International Information and Engineering Technology Association
Advancing the World of Information and Engineering

Revue d'Intelligence Artificielle
Vol. 37, No. 1, February, 2023, pp. 231-237
Journal homepage: http://iieta.org/journals/ria

Tree-Based Approach's to Mitigate the Heterogeneity Concerns among Different file Systems: A Possible Solution

Sheikh Amir Fayaz[1], Majid Zaman[2*], Iqbal Hasan[3], Waseem Jeelani Bakshi[1], Sameer Kaul[1]

[1] Department of Computer Sciences, University of Kashmir, J&K 190006, India
[2] Directorate of IT & SS, University of Kashmir, J&K 190006, India
[3] Department of Computer Sciences, Jamia Millia Islamia, Delhi 110025, India

Corresponding Author Email: zamanmajid@gmail.com

https://doi.org/10.18280/ria.370129

Received: 21 December 2022
Accepted: 8 Feburary 2023

Keywords:
heterogeneous file systems, DFS, BFS, windows, Linux

ABSTRACT

It might be quite difficult to search for words or phrases in the many file formats that are based on various operating systems. To deal with this level of heterogeneity in different file systems, several academics have put up a number of alternative strategies. Given that data is stored in various formats and is controlled by several operating systems, such solutions, however, proved to be extremely time-consuming. The suggested techniques work best when the data is kept in a single source. The idea of bottlenecking and the resulting heterogeneity in file systems are two main issues with looking for a certain collection of data objects (folder, file, or directory) across many platforms (Windows, Linux, and so forth). We made an effort to suggest fundamental searching methods that may deal with the issue of heterogeneity while increasing efficiency and maintaining dependability. Our method makes use of a concept known as tree-based breadth first search (BFS) and depth first search techniques (DFS) to limit to an absolute minimum the amount of I/O operations that might be required in the heterogeneous environment. The experiment was run on Windows and Linux computers, and it was discovered that by using these strategies, the heterogeneity issues may be greatly decreased, leading to some encouraging outcomes.

1. INTRODUCTION

In the last two decades, data has grown at an exponential rate, and data analysis has become more important for many real-world challenges in fields such as health, weathercasting, academic, and commercial sectors, among others, since it provides value by providing useful insights from data. As we progress toward huge data sources with massive volumes of data, the potential for important discoveries grows; nevertheless, so does the volume of worthless data [1, 2]. Furthermore, the data to be processed is commonly stored in a number of formats, ranging from fully organised to semi-structured to completely unstructured, such as free text. As a result, the challenges in processing all of the data available today are in expressing and answering questions [3].

The idea of heterogeneity may be observed in the many storage platforms supported by various file systems such as Linux and Windows. Windows stores data in its architecture, but Linux stores data in its architecture. The user wishes to access data from both heterogeneous systems, and several strategies have been developed to overcome the heterogeneity [4]. It is critical to extract vital information from many platforms. These sources of information might contain various sorts of data, such as data stored at database levels and various file systems such as pdf/txt and HTML file formats. Many researchers have used different techniques in order to tackle with the heterogeneity concerns which include parallel system reliability analysis with a CECBO algorithm [5], HAS systems at parallel systems [6], keyword based searching techniques [7], indexing techniques [8], integrating different data sources [9] and, other various techniques [10] and so on [11, 12]. All of these approaches have their pros and cons and they mainly security concerns and permanent and transient failures. As a result, a good approach to break the heterogeneity levels across storage devices run by heterogeneous platforms is required. The technique should operate in a heterogeneous environment, allowing data sharing and searching across many different platforms; providing massive scalability of data, servers, and clients; ensuring high availability of customer data; and incorporating centralized, automated storage and data management, which helps to reduce storage management costs.

Therefore, in order to give true insights, we want a one-of-a-kind system that can function and execute queries from one or more file systems, such as in Linux and Windows. In general, looking for data in various directories and folders of Windows and Linux requires distinct commands since the operating systems are different. However, this method is restricted in scope because it only works with specific file system types. To search in each and every directory, we must employ some promising data structure strategy and technology. The main objective to carry out this study is to provide individual users with simple and flexible access to their own data and break the heterogeneity constraints when accessing the data. Because personal data is very sensitive, privacy and ethical concerns must be addressed when working with this sort of information.

The study is organised as follows: Section 2 elaborates the most recent research on heterogeneity, followed by Section 3 display of the various operating system search strategies, and

Section 4 definition of the generic file systems used by Windows and Linux. The implementation of the tree-based algorithms is further demonstrated in Section 5, the evaluation findings are defined in Section 6, the overall paper is discussed in Section 7, and the study is finally concluded in Section 8.

