BDViewer - A Web-Based Big Data Processing and Visualization Tool

Yan Li, Junming Ma, Bo An, Donggang Cao
Key Lab of High Confidence Software Technologies (Ministry of Education), Peking University
{liyan19960701, mjm520, anbo, caodg}@pku.edu.cn

***Abstract*—The size of data sets being collected and analyzed in data science field is growing rapidly, making traditional big data processing solution prohibitively expensive. Especially when the data sets are too large, distributed techniques are inevitable even for simple embarrassing parallel jobs. However, distributed computing is still inaccessible to a large number of users. For example, many average users are still struggling with complex cluster management and configuration tools[24] even just for summing up a group of numbers in a large data file. In this paper, we present *BDViewer*, a web-based big data processing and visualizing tool. *BDViewer* uses JavaScript plugins to enable users to view, process and visualize their large data files just through a web browser. By just clicking a button, users can open a large data file online and view the file contents immediately no matter how large the file is. In the back-end, *BDViewer* is built on a virtual private cloud system. Users' operations in a web browser are converted into map-reduce jobs and MPI tasks that are executed on the cloud. At the end of this paper, some experiments are carried out, which demonstrate *BDViewer*'s effectiveness and ease of use.**

Keywords—Big Data, Data Processing, Data Visualization, Distributed Computing, Parallel Computing

I. MOTIVATION

With the arrival of large data age, data size is increasing rapidly. Stand-alone data processing environment can hardly satisfy the need of data scientists. For example, Excel can only provide most of its features when data size is relevantly small. When file size achieves GB-level, Excel cannot even open it within an acceptable time. Some web based systems such as Google Sheet, Office Online, etc., also provide abundant data processing and visualization features. However, they cannot support large dataset quite well either. For instance, when a file is larger than 5MB, Excel Online can no longer open it.

A common solution for large dataset processing is to use distributed and parallel techniques. For example, users can rent a group of machines from AWS or Alibaba to deploy their own Spark or MPI clusters. However, those academic and commercially-successful platforms still present high barriers to entry for the average data scientist or scientific computing user. Many users are still having troubles with complex cluster configuring, even for running simple embarrassingly parallel jobs[24]. For example, a data scientist has to write a customized Hadoop/Spark program just to get the summation of a column in a large csv file.

We argue that a web-based large data processing and visualization solution is a viable choice for those users. This paper presents *BDViewer*[1], a web-based large data processing and visualization tool, which has the following characteristics: 1) users own their private virtual cluster where distributed processing environment is already set up. 2) large data automated processing is supported. 3) all a user need is a modern web browser to complete data processing in an interactive manner.

To evaluate *BDViewer*, some data processing and visualization experiments for big csv files are carried out. These csv files range from GB level to dozens of GB level. *SUM, SORT, MAX,* and some other commonly used operations are all performed on these files. And it turns out to be efficient and easy to use.

This paper is organized as follows. Section II details related work. Section III presents the architecture and design philosophy of *BDViewer*. Some key issues during implementation are explained in Section IV. Some experiments are presented in Section V. Finally, we conclude in Section VI.

II. RELATED WORK

BDViewer aims to process and visualize large data sets in a user-friendly way. Current data processing tools or data visualization tools, such as Excel, Google Sheet, Infogram, etc., usually do not provide large data sets supporting. Another warehouse based data processing system, Hive, supports distributed big data processing using simple SQL-like declarative language[17]. However, Hive is built on the top of Hadoop, which usually has a high latency. Besides, it's hard to use Hive for data visualization.

A. Data Processing Tools

Current data processing tools can be separated into two categories as stand-alone and cloud-based.

