Composite file Candidates identification based on Clustering

Team: pandora's box
Apurva Katti, Hussain Md Mehedul Islam, Rituparna Khan

***Abstract*—Existing systems provided the one-to-one mapping of logical files to physical metadata. A new approach, [1] proposed composite-file file system with the technique of many-to-one mapping and explored different mapping strategies by studying the metadata of the files. As an extension of the composite-file file system, our work mainly aims at extending the existing work and thereby exploring different strategies to combine files (cluster) together. We have implemented the composite file-file system using the directory-based strategy, time-based strategy, and session-based strategies. We have explored those strategies by grouping them together using both frequent mining techniques and clustering techniques. FP-Growth algorithm and dynamic hashing and pruning based Apriori helped us to realize files accessed as transactions whereas the clustering algorithms like the K-Means and Hierarchical Agglomerative Clustering helped us realize them as clusters. By using different approaches to clustering the files, we could have more data to do benchmark testing on Log and Web Log data. The clusters and transactions have been evaluated by using a testing benchmark in the experimentation phase and we have considered different workloads to infer the results.**

I. Introduction

File system performance optimization is one of the most popular research areas in the field of the Operating systems. Common optimizations of file system use the one-to-one mapping phenomena of logical files to their physical meta data representations. Existing experiments show that in $>80\%$ of cases access to files is smaller. These smaller files mostly are accessed together where their meta data is nearly the same. This understanding is the base of the Composite-file file system(CFFS) where many logical files are grouped together associated with a single i-node with other attributes. This approach significantly reduces the overhead of accessing smaller files since their metadata is the main dominating field in case of smaller files.

One of the in-house analysis [1] confirmed that more than 80% of accesses are to files smaller than 32 bytes. So, about 40% of this access time is used for meta data access, which can be reduced by decoupling one-to-one mapping strategy. Another Observations show that many files share similar file attributes, like the number of file owners, permission patterns, etc. which are limited. Edel et al [2] present a work where upto 75% meta data was compressed. So we can infer that there are some opportunities to optimize the redundant meta data information. Files tend to be accessed together, as shown by [3]–[6]. Pre-fetching might be an important optimization but fetching separately for each smaller files is an overhead. For example, accessing 32 small files can incur 50% higher latency than accessing a single file with a size equal to the sum of the 32 files, even with warm caches. These observations impose merit of performance improvement by consolidating smaller files that are accessed together.

The Composite-file File System(CFFS) introduces an intrinsic physical representation called a composite file, which holds the content of small files that are frequently accessed together. The original information stored in small files' inodes are deduplicated and stored as extended attributes of a composite file. The extended attributes also record the locations within the composite file for individual small files. The CFFS can be configured three ways.

- **Directory-based Consolidation:** files within a directory (excluding sub directories) form a composite file.
- **Embedded reference consolidation:** file references within file contents are extracted to identify files that can form composite files.
- **Frequency-mining-based consolidation:** files that are accessed together frequently form composite files.

In this course project, we will work on this frequency-mining based consolidation- find the candidates sets for composite files consolidation using several clustering algorithms. We will study about different clustering methodologies and perform frequency analysis since composite file system deals with concatenating files that are accessed together.

II. Literature Survey

The idea of composite file system is based on grouping files with similar metadata. [1] mentions about identifying composite file membership based on three factors: Directory based consolidation, embedded reference and frequently accessed files. Directory based consolidation is considered because of spatial locality available around the directories in current file systems. For consolidating files based on directories, sub-directories were excluded. Embedded based consolidation references file system based on embedded file references in files. For example, hyperlinks may be embedded in html files and a web crawler can be used to access each web page based on the links. Embedded based consolidation maybe useful while accessing directories, however it is difficult to pull references apart from text based format. Also, we need to know the specific file format for doing consolidation. Frequency based consolidation is used for identifying frequently accessed files. The general idea is that,if a set of files is accessed frequently, its subsets must be frequent as well. Apriori algorithm is used for frequency based consolidation and explained. Since, the

itemset generation based on Apriori was high, they could not use it extensively. For our implementation, we have introduced FP-growth for identifying frequently accessed files and is explianed in detail in teh later sections.

[7] mentions about doing survey on various frequent pattern mining algorithms using a transactional database. It is based on data of a smart home where sensors are installed on daily usage objects and patients while performing their daily tasks, touch these objects and these sensor items are maintained in database. In this paper they concluded FP growth algorithm is faster than Apriori and it scans the entire database much less number of times.

[8] mentions about applying frequent mining on web log data. They have divided web log into three categories: (i) Web content mining, (ii) Web structure mining and (iii) Web usage mining. Data mining on web logs were done by following the usual steps: preprocessing, pattern discovery and pattern analysis. For understanding the minimum support threshold values that should be used, they did some experimentation. The frequent itemset discovery and association rule mining was achieved using ItemsetCode algorithm with different minimum support and minimum confidence threshold values. They used two other algorithms namely PD-tree and SM-tree for frequent mining. Their main objective is to find the hidden information from the web log data.