2. LITERARURE REVIEW

Since there are many approaches which are currently in process that deals with the heterogeneity and those include a keyword based approaches, SAN system approach, indexing technique and many more. Some of the solutions provided by many researchers are mentioned below:

In the study [13], the impact of heterogeneity was introduced and elaborated upon, and the overall performance was determined by conducting an experiment. The experiment involved accessing the parallel Orange file system using IOR with four distinct HServers and four separate SServers.

In all, about 16 servers were employed in these trials. Overall, the performance of hybrid clusters slightly outperforms that of homogeneous clusters with four low-speed HServers. As a result, the results indicated that the more diverse the hardware, the worse the overall performance of the system.

A study titled "Multi-Source Heterogeneous Data System Architecture and Technology for Telecom Operator" [14] was proposed based on a new multi-source heterogeneous data (MSHD) with four modules: data access, data pre-processing, data storage and administration, and data application and visualization for telecom operators, in order to effectively access multisource platforms and make use of valuable data resources. Internal telecom applications as well as vertical industry applications may benefit from the MSHD system.

In order to handle the data heterogeneity in share-disk file systems the authors [15] developed and evaluated a method for load management in shared-disk file systems built on heterogeneous computer clusters. The study distributes file sets among cluster server nodes to balance file metadata workload and also reacts to changing server resources caused by failure and recovery, as well as dynamically adding or deleting servers. Furthermore, they used a technique known as adaptive, non-uniform (ANU) randomness to continually optimize load allocation. ANU randomization exploited the benefits of hash-based, randomized placement approaches in terms of scalability and metadata minimization. It also avoided the downsides of hashing, such as load skew, inability to deal with heterogeneity, and lack of processability. The simulation results suggest that our load-management method outperforms a predictive approach.

The authors [16] suggest FlexGate, a heterogeneous data centre gateway that combines configurable packet processing hardware and software gateways to accommodate large amounts of inbound/outbound traffic. They presented two function concepts based on observations in operational data centres that may fully use the processing benefits of hardware/software platforms. In addition, two load balancers were used to efficiently distribute traffic to hardware pipelines/software gateways. Experiments revealed that the FlexGate technique can handle at least 1.5Tbps of traffic with an average delay of 1.28 micro seconds.

Heterogeneity of parallel file systems (PFS's) always remains a concern and many studies are proposed and out of which a study on HAS: Heterogeneity-Aware Selective Layout Scheme for Parallel File Systems on Hybrid Servers was taken into consideration for hybrid PFSs [17], where a unique heterogeneity-aware selective data layout technique was presented. HAS reduces inter-server load imbalance by skewing data delivery across heterogeneous servers based on storage performance. Depending on a newly designed selection and distribution algorithm, HAS adaptively picked the ideal data layout from three typical candidates based on the application's data access patterns to greatly increase the overall system's I/O efficiency. The authors of the study used HAS inside OrangeFS to enable efficient data distribution for data-intensive applications. The results confirmed that HAS greatly improved the I/O performance of hybrid PFSs when compared to conventional data layout pareto - optimal solutions.

The authors [18] developed a graph-based keyword search solution in heterogeneous data sources, where as the amount and diversity of data sources rise, query response must accommodate for heterogeneous data models with varied structures, or maybe no structure at all. The authors devised a novel query approach for exploiting such disparate information using keyword search. Given a list of search terms, the authors arranged their underlying data as graphs, and their algorithm develops linkages between them inside and across the varied datasets included in the graph. They expanded on previous work on keyword search in structured and unstructured data, which they improved by adding the data heterogeneity dimension, which makes the keyword search problem computationally more challenging. They made use of both synthetic and real-world datasets to construct the method and assess its performance.

Because all of the above research had limitations, we provide a possible solution to the heterogeneity problem. Among the present issues are data processing, data management, and, most critically, data storage. Many academics are inspired to construct sophisticated heterogeneous systems by data processing; yet, assessing enormous amounts of data need more computer capability. Furthermore, data management and storage are ongoing difficulties with no clear solution in sight. As a result, the major goal of our research is to solve these present challenges. All of the systems under consideration support just a small number of data sources. Wrappers must be produced manually or hard-coded, and it appears that just a few data sources are supported.