- **Stand-alone data processing tools.** When data size is not large enough, most users would like to analyze data in a PC environment. E.g., Excel and Spss can almost satisfy most researcher's needs. Furthermore, if one pursues more customized service, they can choose

[1] https://github.com/iteratorlee/BDViewer

0730-3157/18/$31.00 ©2018 IEEE
DOI 10.1109/COMPSAC.2018.00080

Python as their best option, which can be easily extended by plenty of third party packages, e.g., numpy, Pandas[4,9], etc. Each of these data processing tools is easy to operate and has a remarkable efficiency. However, due to stand-alone restraints, they seem helpless when facing large data file.

- **Cloud-based data processing tools.** This kind of tools run in a cloud environment. Users could rent a group of machines from cloud service vendors such as Alibaba and Amazon to build their own data processing clusters to achieve better performance. But this approach requires users to familiarize themselves with not only data analysis programming techniques but also the configuring of clusters. Thus, it is not a friendly way for non-professionals.

B. *Data Visualization Technology*

Data visualization technologies have been quite mature[5]. Tools either in stand-alone environment or in a web environment using JavaScript can transfer boring data into a vivid chart.

- Taking Excel as an example in a stand-alone environment, users can open a data file and do some graphics work or some calculation. But when file size is over a certain threshold, Excel will never open the whole file but load the starting lines instead. This strategy has the drawback that users cannot view the data beyond those loaded.
- Infogram[19] is a web-based data processing tool. Users can build their tables and draw beautiful charts based on it. However, there exist strict limits on the size of files to be uploaded. When file size reaches hundreds of MBs, both uploading, opening and visualization are quite slow.

C. *Warehouse Solution Based Data Processing*

Hadoop is a popular open-source map-reduce implementation which is being used as an alternative to store and process extremely large data sets on commodity hardware. However, the map-reduce programming model is very low level and requires developers to write custom programs which are hard to maintain and reuse.

Apache Hive[16-18] is a warehousing solution over a map-reduce Framework. Hive supports queries expressed in a SQL-like declarative language - HiveQL, which are compiled into map-reduce jobs executed on Hadoop.

However, what Hive emphasizes is large scale rather than real-time. Hive is built on Hadoop, which is based on static batch processing. Hadoop usually has a high latency and requires a lot of overhead in job submission and scheduling. For example, Hive typically performs queries on hundreds of megabytes of data sets with a typical time-of-day delay. Besides, Hive does not support interactive data visualization.

III. System Design and Architecture

A. *System Components*

Figure 1 shows the major components of *BDViewer*. The main components of *BDViewer* are:

- **Web Interface** – Based on Jupyter Notebook, *BDViewer* provides user-friendly web interfaces like interactive tabulation for csv file viewing and beautiful charts for data visualization.

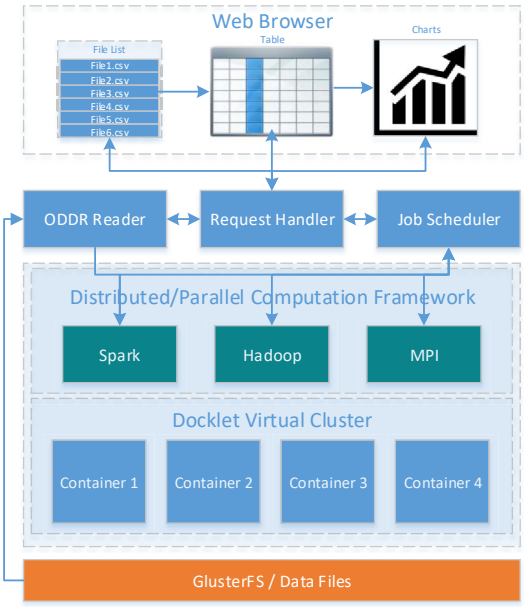

Figure 1. System Architecture of BDViewer

- **IO Components, Job Scheduler and Request Handler** behind front-end provide smart file loading and job scheduling. ODDR reader can load file data selectively according to the demand of computing tasks. For more details, see Section IV part B.
- **Distributed and Parallel** algorithms run in Spark/Hadoop/MPI clusters to support data processing acceleration. Users' operations in web browsers are automated converted into map-reduce and MPI tasks. For more details about these algorithms see Section IV part C.
- A **Virtual Cloud Operating System** at the bottom of *BDViewer* provides software environment for distributed and parallel computing. We adopt Docklet as the cloud system in *BDViewer*.