[9] Have considered frequent pattern for looking through web pages. The pattern is the set of web pages visited together in a session, whose support is above the minimum support threshold. Frequent patterns are helpful to identify pages accessed together in one session. They have aimed for the following in their paper: (1) examine the features of the web log data and frequent patterns presented in the data; (2) compare the performance of the representative FP mining approaches – candidate-generation-and-test approach (Apriori based) and pattern-growth approach (FP-growth based) on mining FPs from web logs. Their contribution is as follows: (1) extensive experiments have been conducted to show the difference of the web log data from the artificial supermarket transaction data and relational data, in particular, the average and maximal transaction size, the number of frequent items, the number of frequent patterns and the maximal length of frequent patterns are examined; (2) extensive experiments have been conducted to show the performance of Apriori and FP-growth on real world web log data in contrast to artificial supermarket transaction data and relational data;
(3) a frequent pattern mining algorithm – Combined Frequent Pattern Mining (CFPM) algorithm is proposed. The main idea of CFPM is to combine candidate-generation-and-test approach and pattern-growth approach. Besides using a predefined minimum support threshold, they have used a predefined minimum length (L) of FPs. In CFPM, short FPs with length less than or equal to L are mined by candidate-generation-and-test approach, while long FPs with length greater than L are mined by pattern-growth approach. We have implemented the CFPM algorithm.

[10] Mentions about designing SSH (Sketch, Shingle and Hashing) as a faster hashing scheme. SSH uses a novel combination of sketching, shingling and hashing techniques to produce (probabilistic) indexes which align with dynamic time warping(DTW) similarity measure. The generated indexes are then used to create hash buckets for sub-linear search. Their results show that SSH is very effective for longer time sequence and prunes around 95% candidates, leading to the massive speedup in search with DTW. DTW is most widely used similarity measure for time series because it combines alignment and matching at the same time. However, branch and bound based pruning are only useful for very short queries and the bounds are quite weak for longer queries. Due to the loose bounds branch and bound pruning strategy boils down to a brute-force search. The focus of this paper is on the problem of similarity search with time series data. Data independent hashing techniques derive from the rich theory of Locality Sensitive Hashing(LSH) and are free from all the computational burden. Hashes can be invariant to spurious transformations on time series such as shifting. In this paper they have provided a data independent hashing scheme which respects alignments. The focus is on data-independent hashing schemes which scale favorably.

[11] proposes new clustering approach based on page's path similarity for navigation patterns mining. They have worked with web logs and considered various parameters for clustering. For example, for user identification they have mentioned about using IP address, cookies, direct authentication. Other parameters considered are session identification which can be done based on timestamps. After all the preprocessing, they have transformed the data in a way that can be used for clustering algorithms. For example, converting strings to integers or truncating date fields. With all these, they have tried to build a navigation pattern mining by building two matrices, one representing website user's opinions and the other one representing pages' path similarity. They have also considered co-occurence matrix and path similarity matrix. The path similarity matrix is built using filepaths and considering directories in the filepaths. Using these matrices, they have used a DFS algorithm for clustering.

[12]Mostly mentions about pre-processing web logs for using them in clustering. The pre-processing steps removes the following from the web log: The entries having suffixes like .jpg, .jpeg, .css, .map etc., Entries having status code failure, all records which are not contain method " GET", navigation sessions performed by Crawler ,Spider, and Robot. For user identification they have considered these: Different IP addresses refer to different users, the same IP with different operating systems or different browsers should be considered as different users, while the IP address, operating system and browsers are all the same, new user can be determined whether the requesting page can be reached by accessed pages before, according to the topology of the site. Users is uniquely Identified by combination of referrer URL and user agent. After identifying userIDs, they have identified sessions. After all these steps, they have considered pattern recognition. This phase consists of different techniques derived from various fields such as statistics, machine learning method mainly have Association Rules, data mining, pattern recognition, etc. applied to the Web domain and to the available data. The clustering algorithms used by them are K-Means, Fuzzy C-

Means, Neural Network, Association rule. For prediction they have used Decision tree, Support Vector Machines and KNN. For pattern mining, they have used Structured Query Language (SQL), Online Analytical Processing (OLAP).

[13]This paper majorly introduces the techniques used for Session Identification in HTTP web requests.A request represents the data of the HTTP request made by the User which gets recorded in the log files. A session is a set of requests made by the user machine in a certain interval of time. Sessions are formed by the heuristics mentioned in the paper.This idea is used in our work as well.

[14]This paper introduces new techniques for cleaning, User Identification,Session Identification, Transaction Identification.Traditional user identification is carried out based on certain established heuristics. 1) Different IP address refer to different users,since each IP address is unique to every host system.2) The same IP address with different operating system or different browsers should be consider as different user.3)While the IP ,operating system and browsers are all the same, new user can be determined based on the fixed time-window.The sessions are generated by using a dynamic timeout method rather than the fixed time window.The first uses a fixed value of 30 minutes (1800 seconds)or any othe value can be used depending on the number of requests made by the user which indicates the end of each session and the second using the average time spent on the pages of the website by users.

[15]This paper proposes the creation of sessions by using both User Agent and IP address along with a fixed time window which has been implemented in our work as well.It forms the basis of our sessionization method.The paper also has implemented the HAC algorithm for clustering.We have used the HAC algorithm for clustering the similar files but with a slight modification.Instead of using the maximum number in the similarity matrix, we have simply used the complete linkage method of HAC.

[16]This paper introduces the technique for identifying sessions based on ICA(Independent Component Analysis).A session matrix is constructed which consists of navigation patterns.It represents the unique web pages of the website browsed over a period of time.A new matrix is created after normalizing each pattern in the matrix with mean and standard deviation.ICA is applied to new patterns and the minimum weighted values are deleted form the matrix.Penalized Posterior probability based Fuzzy C-Means (PPPFCM) algorithm for clustering user navigation patterns data is also proposed to group the user navigation patterns.Fuzzy clustering depends on the probability of degree of belongingness of the data and based on this we infer that the group belong to the same user navigation patterns instead of belonging to single cluster.