3. HOW SEARCHING WORKS IN DIFFERENT OPERATING SYSTEMS

By default, Windows file searching scans not only the file name but also the contents of the file. Even if we have a super-fast nonvolatile memory express storage SSD, we ask the operating system to go through and inspect every byte on the drive, and each drive may potentially carry billions of bytes of data, so this searching procedure can take a very long time and sometimes frustrating. Therefore, in order to alleviate the searching challenge, Microsoft employs the indexing approach, in which material is indexed and then searched. For example, it keeps note of the most often used place, and when a search is performed, it initially scans that folder exclusively, and if there is a match, it returns a hit; otherwise, it returns a miss. Furthermore, a windows system has one more means of searching which is using command prompt where a simple command can also reach us to the respected file. In the case of

Linux operating systems, we normally don't have a searching window and instead utilize a terminal-based search using various Linux commands. Actually, regardless of the operating system, the libcolombus library algorithms are acting as searching algorithms inside. These techniques include index based searches and contain files like libcds/libcds2 [19] for exact match string, Lucene/Solr [20] for full text searching and information retrieval and so on.

However, one of the key concerns of searching across multiple platforms is whether or not the heterogeneity is truly removed. It is evident that it does not eliminate the issues of heterogeneity. Thus, in this work, we are attempting to provide an overview of a method that may reduce heterogeneity regardless of the operating systems used.

4. GENERAL FILE SYSTEMS: WINDOWS AND LINUX

As we all know, Windows and Linux have various file systems, and each file system has its own storage space. These sections are divided into directories, folders, sub-folders, and finally files [21]. Files might be of any format or type. The basic file systems of Windows (Figure 1) and Linux (Figure 2) are depicted here.