B. *Design Philosophy*

As is shown in Figure 1 and the previous part, some front-end techniques such as interactive JavaScript plugins, some distributed/parallel frameworks such as Spark/MPI and virtual cluster technique are adopted in this system. They all aim at solving the paradox between high performance/large data supporting and user-friendly manners.

- **Why Web-based?** When the data file is too large, it is necessary to run the distributed data processing jobs on remote clusters. For average users, their common approach to access to the Internet is using a web browser rather than using SSH in a terminal. So viewing and processing large data through a web browser is their most comfortable way. Besides, there are numbers of

JavaScript plugins such as D3.js, ECharts.js, etc. that can achieve a high level of data visualization. So it is very suitable to enable users to process and visualize their cloud-storage data through a web browser.

- **Why Spark/Hadoop?** Using Spark/Hadoop is inevitable when the size of data files exceeds the memory of the user's PC. In fact, quite a number of distributed jobs of data scientists are using MapReduce model. So we use Spark/Hadoop in *BDViewer* to support processing large data sets and executing MapReduce jobs.
- **Why MPI?** MPI is mainly used for *sort* in *BDViewer*. We noticed that quite a number of sorting operations are actually used for Top-K. For example, when an NLP scientist counts the word frequency of an article, he always only focuses on a few words that occur in this article most frequently. That means in many cases, people only care about the K^{th} largest/smallest elements of the sorting results. So in *BDViewer* we adopt pseudo sorting rather than full sorting in order to achieve better performance. A common Top-K algorithm using Spark is Partial QuickSort algorithm. But some experiments[6] have proved another Top-K algorithm using MPI is more efficient and scalable than Partial QuickSort. So in *BDViewer* we use MPI-based DC-Top-K[6] algorithm. For more details, see Section IV Part C.
- **Why Docklet?** Spark, Hadoop and MPI all need to be deployed in a cluster environment. And in many enterprises, it is a challenge to be able to run these heterogeneous applications, provide scalability and elasticity support for newly emerged frameworks, and most importantly, share cluster resources effectively. The key idea to solve this problem is Claas (cluster as a service), whose idea is virtualizing the cluster environment for distributed application frameworks. Most applications can directly run in the virtual cluster environment without any modification, which is a great advantage. Docklet is an implementation of Claas. It is easy to use Docklet to pre-configure the software environments and share a physical cluster among multiple users.

IV. SYSTEM IMPLEMEMTATION

In this section, four key issues will be explained in detail, which are separately Jupyter Notebook-based front end design, ODDR, data processing algorithms and data visualization approaches.

A. Front End Design Based on Jupyter Notebook

Jupyter Notebook is an interactive programming software. Users can view their data files stored on the cloud through the file list of Jupyter Notebook and do some online programming on it. Jupyter Notebook is very scalable. It supports developers extending the front end using its built-in *nbextension* mechanism. In the implementing of *BDViewer*, we adopt Handsontable.js as the csv file displaying plug-in to extend Jupyter Notebook. When Jupyter Notebook starts, the front end extension scripts of *BDViewer* modify the URL to which the entry of csv file in the file list points. Thus the csv file viewing page is redirected to a customized page where Handsontable.js transforms the csv file into a beautiful tabulation.

B. ODDR Reader

Selectively loading of a file is quite important in the implementation of *BDViewer*, which is essential in the following scenes. 1)When a user tries to open a large csv file, loading the whole file into RAM will bring high latency. Compromise must be taken to adapt large file loading. 2) When users scroll down the mouse, the following lines need to be loaded dynamically. 3) If users would like to view several certain lines of a file, it is necessary to load file selectively. What should be known is that file locating here is coarse-grained. Users cannot jump to a certain line by specifying the offset of a file because the size of each line might be not equal.