[17]Bisecting k-Means is the combination of k-Means and hierarchical clustering. Instead of partitioning the data into 'k' clusters in each iteration, Bisecting k-means splits one cluster into two sub clusters at each bisecting step until all the k clusters are obtained.There are a number of different ways to choose which cluster to split. We could use the largest cluster at each step to split our cluster or the one with the least overall similarity, or a criterion based on both size and overall similarity.

[18]This paper introduces the method of improving the updation of initial centroids in K-Means clustering algorithm.In K-Means the centroids are usually randomly chosen.In the modified K-Means algorithm, the inital cluster centroid is updated by adding a delta value which is the average distance value of each cluster.The paper also proposes a similarity measure to understand the relationship between several clusters called the variable length vector distance(VLVD).We could not implement this method because we were using K-Means from sklearn and it does not allow us to write our own user-defined function for updation of centroids.

III. Dataset Preprocessing

We have worked on two different types of log to understand the composite-file file system.

A. User Log:

This log is in an extended format. It contains the following informations:

- **Name of the system** - For example, Exec.chrome.log.txt
- **Timestamp** - The time that the server finished processing the request.
- **Request from client**-GET/lib/x86.64-gnu/libtinfo.so.5
- **Http Status Code**
- **Bytes of the file**

*1) **Preprocessing**:* The user log that we received was not in proper format. Also, it contained junk data which had to preprocessed before proceeding further. We read the logs and converted it into a pandas dataframe in python for making it easier for us to read. In that dataframe we made lot of changes to keep only the parts of the log that we were useful. The steps followed for preprocessing are as follows:

- When converted the data to a dataframe, the missing values in the log were marked as '-'. We removed these items from the dataframe.
- The timestamp in the logs had permissions attached to it. We wanted only the timestamp to cluster based on session or time window. We removed the permissions and preserved only the timestamp parameter.
- We removed status code and bytes accessed.
- The filepaths in the log file came along with HTTP request headers. We wanted just the filepaths to create transactions. Hence, we removed the request headers.
- There were many filepaths which contained only directories. We looked for filepaths containing only files and removed the rest.

  Challenges faced: For keeping the files, we initially tried to loop through the entire dataframe. It resulted in our systems getting unresponsive as looping through 6,87,405 lines was not a good idea. Finally, we could find a solution to filter the logs for keeping only files with the help of a dataframe function.

B. Server Log:

This is a department trace log and it is in a combined log format. It contains the following information:

- **IP address:** IP address of the client or remote host.
- **Timestamp:** The time that the server finished processing the request.
- **Request:** This provided us filepaths along with the HTTP request headers. The filepaths had userIds attached to it.
- **HTTP Status Code**
- **Bytes of the file**
- **Useragent string**: The useragent string provided information about the browser and browser versions used by users.

*1) **Preprocessing**:* The server log was in a much better format compared to the user log. However, there were things that we did not require and we continued to pre-process them. We converted the log into pandas dataframe and here are the preprocessing steps:

- We removed missing values from the log. These missing values showed up as '-' in the dataframe.
- We removed permissions from the timestamp in the log.
- We removed status code and bytes accessed.
- We removed all HTTP request headers present along with the filepaths in the log. The request headers removed were 'GET','POST','HEAD','OPTIONS' and 'PROPHEAD'
- The filepaths had various unnecessary informations like 'favico.ico' which suggests that a user visited a webpage and an image icon link was fetched as a request. We retained only filepaths which had userIds appended to it. Since, the department gathers information of users accessing webpages, the presence of userIds are of utmost importance.

IV. Transaction processing

The frequent-pattern mining algorithms used are FP-Growth and Apriori. These algorithms require transaction lists as an input to give us an idea about the frequent itemsets. It is an important step and for creating the transactions, we have used several metrics. Here are the approaches we followed for creating transactions:

- **Time-window and Directory-based transactions**: From the log file we have used time-stamp as a metric. First, we considered files accessed in one second and grouped them together. Second, from the first file group, we have looked for files present in the same directory. Finally, we formed a list with files accessed within one second and occurring in same directories.

- **Time window based transactions**: From the log files, we have considered only time-stamp as a metric to group files. We have kept files accessed within a certain time-frame in a list. For example, files accessed within accessed within a time-window of one, four and seven seconds are kept in separate lists.

These transactions were passed as an input to FP-Growth and Apriori algorithms, to generate clusters.

Apart, from clustering based on frequent-pattern mining algorithms, we have considered other metrics to form clusters. Here are the different approaches:

- **Directory based**: While observing the log files, we found that several types of files are present in the same directory. We wanted to group those files together and hence we pre-processed our dataframe to fetch only directories from the filepaths. After that, we compared the directories with the dataframe and fetched all filepaths matching the directories. Following this approach, we formed several clusters. This is the most basic approach to generate cluster from the log file.

- **Session based**: This metric is used for generating clusters on the server log as the metrics used were present only in server log. These transactions were created based on user sessions. We considered IP address and User agent string to identify a user session. Combination of IP address and User agent string effectively identified users and hence we could use it to generate cluster. We have followed hierarchical clustering algorithms to generate clusters with these metrics.