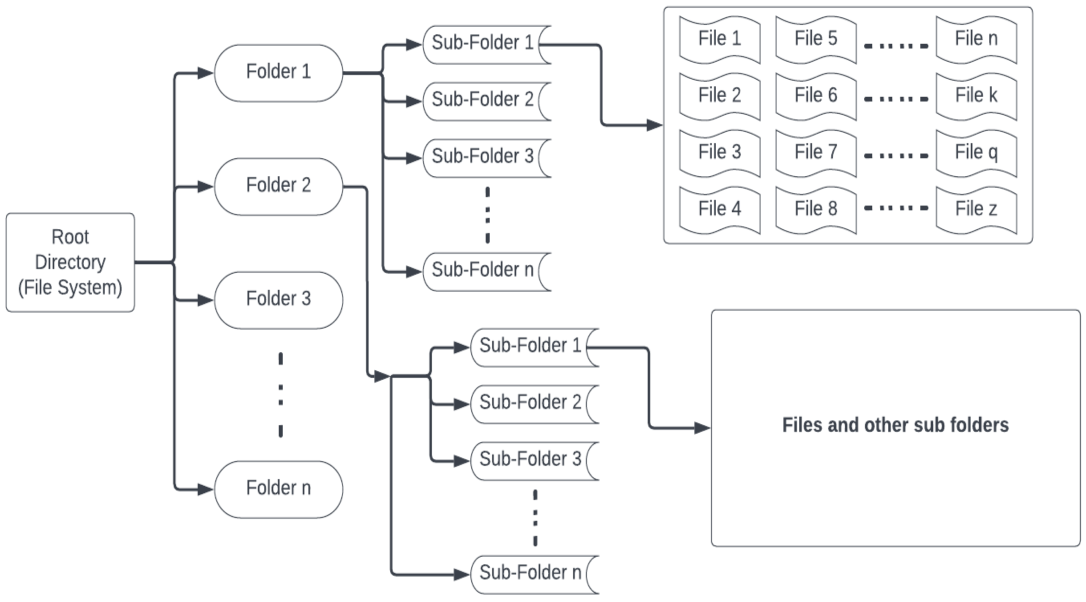

Figure 1. Basic Windows file system

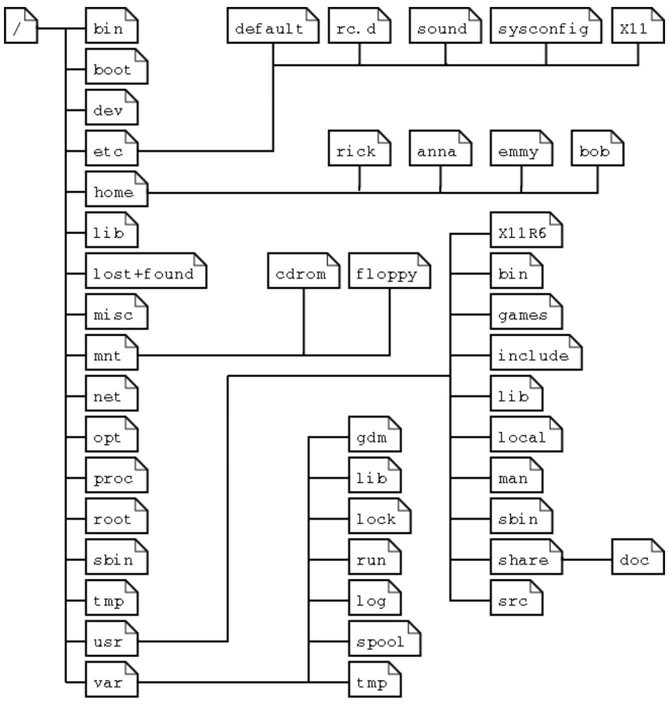

Figure 2. Basic Redhat Linux file system (Source Google)

The file system manages how data is retrieved and saved by keeping track of the disc and its locations. Internally, there are no partitions in the storage discs, but the structure is presented for simple retrieval and storage of the data so that we can understand the data structure, such as its directories and folders, and so on. In Linux, the file system tree begins at the slash, which is symbolized by a forward slash (/). This directory, which contains all underlying directories and files, is sometimes known as the file system's root directory [22]. To highlight their location and avoid confusion with other directories that may have the same name, directories that are just one level below the root directory are sometimes prefixed with a slash. When starting with a fresh system, it is usually a good idea to have a peek in the root directory [23]. Thus, all files systems at least supports the following storage components which include volume: where it is a collection of directories and files, directories: where it is a hierarchy of other directories and files and last if the files: where it is a collection of similar type of data.

5. TREE-BASED APPROACH TO MITIGATE THE HETEROGENEITY CONCERNS: IMPLEMENTATION

BFS and DFS are popular algorithms in computer science and have been used in a variety of applications. In the context of alleviating heterogeneity problems in different data sources, BFS and DFS can have several advantages over other approaches. Some of these advantages include:

- Flexibility: BFS and DFS can be applied to different types of data structures, including trees and graphs, making them highly versatile for a range of applications.
- Ease of Implementation: BFS and DFS algorithms are relatively simple to understand and implement, compared to other complex algorithms.
- Optimality: In some cases, BFS and DFS can provide optimal solutions, as they can be used to find the shortest path or the deepest node in a tree, for example.
- Time Complexity: Both BFS and DFS have time complexities that are linear in the number of nodes in the data structure, making them efficient for large data sets.

Of course, these advantages must be evaluated in light of the specific requirements of each problem, and there may be other approaches that are better suited for a particular use case. But, in general, BFS and DFS can be effective and efficient methods for alleviating heterogeneity problems in different data sources.

To reduce heterogeneity concerns in different operating systems, we will build a rough sketch of the searching algorithms that may be applied to prevent such issues. Breadth first search (BFS) and Depth first search (DFS) are tree traversal strategies included in these algorithms (DFS). BFS and DFS are most often graph based algorithms used to examine each and every node of the data structure.

5.1 Working of DFS algorithm in file systems

Suppose we need to search a particular file in a system which is present in any operating system be it Linux or windows. The general approach in the depth first search algorithm is that it will follow the syntax of the searching of a particular operating system and then it begins searching with the root directory and proceeds deeper until we discover the specific file or file with no path [24].

Because it is a recursive nature, stack data structure can be used to build the DFS algorithm. First, it generates a stack with the first root directory of the file system. It now selects any directory as the starting point of the traverse and adds it to the stack. Push a non-visited directory or file to the top of the stack (next to the vertex at the top of the stack). This step is continued until all directories and files have been visited. If there are no more directories/folders/files, go back and pop the same off the stack. Repeat the steps until the stack is empty i.e. all the files are traversed. Based on the architecture shown above we will implement the DFS algorithm in order to reach to a particular file we are searching for. The general pseudo-code of DFS algorithm implemented in both Linux and Windows systems is shown below:

```
Depth_FS (Node, Children)    //Node can be directories/Folders/Files and
Children denotes the files which are to be traversed
    Children.visited = true
    for each x ∈ Node.Adj [Children] // x denotes the file/s which is/are to
be searched
        if x.visited == false
            Depth_FS(Node, x)
init () {                              // Initialization step
    for each Children ∈ Node
        Children.visited = false
     For each Children ∈ Node
       Depth_FS(Node, Children)    //Function call
}
```

In this pseudocode, node is the current node being processed and neighbor is an unvisited neighbor of the node. The DFS function marks the current node as visited, and then iterates over its unvisited neighbors, calling the DFS function on each of them. This recursive process continues until all nodes have been visited.