Pandas is a popular Python-based data analysis tool. In terms of table-format file, Pandas provides *read_csv* and *read_tabe* IO methods, which both have a C-based IO engine. Method *read_csv* can indicate a group of parameter to achieve customized IO behavior such as *skiprows* enabling users to skip several lines to get the following several lines, *nrows* enabling users to get certain number of lines and *chunksize* supporting a block-by-block reading of a file. These parameters ensure a convenient and efficient approach to read large table-format file.

Based on Pandas, this paper presents an On-Demand Data Reading strategy, referred to as ODDR. ODDR achieves a high performance reading of table file by indicating parameters *skiprows* and *nrows*. ODDR is applied in the following four situations: 1) when users open a file, the back end pre-loads and returns the first 1000 lines of the table file. 2) when mouse cursor scrolls down and exceeds the already loaded range, the backend will load the next 100 lines dynamically. 3) when users would like to jump to a certain line by indicating a specific line index, the backend reads 1000 lines right after the indicated line index. Pandas has a C based IO engine, being able to read file chunk by chunk. Therefore, ODDR consumes little time.

C. Data Processing Algorithms

BDViewer uses Docklet virtual cluster in the back-end for the parallel processing of tabular data. To accelerate data processing, we designed some customized distributed parallel algorithms. These algorithms run on the Spark cluster or MPI cluster on Docklet.

In order to accelerate the data processing more efficiently, we divide the commonly used calculation types into two categories according to their behaviors in memory.

The first category contains those operations whose intermediate calculation results do not take too much memory space, including *max*, *average*, *sum*, etc. E.g., when using map-reduce framework to solve a maximum number of an array, the memory space occupied by the intermediate computation becomes less and less as the data is merged until there only remains one single number. So this kind of operation can be directly implemented using current map-reduce framework. This paper designs a group of Spark/Hadoop algorithms to complete statistical operations of this category, running on the Spark/Hadoop cluster built on Docklet virtual cluster. Take *sum* as an example, the algorithm firstly uses the *map* function to

extract a certain column from the tabular file and then uses *reduce* function to calculate the sum of this column. The pseudo code is shown as Algorithm 1.

Algorithm 1: Parallel Sum

Input $filename; column$
Output sum
1. set up spark configuration
2. $textfile \leftarrow$ spark file of $'filename'$
3. $lines \leftarrow textfile.\text{flatMap}(x \rightarrow x.\text{split}('\backslash n'))$
4. $lines \leftarrow lines.\text{map}(x \rightarrow x.\text{split}(',')_{\text{column}})$
5. $sum \leftarrow lines.\text{reduce}(x, y \rightarrow x + y)$

Algorithm 1. Parallel Summation

But *sort* is quite different[13], the input scale and output scale is the same, which means the intermediate memory space will not decrease during the computing process[14]. On the other hand, considering that table files in *BDViewer* are presented to the user as a unit of a thousand rows of a display unit, the *sort* operation is actually simplified to find the top one thousand elements in the input data. This paper adopts a Top-K algorithm based on divide-and-conquer named DC-Top-K[6], which is a divide-and-conquer algorithm The pseudo code of DC-Top-K is shown as Algorithm 2.