V. Algorithm

A. Apriori

The Apriori algorithm [19] is a classical data mining algorithm. It is used for the extraction of frequent components and association rules. It is designed to operate on a database that contains many transactions, for example items brought into a store by customers. Before drive into the deep, lets explain some key concept used in apriori:

Support: The support of an itemset X, $supp(X)$ is the proportion of transaction in the database in which the item X appears. It signifies the popularity of an itemset.

$$supp(X) = \frac{\text{Number of transaction in which } X \text{ appears}}{\text{Total number of transactions}}$$

Confidence: Confidence of a rule is defined as follows:

$$conf(X->Y) = \frac{supp(X \cup Y)}{supp(X)} \tag{1}$$

It signifies the likelihood of item Y being purchased when item X is purchased.

Lift: The lift of a rule is defined as:

$$lift(X->Y) = \frac{supp(X \cup Y)}{supp(X) * supp(Y)} \tag{2}$$

In our exploration of a frequency-mining-based consolidation, we use a **direct hashing and pruning** based variant of the Apriori algorithm [20]. **DHP** reduce the exploration of candidate generation with the early pruning with the lift associated with each items. We have used a python library for DHP algorithm for our experiment. For simplicity, we illustrates the basic apriori with example in figure 1.

property 1.1: For any frequent item set mining, If a set of items are frequent then all of it's subset must also be frequent.

Property 1.1 is a significant observation behind all frequent item-set mining algorithms. 1 illustrates the apriori algorithm with an access stream to files A, B, C, D, and E (Threshold = 2)

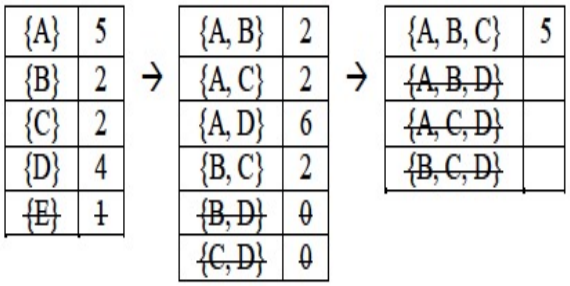


Figure 1: Steps for the Apriori algorithm to identify frequently accessed file sets for a file reference stream E, D, A, D, A, D, A, B, C, A, B, C, A, D. [1]

Initial pass: In the beginning, Apriori count the number of accesses for each file, and then it removes files with counts less than the pre-set threshold for further analysis.

Second pass: In the next phase, it takes the remaining files from the first phase and permutes to build all possible two-file reference sets. For example: if file A access after B or vice versa, it increments the count for A, B. After all counting again remove those below threshold (e.g., B, D).

Third pass: It generates all three-file reference sets based on the remaining two-file reference sets. According to properly 1.1 file sets such as A, B, D are pruned, since B, D is eliminated in the second pass.

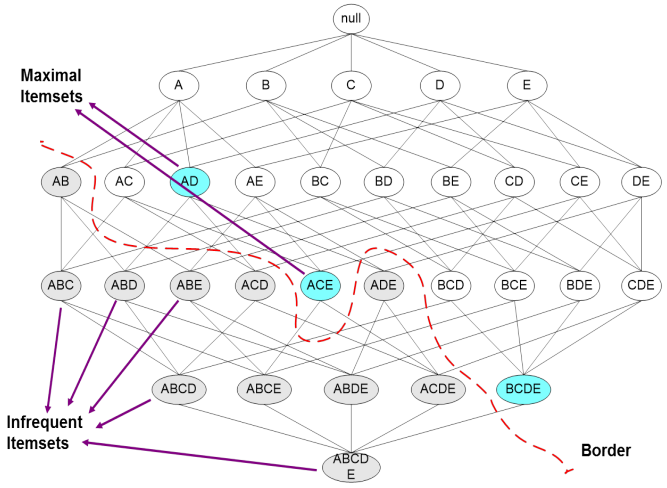


Figure 2: Maximal Frequent Itemset Example [21]

As we can no longer generate four-file reference sets, the algorithm ends. Now, we have {A},{B},{C},{D},{A,B},{A,C},{A,D},{B,C}and {A,B,C} item-sets remaining. Clearly lots of item-sets are subset of other, like: {A,B} is a subject of {A,B,C}. At this point, we keep only the maximal frequent itemset {A,D}, {A,B,C}. **An itemset is maximal frequent if none of its immediate supersets is frequent**. 2 well explain the maximal frequent itemset with an example.

B. FPGrowth:

The FP-Growth Algorithm, proposed by Han in [22], is an efficient and scalable method for mining the complete set of frequent patterns by pattern fragment growth, using an extended prefix-tree structure for storing compressed and crucial information about frequent patterns named frequent-pattern tree (FP-tree). We used python fpgrowth library to identify composite file candidates identification.

FP-Growth follows two step approach and allows frequent itemset discovery without candidate itemset generation.

- **Step 1:** Build a compact data structure called the FP-tree Built using 2 passes over the data-set.
- **Step 2:** Extracts frequent itemsets directly from the FP-tree.

1) FP-Tree Construction: FP-Tree is constructed using 2 passes over the data-set.