Note that this is a basic implementation of DFS and can be modified based on the specific requirements of each problem. Additionally, this pseudocode assumes that the data structure being traversed is represented as a graph, but it can also be adapted for use with trees or other data structures.

The general working of the above pseudocode of DFS is shown below diagrammatically (Figure 3).

The following diagram depicts the general organization of both Windows and Linux file systems, with the parent node being the root node, which is the directory of the specific file system. Because there are n directories in the system and we have only presented a single directory in the preceding graphic. Only these modules are relevant to this study; other modules could be used as the research's capstone project. Following that, the directories contain the folders, and those folders include the file subfolders. The DFS algorithm operates in such a way that it first traverses the directory and then its associated children, and then files of that selected child, and this process employs a stack and alternately push and pop operations until the specific file is searched.

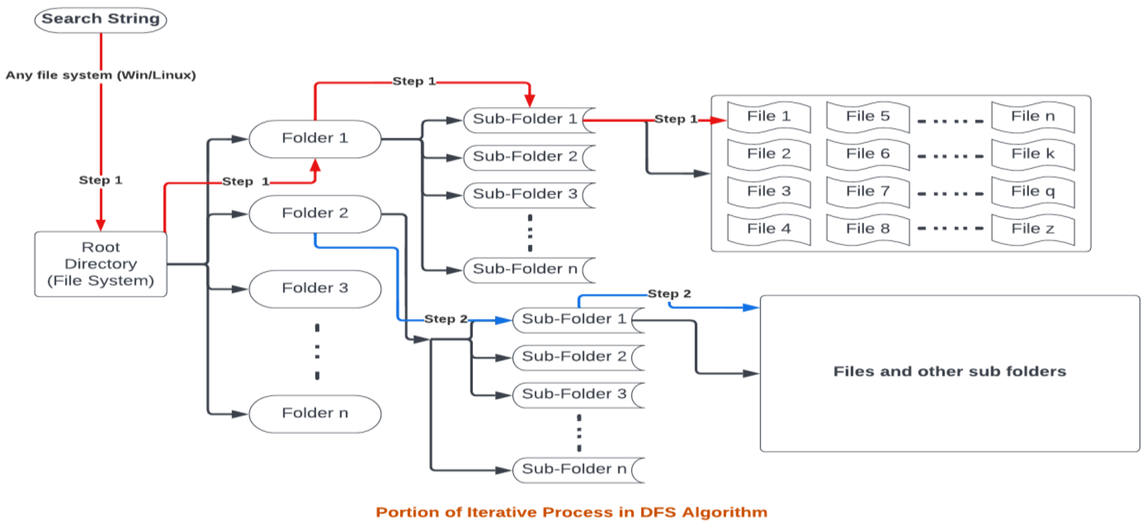

Figure 3. Working of DFS in removing heterogeneity at different file systems

5.1.1 Complexity concerns in DFS

In general, if a system contains n directories/files/folders/subfolders, the complexity of the DFS algorithm will be $O(n)$ [25]. Assume the file system is represented as an adjacent list, with each node keeping track of all associated nodes. Thus the surrounding nodes will have a linear time complexity and the temporal complexity will be:

$$O(n) + O(E) = O\ (n + E)$$

Furthermore, as it follows a structure of undirected traversal, and therefore it appears twice and the complexity will be:

$$O(n) + O(2E)\ O(n + E)$$

In addition, the file system will be coupled with a matrix list, and the DFS algorithm will have a complexity of:

$$O(nxn) = O(n)^2$$

Thus the complexity of DFS algorithm in case of file systems will remain $O(n)^2$. Furthermore, the space complexity in case of DFS will remain $O(n)$ only because it keeps the track of only last visited file/directory/folders of a system.

5.2 Working of BFS algorithm in file systems

The process of traversing each node of the graph is known as breadth-first search. BFS begins with a node, then examines all nodes one distance away from the beginning node, then all nodes two distances away, and so on. BFS use a queue to retrieve the nodes to be visited [26].

