument number DEVID, observation date DT, and observation value V. To test the practical application of different visualization schemes in seismic business, and to evaluate the relevant indicators and ensure the fairness of the experiment, we stored the earthquake data in a MySQL database. The client sent a JSON request, and the server returned JSON format data. To compare and test the system response time, a non-progressive solution based on Flask was developed using Python language. All data in the database were read at one time and then visualized in the Chrome browser by ECharts. In addition, we also compared the PVSEBD performance with that of the relatively new progressive visualization scheme P5 [28], which supports time series data. It provides a declarative visualization generation method that uses GPU parallel processing to improve computing speed, and through progressive visualization, users can quickly perceive intermediate results. In this experiment, the batch size is set at 10 million.

Firstly, the system response time for visualizing 100 million pieces of earthquake observation data with several different visualization schemes was compared, as shown in Fig. 9. Among them, the non-progressive solution took the longest time, more than 500 seconds, whereas the P5 response time was the shortest (360 seconds), mainly because it made full use of GPU computing resources for visualization acceleration. The figure shows that

the system response time grows longer as the visualization processing data scale becomes larger. In a traditional non-progressive scheme, the user has to wait for the system to return all the data, which gives the impression that the system is in a state of suspended animation. Although the total system response time of the PVSEBD visualization scheme was similar to that of non-progressive scheme, users quickly perceived the intermediate results and could stop visualization at any time due to its progressive visualization method.

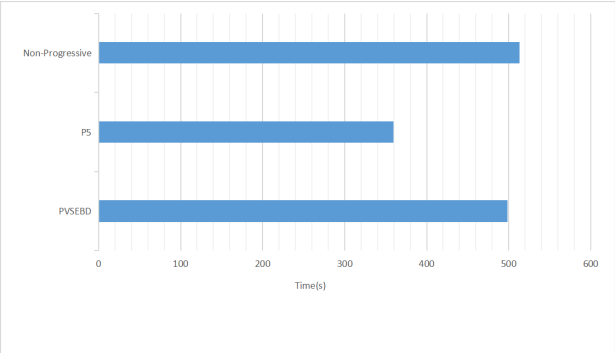

Fig. 9. Visualization system response time (s)

Both P5 and PVSEBD adopt a progressive approach, but P5 requires users to set the batch size manually and keep it unchanged during system operation. To make the response time of each batch more stable, PVSEBD's DRUM mechanism dynamically adjusted the query conditions based on evaluating each query's response time, which meant that that the number of data records returned each time may have been be different, but the response time fluctuated little. To compare the effect of different batch sizes on system response time, the size of P5 and the maximum number of returned data records of PVSEBD were set to 3000, 6000, 10,000, and 30,000. The batch size in PVSEBD is adaptive and can be dynamically adjusted according to data size. Each batch was run 10 times, and the average value was taken as the average response time. The result is shown in Fig. 10.

The batch size seriously affected the response time of the P5 system. If it was set too high, the response time was very long. Although the PVSEBD response time also increased with the increase of data size, it remained within an acceptable user range by dynamically adjusting the batch size. According to the results, the first response time of PVSEBD was nearly four times higher than that of P5, and with the increase in batch size, the response time improved more and more significantly.