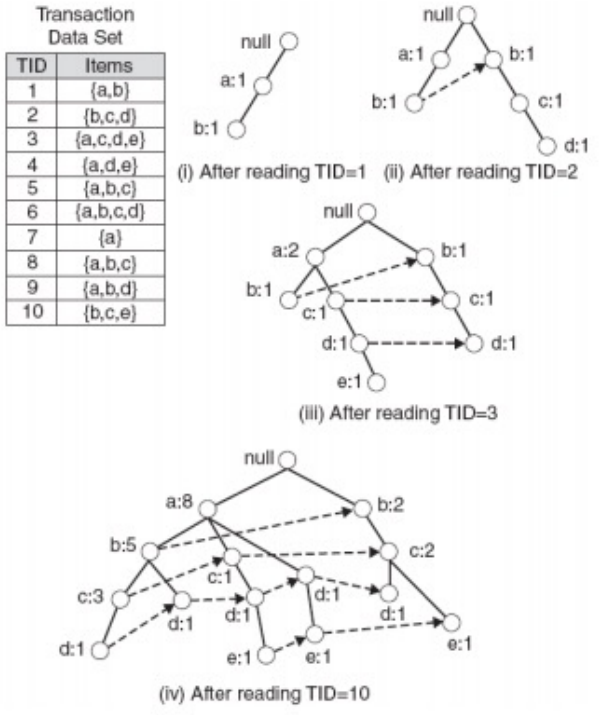


Figure 3: FP-Tree Construction (Example)

Pass 1:

- Scan data and find support for each item.
- Discard infrequent items.
- Sort frequent items in decreasing order based on their support.

FP-Growth use this order when building the FP-Tree, so common prefixes can be shared.

Pass 2:

- FP-Growth reads one transaction at a time and maps it to a path.
- It uses fixed sorted order created from pass 1, so paths can overlap when transactions share items(when they have the same prfix).

- Pointers are maintained between nodes containing the same item, creating singly linked lists (dotted lines)

The more paths overlap, the higher the compression. If the compression is high, FP-tree may fit in memory. After this step we got FP-tree from there Frequent itemsets will be extracted.

2) Frequent Itemset Generation: FP-Growth extracts frequent itemsets from the FP-tree using bottom-up algorithm - from the leaves towards the root. So, First, it extract prefix path sub-trees ending in an item(set). see the example prefix path sub-trees 4 constructed from the complete FP-tree.

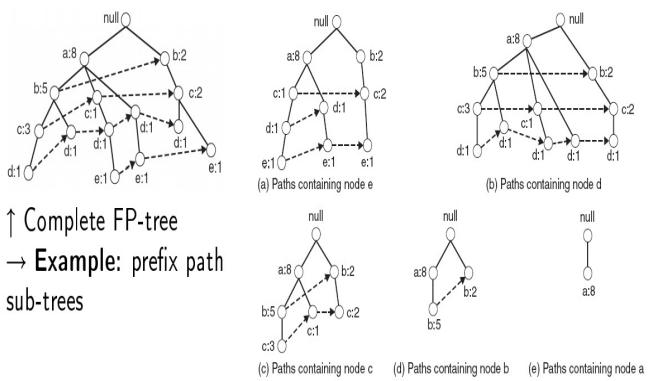


Figure 4: Prefix path sub-trees (Example)

Then each prefix path sub-tree is processed recursively(Divide and conquer approach) to extract the frequent itemsets. Solutions are then merged to return whole list of frequent itemset.

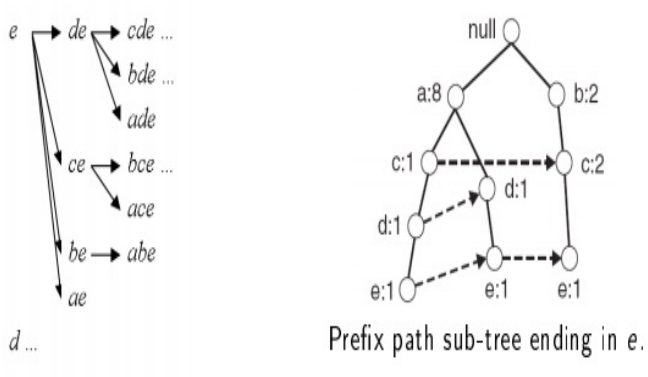


Figure 5: The prefix path sub-tree for e will be used to extract frequent itemsets ending in e, then in de, ce, be and ae, then in cde, bde, cde, etc(Example)

Finally, to extract the frequent itemset we built conditional FP-Tree that would be built if we only consider transactions containing a particular itemset (and then removing that itemset from all transactions).

TID	Items
~~1~~	~~{a,b}~~
~~2~~	~~{b,c,d}~~
3	{a,c,d,e}
4	{a,d,e}
~~5~~	~~{a,b,e}~~
~~6~~	~~{a,b,c,d}~~
~~7~~	~~{a}~~
~~8~~	~~{a,b,e}~~
~~9~~	~~{a,b,d}~~
10	{b,c,e}

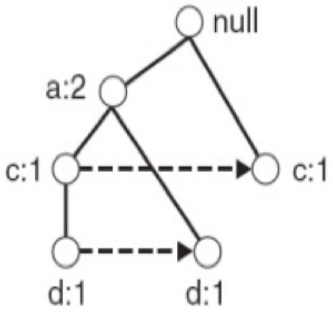
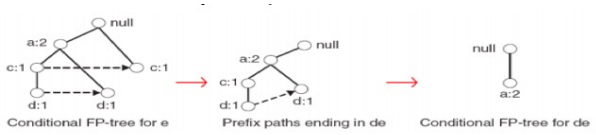
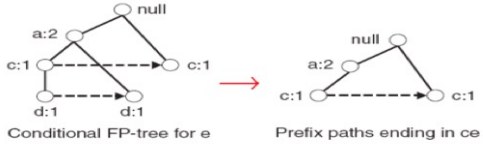


Figure 6: FP-Tree conditional on e

Let's see an example figure 7 of frequent itemset mining with FPGrowth. Let $minSup = 2$ and extract all frequent itemsets containing e. Figure 5 is the prefix path sub-tree for e. Check if e is a frequent item by adding the counts along the linked list (dotted line). If so, We extract it. Yes, count =3 so $\{e\}$ is extracted as a frequent itemset. As e is frequent, find frequent itemsets ending in e. i.e. de, ce, be and ae. Then we need to use the the conditional FP-tree for e to find frequent itemsets ending in de, ce and ae. Note that be is not considered as b is not in the conditional FP-tree for e. Then for each of them (e.g. de), find the prefix paths from the conditional tree for e, extract frequent itemsets, and generate conditional FP-tree (recursive).