The general pseudo-code of the BFS algorithm is shown below:

```
Queue (Q)
Node n visited and add n into Q
While Q not-empty
      Delete folders/directories/files/sub-folders C of Q
      enqueue the visited neighbours of folders/directories/files/sub-
      folders
```

5.2.1 Complexity concerns in BFS

The temporal complexity of the BFS algorithm is $O(n+E)$, where n represents the number of nodes and E indicates the number of connected links. In addition, the BFS algorithm has an $O(n)$ space complexity [26, 27].

6. EXPERIMENTAL EVALUATION

On single PCs with 8GB RAM and 250GB HDD configured with dual boot operating systems (Linux and Windows), we carried out an experiment. The implementation was purely done using python and its associated libraries. In this study, tree-based searching algorithms were created on various platforms (Linux and Windows). Before beginning the implementation process, we assume that we have complete access to all files/directories and folders on both Linux and Windows computers. Following the implementation of the algorithms, we obtained the following results:

Both file systems are built differently depending on the operating system environment. The data saved in the system is accessible to any systems using this file system, regardless of operating system or hardware platform. In our experiment, the implementation of BFS and DFS breaks the degrees of heterogeneity because we simply use different syntax to access the file and the rest of the job remains the same. Our goal was to provide a solution that outperformed other ways for breaking heterogeneity constraints, and these strategies appear to be performing well. Both these searching approaches doesn't break any sort of semantics of the system although it maintain local file system semantics in a network system independent of the platform which we are using.

Furthermore, the same experiment was performed on two individual system with almost same file systems and we feel that the configuration of the systems play major role in these experiments in case of time complexity, but the level of heterogeneity has been removed totally.

7. THEORETICAL ANALYSIS

Breadth-first search (BFS) and depth-first search (DFS) are commonly used algorithms for traversing and searching graph and tree structures. When applied to data integration, these algorithms can be used to alleviate heterogeneity problems that arise from differences in the data sources being integrated.

Heterogeneity can arise from differences in data formats, data structures, data contents, and more. In order to integrate these sources, the data must be transformed and mapped to a common format. BFS and DFS can be used to search the data structure and identify differences between the sources, allowing for the development of mappings that align the data [28, 29].

For example, BFS can be used to perform a top-down search of the data structure, starting from a known common ancestor node and expanding to its descendants. This approach can be useful when the data structure is organized in a hierarchical manner, and the goal is to find a common format that spans all sources. On the other hand, DFS can be used to perform a bottom-up search of the data structure, starting from the leaf nodes and working towards the root. This approach can be useful when the data structure is more complex and there is no clear hierarchy, and the goal is to find the closest common format for each set of nodes.

In comparison to other techniques for alleviating heterogeneity problems in different data sources, BFS and DFS have several advantages. For example, they can be more flexible than techniques that are based on predefined ontologies, as they can adapt to different data structures and formats. They can also be more efficient than techniques that rely on manual mapping, as they can automate the mapping process. Additionally, BFS and DFS can be more scalable than techniques that rely on manual mapping, as they can handle large data sets with ease.

However, it is important to note that BFS and DFS are not always the best choice for every use case. In some cases, other techniques such as semantic web technologies or machine learning algorithms may be more suitable, depending on the specific requirements of the task at hand. In these cases, a comparative analysis should be performed to determine which approach is best suited for the problem at hand.

In conclusion, BFS and DFS are effective algorithms for alleviating heterogeneity problems in different data sources. They have several advantages over other approaches, including flexibility, efficiency, and scalability, but it is important to compare them with other techniques to determine their suitability for a particular use case.

8. DISCUSSION

One of the key concerns in this study was to remove the heterogeneity among the file systems which are operated by different operating systems. We used a tree based search algorithms to check if the heterogeneity level is reduced or not. We assumed that we have full access to the data stored in the files systems and by the experimental analysis we observe that the level of heterogeneity level gets reduced because by applying a particular syntax only we are able to get the data which we are looking for. These DFS and BFS algorithms also perform in a manner that the data can be searched in two ways: by searching those folder/s/files which are closer to the directory and by searching those file/s/folders which are away to the root directory. With this these are also suitable for decision making like in case of puzzles as well. While in case of DFS there is one big advantage of backtracking which help in reducing the time complexity issues in searching the particular file and thus it is considered as one of the best technique in comparison with BFS. Additionally, the user-returned results are far less than the responses trees found, demonstrating the importance of minimality as a criterion for delivering the results. These methods also exactly match the word string.