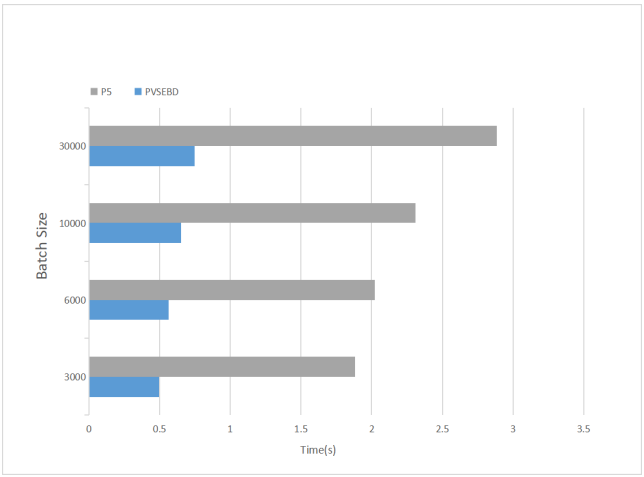

Fig. 10. Average time (s) for PVSEBD and P5 with different batch sizes

7 Discussions

In the era of big data, storage and computing resources are often concentrated on servers. If the server only transmits the results of complex calculations based on a large amount of data to the front end, the data needed to transmitted to the front end can be greatly reduced, the response time can be shortened, and the user experience can

be improved. However, P5 must wait for all the data to be transferred from the server before they can use GPU to accelerate visual computing, which is particularly time-consuming when the amount of data is huge. PVSEBD executes the computation tasks on server sides, only the computation results are transmitted progressively over the network, users can see the visualization result quickly. Therefore, compared with P5, PVSEBD is more suitable for big data environment.

In addition, in order to improve the user's interaction experience, PVSEBD can dynamically adjust the amount of data transmitted each query according to the system response time, which can adapt to different networks and data sizes, especially in big data scenarios. However, P5 requires users to set the batch size manually.

Finally, the AVG conversion solution proposed in this paper based on Cloudberry is applicable to all the unequal interval progressive visualization schemes. A similar solution can be taken for other aggregate functions such as the Min and Max in the progressive visualization schemes.

8 Conclusions and Future Research

This paper introduces the implementation principle of cloudberry big data visualization, and puts forward the progressive visualization scheme of seismic big data PVSEBD for the first time. By using the fragmented loading strategy, the scheme reduced the amount of data transferred each time, shortened the time of each data transmission and improved the user interaction experience.

However, PVSEBD's non-equal interval progressive division method may cause incorrect aggregation function results, this article proposed a translation rules by taking advantage of the accumulability of COUNT and SUM aggregation functions. This rule translated AVG aggregation functions into COUNT and SUM expressions, and then merged the results before sending them to the client. This not only ensured the correctness of the aggregate function results but also did not require changing the Cloudberry architecture.

Finally, the experimental results show that compared with the non-incremental scheme, the PVSEBD can see the results immediately without a long wait. Compared with the P5 progressive visualization scheme, PVSEBD's response time for each batch was shorter, and with the increase of data size PVSEBD maintained a response time within an acceptable user range. Therefore, PVSEBD shortens the user's response time, avoids long waiting time, and improves the user interaction experience.

The next step is to introduce GPU computing resources into PVSEBD to improve system performance.

Acknowledgement

We would like to thank Professor Chen Li from the University of California, Irvine, for his constructive suggestions for our progressive real-time visualization solution for earthquake big data. This work was supported by the National Key R&D Program of China (Grant no. 2018YFC1503806) and the National Natural Science Foundation of China (42164002).

References

[1] T.A. de Almeida, R.F. de Franco, R. Bonacin, Designing Data Visualization Dashboards to Support the Prediction of Congenital Anomalies, International Conference on Human-Computer Interaction (2021) 143-162.
[2] X. Chen, X. Chen, Data visualization in smart grid and low-carbon energy systems: A review, International Transactions on Electrical Energy Systems (2021) e12889.
[3] M.O. Ward, G. Grinstein, D. Keim, Interactive data visualization: foundations, techniques, and applications, CRC press, 2010.
[4] M. Tennekes, M. Chen, Design Space of Origin-Destination Data Visualization, Computer Graphics Forum 40(3)(2021) 323-334.
[5] C. Ceccarini, S. Mirri, P. Salomoni Ceccarini, On exploiting Data Visualization and IoT for Increasing Sustainability and Safety in a Smart Campus, Mobile Networks and Applications 2021(1-10).
[6] A. Kiefer, M. Rahman, An Analytical Survey on Recent Trends in High Dimensional Data Visualization, arXiv preprint arXiv, 2021.
[7] M.L. Waskom, Seaborn: statistical data visualization, Journal of Open Source Software 6(60)(2021) 3021.
[8] N. Bikakis, Big data visualization tools, arXiv preprint arXiv, 2018.
[9] C. Ceccarini, HCI methodologies and Data Visualization to foster user awareness, 2021.
[10]X. Liu, N. Xie, K. Tang, Lightweighting for Web3D visualization of large-scale BIM scenes in real-time, graphical models 88(2016) 40-56.