(a)



(b)

Figure 7: FP-Growth Example (a) $e \rightarrow de \rightarrow ade(\{d,e\},\{a,d,e\})$ are found to be frequent (b) $e \rightarrow ce(\{c,e\})$ is found to be frequent

VI. Sessionization

Web based mining methods become important in order to study in depth about the increasing web-based applications and to derive and discover patterns. Since the data considered is huge, we had to come up with an idea to segregate it into smaller subsets and study the behavior on them. This is usually done by parsing the web log data in the form of tokens and they are uniquely identified to help us classify the file paths. The web log data provided several useful information(fields) to assert the underlying concepts of web log records. As

discussed previously, we had to pre-process certain irrelevant information from our data.

Next, in order to establish the relationship between several lines of web log data, we used the concept of sessionization. User Session Identification is the process of segmenting the user activity log of each user into sessions each representing a single visit to a website [15]. As per our data, instead of the visits to a particular website, we considered the visits to a file. We were able to conveniently extract two important features of the likes of IP address and User Agent which represent a single user. In many of the papers on web log mining they have considered using only IP address to identify a user, where as we decided to use both IP address and User Agent since both can equally help us to define a user better. There is also one more parameter called the User ID, which when used in combination with IP address and User Agent greatly help to identify a user much better but we could not use this information in our case since it was not possible to directly extract the UserID from the data. IP address represents the username of the host system which is assigned by the Internet Service Provider (ISP) and User Agent is nothing but the name of the browser which the user uses to send the requests to the web server. For example, Mozilla Firefox, Googlebot etc.

Using the heuristic that identifies a single user by using both IP address and User Agent and that permits only a fixed time gap between two requests, we can identify the sessions. Any new request can belong to the session already identified if it has the same IP address and User Agent as the request in the previous sessions and that this request was put in the fixed time window. The idea of using both the parameters is simply to avoid complications that arises when the requests arrive from the proxy. If two or more users use the same proxy, then they can get identified by their same IP. Hence the User Agent helps identify better. In Layman's terms any user accessing files will also wander around to access similar files within a specific time period.

For example, Let P1, P2.... Pn represent the file paths after pre-processing steps. If a user accesses the file paths P1, P2...P3, P4 in the time window of 15 minutes then the session can be represented as S= P1, P2, P3, P4 also in the same order of navigation.

To study this behavior better we simply concatenate the time-window parameter. In our project we fixed the time-window parameter to take the values of 15 minutes, 30 minutes and 45 minutes. Accordingly, we were able to identify 998, 933 and 909 sessions in the window mentioned above. To link two sessions and understand the commonality between them we next had to come up with a similarity/comparison parameter. It is based on the concept of subsets i.e., pairs of clusters are merged together if a session is a subset of another session. Subset is a set which contains the same items in the same order of occurrence.

This method is taken from the paper [15] but instead of finding the proper subset we simply used subset concept to group different sessions. For example, consider the six sessions along with their navigation patterns

The similarity matrix is given by:

Algorithm 1: Similarity Between Sessions(SBS)

Input: Two Users session $S_i - pat_i$ and $S_j - pat_j$
Output: Similarity Between S_i, S_j

```
if S_is pattern is a subset of S_js pattern then
    return p_i.length > p_j.length ? return p_j.length :
     return p_i.length
else
    return 0
```

Example:

S1: P_1 P_2 P_3 P_4 P_5
S2: P_1 P_2 P_3 P_5 P_6
S3: P_2 P_3 P_4 P_5
S4: P_3 P_4
S5: P_3 P_5 P_6
S6: P_3 P_2

A. *Hierarchical Agglomerative Clustering(HAC)*

It is an algorithm used for clustering similar groups together. It is a bottom up approach where initially each item is considered as a single cluster and as and when it moves upwards it merges the pairs of clusters in one large cluster based on a comparison parameter and finally all the clusters are merged into one large cluster. In general, the merges and splits are done in a greedy manner and the complexity of this algorithm is O(n2). It can be of two types Single Linkage and Complete Linkage. In Single Link Linkage, the similarity of clusters is based on most similar members or it combine two clusters that contain the closest pair of elements and not belonging to the same cluster. In Complete Linkage, the similarity of clusters is based on the most dissimilar members or it combines two clusters when the distance between them are the farthest from each other. We have considered the Complete Linkage method in our work. To understand the algorithm let us take an example, consider the distance matrix below:

The example is taken from [23]. Since the distance between 3 and 5 is the least they get clustered into 35 cluster. Next step is to obtain a new matrix by removing the entries for 3 and 5 and replacing the entries by 35. We should find the distance between each element and 35 cluster. The distance should be the maximum of the distance between this item and 3 and this item and 5. d (1,3) =3 and d (1,5) =11 hence, d (1,35) =11.Next, the items (2 and 4) with are the smallest distance gets clustered next. This process continues till all the elements gets clustered.