9. CONCLUSION AND FUTURE WORK

To eliminate heterogeneity in searching huge sets of heterogeneous documents on different platforms (windows and Linux in this study), has shown to be a powerful, responsive, and resilient approach, which is the major purpose behind providing BFS and DFS. We have discussed our performance optimization methods and the architecture of BFS and DFS on Windows and Linux systems. With comparison to the previous system, all improvements made to the system have significantly improved performance by reducing reaction time. Our method aids in improving search performance in distributed and heterogeneous file systems since these approaches are extremely responsive and fault-tolerant. We are currently employing the same methodology to work on three other file systems, as well as parallel file systems, and we are hoping that these approaches will perform significantly better in every heterogeneous, parallel, and distributed setting.

REFERENCES

[1] Kenny, M., Schoen, I. (2021). Violin SuperPlots: Visualizing replicate heterogeneity in large data sets. Molecular Biology of the Cell, 32(15): 1333-1334. https://doi.org/10.1091/mbc.E21-03-0130

[2] Kaul, S., Fayaz, S.A., Zaman, M., Butt, M.A. (2022). Is decision tree obsolete in its original form? A Burning debate. Revue d'Intelligence Artificielle, 36(1): 105-113. https://doi.org/10.18280/ria.360112

[3] Fayaz, S.A., Zaman, M., Butt, M.A. (2022). Numerical and experimental investigation of meteorological data using adaptive linear M5 model tree for the prediction of rainfall. Review of Computer Engineering Research, 9(1): 1-12. http://dx.doi.org/10.18488/76.v9i1.2961

[4] Li, Z., He, Y., Yu, H., Kang, J., Li, X., Xu, Z., Niyato, D. (2022). Data heterogeneity-robust federated learning via group client selection in industrial IoT. IEEE Internet of Things Journal, 18: 17844-17857. https://doi.org/10.1109/JIOT.2022.3161943

[5] Cheng, J., Zhao, W. (2021). Parallel system reliability analysis with a CECBO algorithm. Structural and Multidisciplinary Optimization, 64(1): 71-88. https://doi.org/10.1007/s00158-021-02857-8

[6] Michlowicz, E., Wojciechowski, J. (2021). A method for evaluating and upgrading systems with parallel structures with forced redundancy. Eksploatacja i Niezawodność, 23(4). https://doi.org/10.17531/ein.2021.4.19

[7] Qi, L., He, Q., Chen, F., Dou, W., Wan, S., Zhang, X., Xu, X. (2019). Finding all you need: web APIs recommendation in web of things through keywords search. IEEE Transactions on Computational Social Systems, 6(5): 1063-1072. https://doi.org/10.1109/TCSS.2019.2906925

[8] Mishra, P., Poddar, R., Chen, J., Chiesa, A., Popa, R.A. (2018). Oblix: An efficient oblivious search index. In 2018 IEEE Symposium on Security and Privacy (SP), pp. 279-296. https://doi.org/10.1109/SP.2018.00045

[9] Subramanian, I., Verma, S., Kumar, S., Jere, A., Anamika, K. (2020). Multi-omics data integration, interpretation, and its application. Bioinformatics and Biology Insights, 14: 1177932219899051. https://doi.org/10.1177/1177932219899051

[10] Willms, E., Cabañas, C., Mäger, I., Wood, M.J., Vader, P. (2018). Extracellular vesicle heterogeneity: Subpopulations, isolation techniques, and diverse functions in cancer progression. Frontiers in Immunology, 9: 738. https://doi.org/10.3389/fimmu.2018.00738

[11] Altaf, I., Butt, M.A., Zaman, M. (2021). A pragmatic comparison of supervised machine learning classifiers for disease diagnosis. In 2021 Third International Conference on Inventive Research in Computing Applications (ICIRCA), pp. 1515-1520. https://doi.org/10.1109/ICIRCA51532.2021.9544582

[12] Fayaz, S.A., Zaman, M., Butt, M.A. (2021). An application of logistic model tree (LMT) algorithm to ameliorate Prediction accuracy of meteorological data. International Journal of Advanced Technology and Engineering Exploration, 8(84): 1424-1440. http://dx.doi.org/10.19101/IJATEE.2021.874586

[13] He, S., Sun, X.H., Feng, B. (2014). S4D-cache: Smart selective SSD cache for parallel I/O systems. In 2014 IEEE 34th International Conference on Distributed Computing Systems, pp. 514-523. https://doi.org/10.1109/ICDCS.2014.59