Algorithm 2: DC Top k algorithm

Input $r_0, \dots, r_{n-1}; k$
Output $top_0, \dots, top_{k-1}$
1. Divide $r_0, \dots, r_{n-1}$ into k groups
2. **for** $i \leftarrow 0$ to $k-1$ **do**
3. $Max_i \leftarrow$ the maximum of group i
4. $Maxloc_i \leftarrow$ the position of Max_i in th group i
5. **endfor**
6. $threshold \leftarrow \min(Max_0, \dots, Max_{k-1})$
7. $G_{min} \leftarrow$ the group id of $threshold$
8. $Q \leftarrow 0$
9. **for** $i \leftarrow 0$ to $k-1$ **do**
10. **if** $i = G_{min}$ **then**
11. **continue**
12. **endif**
13. **for** $j \leftarrow 0$ to $N-1$ **do**
14. **if** $r_{i*N+j} > threshold$ & $j \neq Maxloc_i$ **then**
15. $r_Q \leftarrow r_{i*N+j}$
16. $Q \leftarrow Q + 1$
17. **endif**
18. **endfor**
19. **endfor**
20. **for** $i \leftarrow 0$ to $k-1$ **do**
21. $r_{Q+1} \leftarrow Max_i$
22. **endfor**
23. $top_0, \dots, top_{k-1} \leftarrow$ max_k($r_0 \sim r_{k+Q-1}$)
24. **return** $top_0, \dots, top_{k-1}$

Algorithm 2. DC-Top-K Algorithm

DC-Top-K is easy to be implemented using parallel computation framework. This paper presents a MPI-based parallel DC-Top-K algorithm. The whole algorithm takes place as 5 steps:

1) the master node calculates the line number of the file needed to be sorted, then gets the reading *bias* each worker node should begin reading from.

2) each worker node begins to read data from the *bias* calculated by the master node, then gets a local *threshold* and send it to the master node.

3) the master node gets the minimum value of all local *threshold*s as the global threshold and broadcast it.

4) each worker traverses local data to find those larger than global *threshold* and send them to the master node.

5) the master node collects all the data sent from each worker and performs a quick sort on them to get top *k* ones.

D. Data Visualization Approaches

This paper provides two kinds of data visualization. The first kind is visualization for partial data, and the other is for the whole data.

- **Visualization for partial data.** When a form file is opened, a user can select the appropriate area for histogram, line chart or pie chart drawing. The whole process happens at the front end because the data that need to be processed is small.
- **Visualization for whole data.** Visualization for the whole data is not directly performed at the front end using JavaScript. Because when data file is too large, the browser load will be very heavy when visualization is performed directly through the front end JavaScript[15]. So this paper uses the following strategy to visualize the overall data:

  1) user clicks a certain column and immediately the front end sent the column index to the back end.

Algorithm 3: Heatmap

Input $filename; column$
Output $heatmap$
1. front end send request to server
2. $column_{max} \leftarrow$ the largest number
3. $column_{min} \leftarrow$ the smallest number
4. k section $\leftarrow [column_{min}, column_{max}]$
5. **for** $section_i$ in $sections$ **do**
6. $freq_i \leftarrow$ number of elements in this interval
7. **endfor**
8. send $freq_0, \dots, freq_{k-1}$ to front end
9. front draw $heatmap$

Algorithm 3. Heatmap Algorithm

2) the back end processes the selected column to get the maximum and the minimum value.

3) the range [*min_val, max_val*] is distributed to 1000 sections, the back end calculates the frequency at which data falls in each subinterval and sends the frequency array to the front end.

4) the front end transfers the frequency array to a heatmap using the JavaScript plugin Heatmap.JS.

The pseudo code is shown as Algorithm 3. This algorithm is implemented using Spark

V. EXPERIMENTS

In this section, some experiments are carried out. Data files are stochastically generated csv files where data follows Gauss

distribution. Every file has 10 columns and for each column of data samples, the overall X has $X \sim N(500,150^2)$. Data size varies from 1GB to 10GB. The testbed is a Docklet virtual cluster with 4 virtual nodes. Each node is equipped with a virtual four-core Intel Xeon E5 2660 v4 CPU (2.00 GHz) processor. And each node has 4GB RAM and 4GB SATA disk with 7200 RPM. Each virtual node is configured with the same software environment such as Spark, MPI, etc.