Similarly, in our work, we have grouped the clusters based on the similarity matrix that was established earlier using complete linkage. The departmental web log data received was very large to complete the similarity matrix as it compares(subset) every session to all the sessions and return the similarity measure i.e., the two for loops in our code would give the complexity of O(n2) and since the original data contained 1 million lines, the code ran forever. Thus, we considered only the first 10,000 and 50,000 lines of our data

	S1	S2	S3	S4	S5	S6
S1	-	0	4	2	0	0
S2	0	-	0	0	3	0
S3	3	0	-	2	0	0
S4	2	0	2	-	0	0
S5	0	3	0	0	-	0
S6	0	0	0	0	0	-

	1	2	3	4	5
1	0				
2	9	0			
3	3	7	0		
4	6	5	9	0	
5	11	10	2	8	0

for analysis. HAC is implemented by using Agglomerative Clustering package from sklearn. We had to set the parameter affinity to precomputed and the linkage to complete and the algorithm was fit on similarity matrix calculated above.

Algorithm 2: HAC

Input: A set of session and it's navigation pattern = $\{S_i - pat_i, S_j - pat_j \ldots S_n - pat_n\}$

Output: Set of clusters $c = \{c1, c2, ...ck\}$

do
- **foreach** *session* s_i *from the set* **do**
  - **foreach** *session* s_j *from the set* **do**
    - calculate $d_{(i,j)} \leftarrow SBS_{(si,sj)}$
    - **if** *if all entries in* $d_{(i,j)}$ *are zero* **then**
      - break
  - **for** $i \in n$ **do**
    - **for** $j \in n$ **do**
      - $merge(s_i, s_j)$ based on complete linkage method
      - replace the values in the matrix and find the next cluster to merge

while *True*

The results of HAC was realized by plotting the Dendrogram. The dendrogram was given below represents the clusters in the time window of 15 minutes.

B. K-Means

It is an unsupervised learning algorithm used to group data which are similar to each other. Since in our project data we did not have any labels, the data becomes unlabeled data and the technique by which we analyze the data is called the Unsupervised Learning. The main goal is to find the groups using the method of centroids. Each cluster is associated with a centroid. Next each point gets assigned to the cluster with the closest centroid. Number of clusters is always specified, and,

	35	1	2	4
35	0			
1	11	0		
2	10	9	0	
4	9	6	5	0

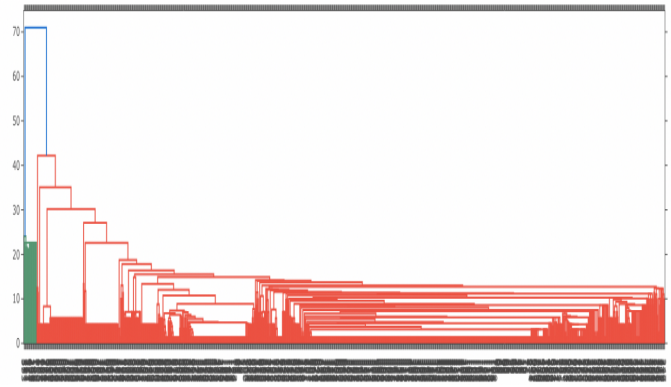


Figure 8: Dendrogram results of HAC

in our experimentation, we considered the values of k=5 and k=10.Initial centroids are usually chosen randomly. Centroid is nothing but the mean of the points in a given cluster. The concept of closeness is used measured by Euclidean distance between two points where as we could also use cosine similarity, correlation etc.

Algorithm 3: K-MEANS

Input: A set of files grouped based on their sub directory and the number of clusters(k)

Output: Set of clusters $c = \{c1, c2, ...ck\}$

do
- K clusters by assigning all points to the closest Centroids
- Recompute the Centroid of each cluster

while *The Centroid don't change*

Complexity of K-Means is usually given by O(n*k*l*d) where n represents the number of points, k represents the number of clusters, l represents number of iterations and d represents the number of attributes. In our work we implemented the K-Means algorithm by using the K-Means from the sklearn package by setting the value of k. Next, we implemented K-Means by selecting the directory-based approach as our feature. Directory based as mentioned is nothing but the grouping of all the files based on the inner most directory. Since all values should be in numerical format for K-Means algorithm we had to convert the categorical data in to numbers. Following we generated the dummies on subfolders and then normalized the data using the MinMaxScaler. In order to plot the K-Means graph we had to reduce the dimensions of our data hence we simply used the t-SNE algorithm. It is a

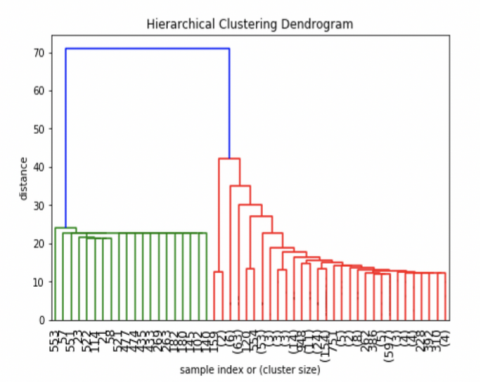


Figure 9: Last 50 merged clusters

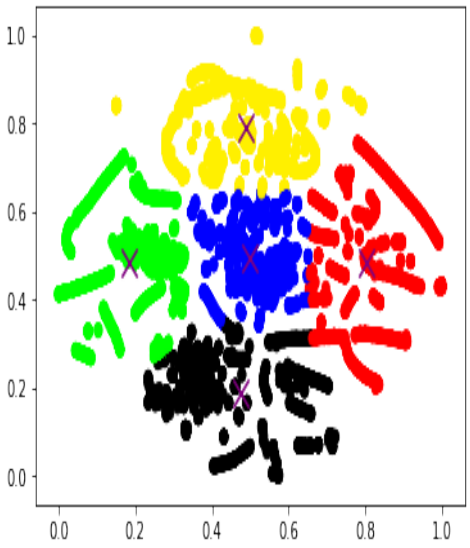


Figure 10: K-Means algorithm using Directory based(k=5)