[11]W. Zhou, K. Tang, J. Jia, S-LPM: segmentation augmented light-weighting and progressive meshing for the interactive visualization of large man-made Web3D models, World Wide Web, 21(5)(2018) 1425-1448.
[12]C. Philippe, R. Jonas, F. Jean-Daniel, Cartolabe: A web-based scalable visualization of large document collections, arXiv preprint arXiv, 2020.
[13]Y. Matias, J.S. Vitter, M. Wang, Dynamic maintenance of wavelet-based histograms, VLDB, 2(2.1)(2000) 2-4.
[14]Y. Park, M. Cafarella, B. Mozafari, Visualization-aware sampling for very large databases, 2016 IEEE 32nd International Conference on Data Engineering (ICDE), 2016.
[15]L. Battle, M. Stonebraker, R. Chang, Dynamic reduction of query result sets for interactive visualizaton, 2013 IEEE International Conference on Big Data, 2013.
[16]S. Chaudhuri, G. Das, V. Narasayya, Optimized stratified sampling for approximate query processing, ACM Transactions on Database Systems (TODS) 32(2)(2007) 9-es.
[17]J. Gao, H. Tembine, Distributed mean-field-type filters for big data assimilation, 2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS), 2016.
[18]P. Axelsson, DEM generation from laser scanner data using adaptive TIN models, International archives of photogrammetry and remote sensing 33(4)(2000) 110-117.
[19]Z. Hui, Y. Hu, Y.Z. Yevenyo, An improved morphological algorithm for filtering airborne LiDAR point cloud based on multi-level kriging interpolation, Remote Sensing 8(1)(2016) 35.
[20]M.A. Procopio, Progressive Visualization for Big Data Exploratory Analysis, Tufts University, 2021.
[21]G. Ellis, A. Dix, Density control through random sampling: an architectural perspective, Proceedings Sixth International Conference on Information Visualisation, 2002.
[22]A. Dix, G. Ellis, By chance enhancing interaction with large data sets through statistical sampling, Proceedings of the Working Conference on Advanced Visual Interfaces, 2002.
[23]A. Das Sarma, H. Lee, H. Gonzalez, Efficient spatial sampling of large geographical tables, Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, 2012.
[24]S.L. Lohr, Sampling: design and analysis, Chapman and Hall/CRC, 2019.
[25]E. Bertini, G. Santucci, Give chance a chance: modeling density to enhance scatter plot quality through random data sampling, Information Visualization 5(2)(2006) 95-110..
[26]A. Woodruff, J. Landay, M. Stonebraker, Constant density visualizations of non-uniform distributions of data, Proceedings of the 11th annual ACM symposium on User interface software and technology, 1998.
[27]A. Woodruff, J. Landay, M. Stonebraker, VIDA: (Visual Information Density Adjuster), CHI'99 Extended Abstracts on Human Factors in Computing Systems, 1999.
[28]D. Rafiei, Effectively visualizing large networks through sampling, VIS 05. IEEE Visualization (2005) 375-382.
[29]Z. Liu, B. Jiang, J. Heer, imMens: Real-time visual querying of big data, Computer Graphics Forum 32(3pt4)(2013) 421-430.
[30]C. Ahlberg, B. Shneiderman, Visual information seeking: Tight coupling of dynamic query filters with starfield displays, The craft of information visualization (2003) 7-13.
[31]H. Fang, S. Liang, Spatially and temporally continuous LAI data sets based on an integrated filtering method: Examples from North America, Remote Sensing of Environment 112(1)(2008) 75-93.
[32]W. Zhang, J. Qi, P. Wan, An easy-to-use airborne LiDAR data filtering method based on cloth simulation, Remote Sensing 8(6)(2016) 501.
[33]M. Procopio, A. Mosca, C.E. Scheidegger, Impact of Cognitive Biases on Progressive Visualization, IEEE Transactions on Visualization and Computer Graphics, 2021.
[34]K.A. Cook, J.J. Thomas, Illuminating the path: The research and development agenda for visual analytics, Pacific Northwest National Lab.(PNNL), Richland, WA (United States), 2005.
[35]M. Glueck, A. Khan, D.J. Wigdor, Dive in! Enabling progressive loading for real-time navigation of data visualizations, Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, 2014.
[36]S. Rinzivillo, D. Pedreschi, M. Nanni, Visually driven analysis of movement data by progressive clustering. Information Visualization 7(3-4)(2008) 225-239.
[37]J.-D. Fekete, ProgressiVis: A toolkit for steerable progressive analytics and visualization, 1st Workshop on Data Systems for Interactive Analysis (2015) 5.
[38]P.C. Wong, W. Cowley, H. Foote, Visualizing sequential patterns for text mining, IEEE Symposium on Information Visualization 2000. INFOVIS 2000. Proceedings. IEEE (2000) 105-111.
[39]P.C. Wong, Visual data mining, IEEE Computer Graphics and Applications 19(5)(1999) 20-21.
[40]C.D. Stolper, A. Perer, D. Gotz, Progressive visual analytics: User-driven visual exploration of in-progress analytics, IEEE Transactions on Visualization and Computer Graphics 20(12)(2014) 1653-1662.
[41]J. Ayres, J. Flannick, J. Gehrke, Sequential pattern mining using a bitmap representation, Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining, 2002.
[42]J.K. Li, K.-L. Ma, P5: Portable progressive parallel processing pipelines for interactive data analysis and visualization, IEEE transactions on visualization and computer graphics 26(1)(2019) 1151-1160.
[43]Y. Luo, C. Chai, X. Qin, Visclean: Interactive cleaning for progressive visualization, Proceedings of the VLDB