First we tested *BDViewer*'s performances on basic statistical operation like *SUM* and *AVERAGE*. Next we evaluated the time consumption of Heatmap drawing. Then we judged ODDR's performance by letting it get the middle 1000 lines of some given files. Finally, we compared Naïve-Top-K algorithm and MPI-based DC-Top-K algorithm and demonstrated the effectiveness of the later one.

A. Csv File Opening Time Consumption

As the ODDR is designed to load the beginning 1000 lines when a user tries to open a file, so the file opening time is irrelevant to the file size. As is shown in Table 1, no matter how large the csv file is, the time consumption would be approximately 0.01 second.

File Size/GB	Opening Time Consumption/s
1	0.00952466
2	0.00921495
3	0.00913947
4	0.00862720
5	0.00882690
6	0.00963123
7	0.01019456
8	0.00911684
9	0.00973299
10	0.01020336
Average	0.00942121

Table 1. File Opening Time Consumption

B. Data Processing Operation Time Consumption

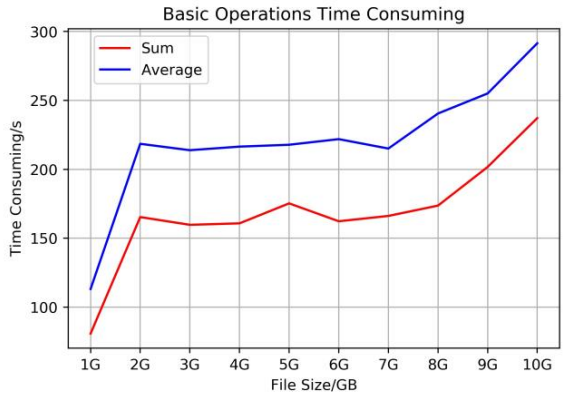

Figure 3. Data Processing Operation Time Consumption

When a user selects a column of an already opened file and then clicks the *Sum/Ave* button, the back end will run the specific Spark program to calculate the summation/average of the selected column. Figure 3 shows the time consumption of *Sum* and *Ave* operation. The tested files range from 1GB to 10GB. When the file size reaches 1GB, *BDViewer* can get the summation in 40~50 seconds and get the average in 2 minutes.

When the file size ranges from 2GB to 8GB, the *SUM* operation can complete within 2 minutes and the *AVE* operation can terminate within 4 minutes. And when file size is close or equal to 10GB, it will cost *BDViewer* less than 4 minutes to get the summation and less than 5 minutes to get the average value.

Figure 4 shows the performance of Heatmap drawing. It can be deduced that the heat map drawing time consumption approximately has a linear relationship with the file size. When file size achieves 10GB, it will take BDViewer about 15 minutes to get the Heatmap of the selected column.