[14] Xu, L., Cui, G., Hu, X., Liu, S., Jia, Y., Zhang, T., Gao, J., Dong, R., Zhou, Y., Cheng, X., He, X. (2021). Architecture and technology of multi-source heterogeneous data system for telecom operator. In: Wang, Y., Xu, L., Yan, Y., Zou, J. (eds) Signal and Information Processing, Networking and Computers. Lecture Notes in Electrical Engineering, vol 677. Springer, Singapore. https://doi.org/10.1007/978-981-33-4102-9_120

[15] Wu, C., Burns, R. (2003). Handling heterogeneity in shared-disk file systems. In SC'03: Proceedings of the 2003 ACM/IEEE Conference on Supercomputing, pp. 7-7. http://dx.doi.org/10.1109/SC.2003.10045

[16] Qian, K., Ma, S., Miao, M., Lu, J., Zhang, T., Wang, P., Sun, C., Ren, F. (2019). Flexgate: High-performance

heterogeneous gateway in data centers. In Proceedings of the 3rd Asia-Pacific Workshop on Networking 2019, pp. 36-42. https://doi.org/10.1145/3343180.3343182

[17] He, S., Sun, X.H., Haider, A. (2015). HAS: Heterogeneity-aware selective data layout scheme for parallel file systems on hybrid servers. In 2015 IEEE International Parallel and Distributed Processing Symposium, pp. 613-622. https://doi.org/10.1109/IPDPS.2015.23

[18] Anadiotis, A.C., Haddad, M.Y., Manolescu, I. (2020). Graph-based keyword search in heterogeneous data sources. In BDA 2020-36ème Conférence sur la Gestion de Données–Principes, Technologies et Applications. https://doi.org/10.48550/arXiv.2009.04283

[19] F. Claude. libcds Compact Data Structures Library. Website. 2008. https://github.com/fclaude/libcds.

[20] Apache Software Foundation. Apache Lucene. Website. 2014. https://lucene.apache.org, acessed on Sep. 24, 2022.

[21] Neyaz, A., Shashidhar, N. (2019). USB artifact analysis using windows event viewer, registry and file system logs. Electronics, 8(11): 1322. https://doi.org/10.3390/electronics8111322

[22] Mohd, R., Butt, M.A., Baba, M.Z. (2020). GWLM–NARX: Grey Wolf Levenberg–Marquardt-based neural network for rainfall prediction. Data Technologies and Applications, 54(1): 85-102. https://doi.org/10.1108/DTA-08-2019-0130

[23] Fayaz, S.A., Zaman, M., Butt, M.A. (2022). A hybrid adaptive grey wolf Levenberg-Marquardt (GWLM) and nonlinear autoregressive with exogenous input (NARX) neural network model for the prediction of rainfall. International Journal of Advanced Technology and Engineering Exploration, 9(89): 509. https://doi.org/10.19101/ijatee.2021.874647

[24] Gopisetty, R., Ragunathan, T., Bindu, C.S. (2020). Improving performance of the distributed file system using speculative read algorithm and support-based replacement technique. International Journal of Advanced Research in Engineering and Technology, 11(9). pp. 602-611.

[25] Perpignan, C., Robin, V., Eynard, B. (2018). From ecodesign to DFS in engineering education. In DS 93: Proceedings of the 20th International Conference on Engineering and Product Design Education (E&PDE 2018), Dyson School of Engineering, Imperial College, London, 6th-7th September 2018, pp. 622-627.

[26] Zhang, F., Lin, H., Zhai, J., Cheng, J., Xiang, D., Li, J., Chai, Y., Du, X. (2018). An adaptive breadth-first search algorithm on integrated architectures. The Journal of Supercomputing, 74(11): 6135-6155. https://doi.org/10.1007/s11227-018-2525-0

[27] Rehman, A., Butt, M.A., Zaman, M. (2021). A survey of medical image analysis using deep learning approaches. In 2021 5th International Conference on Computing Methodologies and Communication (ICCMC), pp. 1334-1342. https://doi.org/10.1109/ICCMC51019.2021.9418385

[28] Fayaz, S.A., Zaman, M., Kaul, S., Butt, M.A. (2022). Is deep learning on tabular data enough? An assessment. International Journal of Advanced Computer Science and Applications, 13(4). https://dx.doi.org/10.14569/IJACSA.2022.0130454

[29] Fayaz, S.A., Kaul, S., Zaman, M., Butt, M.A. (2022). An adaptive gradient boosting model for the prediction of rainfall using ID3 as a base estimator. Revue d'Intelligence Artificielle, 36(2): 241-250. https://doi.org/10.18280/ria.360208