Endowment 13(12)(2020) 2821-2824.
[44]J. Jia, C. Li, X. Zhang, Towards interactive analytics and visualization on one billion tweets, Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, 2016.
[45]ISG-ICS/cloudberry: Big Data Visualization. <https://github.com/ISG-ICS/cloudberry>, 2016 (accessed 18.04.16).
[46]Cloudberry Middleware. <https://github.com/ISG-ICS/cloudberry/wiki/Cloudberry-Middleware>, 2017 (accessed 03.02.17).
[47]D. Florescu, A. Levy, D. Suciu, Optimization of run-time management of data intensive web-sites, VLDB 99(1999) 7-10.
[48]J. Jia, Supporting Interactive Analytics and Visualization on Large Data, University of California, Irvine, 2017.
[49]J. Jia, C. Li, M.J. Carey, Drum: A rhythmic approach to interactive analytics on large data, 2017 IEEE International Conference on Big Data, 2017.
[50]M. Ahmad, S. Duan, A. Aboulnaga, Predicting completion times of batch query workloads using interaction-aware models and simulation, Proceedings of the 14th International Conference on Extending Database Technology, 2011.
[51]W. Wu, Y. Chi, H. Hacígümüş, Towards predicting query execution time for concurrent and dynamic database workloads, Proceedings of the VLDB Endowment 6(10)(2013) 925-936.
[52]Register Dataset. <https://github.com/ISG-ICS/cloudberry/wiki/register-dataset>, 2019 (accessed 10.08.19).