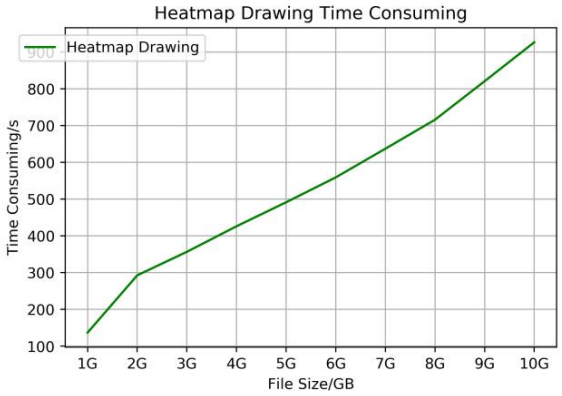

Figure 4. Heatmap Drawing Time Consumption

C. Skip-To Performance

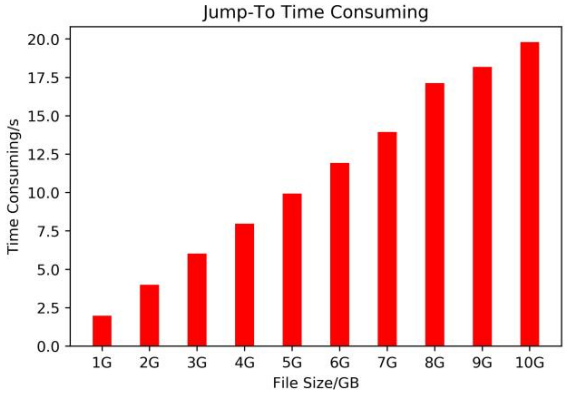

Figure 5. Skip-To Performance

When users click the *Skip To* button, the ODDR reader will return the next 1000 lines from the line that the user indicates. We test the performance of ODDR by letting it jump to the middle of the file. For examples, a 1G csv file in our experiment file sets has 25000000 lines so we let ODDR reader get the $12500000^{th} \sim 12501000^{th}$ lines and get the time consumption. Figure 5 shows the time consumption of *Jump-To-Middle* operation. It can be seen that the performance of ODDR is quite promising. Within 20 seconds, it can get the middle 1000 lines of a 10GB csv file.

D. Sorting Performance

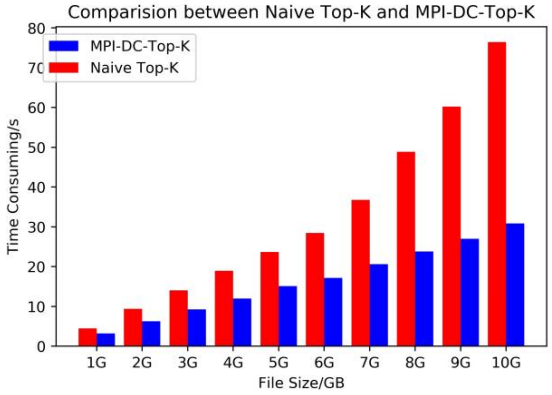

Figure 6. DC-Top-K Performance

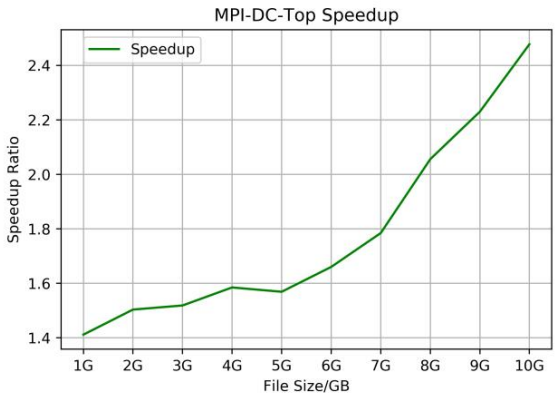

Figure 7. DC-Top-K Speedup

We evaluated the performance of MPI-based DC-Top-K algorithm by comparing it to a naïve Top-K algorithm. Naïve Top-K algorithm simply loads the whole file into RAM and then does a quick sort of it to get the top 1000 lines. MPI-based DC-Top-K runs in a 4-node Docklet cluster and each node runs two MPI tasks. Figure 6 shows the time consumption of this two algorithms. When file size if about 10GB, MPI-DC-Top-K can still get the top 1000 lines within half a minute. And the larger the file size is, the more obvious the speedup effect is. As is shown in Figure 7, while the file size increases linearly, the speedup ratio increases in a super-linear speed. When file size is 10GB, the speedup ratio can reach about 2.5.

VI. Conclusion

The complexity of configuring big data processing system and writing customized parallel/distributed programs presents a high barrier for a number of users. In this paper we present *BDViewer*, a web based big data processing and visualizing tool, to provide the user an ease-of-use approach to do big data processing and visualization.

Acknowledgment

This work is supported by the National Science and Technology Major Project under Grant No.2016YFB1000105; the National Natural Science Foundation of China under Grant No. 61272154, 91318301; the Science Fund for Creative Research Groups of China under Grant, No. 61421091.

References

[1] Bo An, Xudong Shan, Zhicheng Cui, Chun Cao and Donggang Cao, Workspace as a Service: an Online Working Environment for Private Cloud, 2017 IEEE Symposium on Service-Oriented System Engineering (SOSE), San Francisco, 2017, pp. 19-27. DOI:10.1109/SOSE.2017.11

[2] Donggang Cao, Bo An, Peichang Shi, Huaimin Wang, Providing Virtual Cloud for Special Purposes on Demand in JointCloud Computing Environment, JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY, 32(2):211-218, Mar 2017, DOI: 10.1007/s11390-017-1715-1

[3] Kluyver, Thomas, et al. "Jupyter Notebooks-a publishing format for reproducible computational workflows." ELPUB. 2016.

[4] McKinney, Wes. Python for data analysis: Data wrangling with Pandas, NumPy, and IPython. " O'Reilly Media, Inc.", 2012.

[5] Handbook of data visualization[M]. Springer Science & Business Media, 2007.

[6] Zhengyuan Xue, Ruixuan Li, Heng Zhang, Xiwu Gu, Zhiyong Xu, "DC-Top-k: A Novel Top-k Selecting Algorithm and Its Parallelization" of the 45th International Conference on Parallel Processing, 2016, pp. 372-375

[7] Hunter, John D. "Matplotlib: A 2D graphics environment." Computing In Science & Engineering 9.3 (2007): 90-95.

[8] Bo An, Junming Ma, Donggang Cao, Gang Huang, Towards Efficient Resource Management in Virtual Clouds, The 8th International Workshop on Joint Cloud Computing (JCC2017), Atlanta, USA, Jun 5-8, 2017.

[9] McKinney, Wes. "pandas: a foundational Python library for data analysis and statistics." Python for High Performance and Scientific Computing(2011): 1-9

[10] Dory, Michael, Allison Parrish, and Brendan Berg. Introduction to Tornado: Modern Web Applications with Python. " O'Reilly Media, Inc.", 2012

[11] Zaharia, Matei, et al. "Spark: Cluster computing with working sets." HotCloud 10.10-10 (2010): 95

[12] Gropp, William, Ewing Lusk, and Rajeev Thakur. Using MPI-2: Advanced features of the message-passing interface. MIT press, 1999

[13] Vitter J S. External memory algorithms and data structures: Dealing with massive data[J]. ACM Computing surveys (CsUR), 2001, 33(2): 209-271

[14] Yang H, Dasdan A, Hsiao R L, et al. Map-reduce-merge: simplified relational data processing on large clusters[C]//Proceedings of the 2007 ACM SIGMOD international conference on Management of data. ACM, 2007: 1029-1040

[15] Handbook of data visualization[M]. Springer Science & Business Media, 2007

[16] Thusoo A, Sarma J S, Jain N, et al. Hive: a warehousing solution over a map-reduce framework[J]. Proceedings of the VLDB Endowment, 2009, 2(2): 1626-1629.

[17] Thusoo A, Sarma J S, Jain N, et al. Hive-a petabyte scale data warehouse using hadoop[C]//Data Engineering (ICDE), 2010 IEEE 26th International Conference on. IEEE, 2010: 996-1005.

[18] Huai Y, Chauhan A, Gates A, et al. Major technical advancements in apache hive[C]//Proceedings of the 2014 ACM SIGMOD international conference on Management of data. ACM, 2014: 1235-1246.

[19] Infogram https://www.infogram.com

[20] Handsontable https://www.handontable.com

[21] Team R J S. Requirejs[J]

[22] Zikopoulos P, Eaton C. Understanding big data: Analytics for enterprise class hadoop and streaming data[M]. McGraw-Hill Osborne Media, 2011. MLA

[23] White T. Hadoop: The definitive guide[M]. " O'Reilly Media, Inc.", 2012.

[24] Jonas E, Pu Q, Venkataraman S, et al. Occupy the Cloud: Distributed computing for the 99%[C]//Proceedings of the 2017 Symposium on Cloud Computing. ACM, 2017: 445-451.