nonlinear dimension reduction technique for embedding higher dimensional data into a low dimensional space. After reducing the data into two dimensions, we normalize it for the second time using the MinMaxScaler and then give the scaled data to the K-Means algorithm. Initially we considered K-Means even for the sessions process generated by the method mentioned above. But since the data was too large reducing the higher dimensional data to two dimensions was impossible even when we considered only the first 10,000 lines of the original data. Hence, we decided to try on by considering the directories. We plotted the results of K-Means for both values 5 and 10 and is given by:

VII. Experiment

In our experiment, We have implemented our cluster transaction processing and frequent itemset mining in python. Our main target of producing different types of clusters is to create frequent mining based concatenation of meta-data so that reclaiming data from disk become faster. So, We have to minimize the cache effect to accurately test our cluster quality. We have built a 64bit Ubuntu virtual environment with 4GB

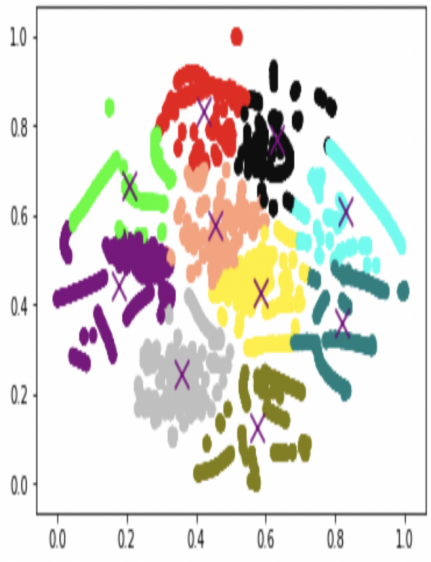


Figure 11: K-Means algorithm using Directory based(k=10)

of memory and 3 CPU. We clone the same environment into another machine to speed up our experimental measurement. We have used one machine for testing the time window, time window, and directory, and directory-based clustering quality testing. Another machine was used for testing the HAC based clustering.

We have evaluated our clusters with 2 different logs, (i) Web Trace Log from a personal computer, we call it **UserLog**. The trace size is 60.5MB and it contained over 0.678M file-system-related calls. After data-cleaning we got 29K file-system calls to generate our cluster (ii) Department HTTP Log, from our departmental web server (01/01/2015 03/18/2015). The department traces log size 203.8MB and it contains 10M file-system-related calls. We call it **server Log**. Even after data-cleaning server Log is still very large to process, So we took the 30K file-system calls to log from the middle of the Server Log where data are very much denser.

We have used 2 scripts for examining our cluster quality. First one, take the original trace log and our clusters as input to create a dummy file system and hard links metadata among clustered files. The second script used to replay the whole trace log on the dummy file system we have just created and use the hard-links to access the clustered files. It also measures the total number of bytes read, total elapsed time and returns the average latency, which is the measurement of how good or bad our clusters are. We have conducted multi-threaded, zero-think-time trace replays on disk storage. We have used our virtual environment for our replay experiment.

Each of the cluster hard-link, dummy File System generation and replay experiment took us around 1hr for User Log. In case of Server Log, this process took around 9hrs, hence, we could not generate more results on server Log

Table 1: Time Window based Cluster Quality

Log Type	Algorithm	Time Window	minSupport	Avg Cluster Size	Cluster construction time(ms)	Total Bytes Read	Total Elapsed Time(ms)	Average Latency(ms)	Baseline Information
User Log	FPGrowth	1	10	3.35	15.40	125862291	7761452	1,247.62	Total bytes read - 246864585 Total elapsed time - 29768880 Average latency - 1050.901260 Average bytes read - 8714.815724
			50	2.50	9.12	125862291	7761452	1,247.62	
			100	2.50	8.80	148784107	13203627	1,167.02	
		4	10	3.86	4.98	134813135	9122696	1,353.72	
			50	3.00	3.78	148648328	15007399	1,326.91	
			100	3.00	1.96	163320051	16729644	1,120.91	
		7	10	3.25	5.36	138689594	13157027	1,661.03	
			50	2.67	3.22	148864865	8745707	**772.38**	
			100	0.00	NA	0	NA	NA	
	Apriori	1	0.05	8.00	7.79	135,054,107	9071621	1,345.94	
			0.15	4.00	4.17	148,403,126	14521699	1,283.18	
		4	0.05	12.00	57.18	126,928,690	9580987	1,531.98	
			0.15	4.00	3.69	148,788,574	9067037	**800.90**	
		7	0.1	6.00	3.91	139,458,929	7755313	**1,001.72**	
			0.15	4.00	2.38	149,165,230	8371852	**739.24**	
Server Log	FPGrowth	1	100	2.02	1.27	652190704	7999525	**4,489.07**	Total bytes read - 4049465258 Total elapsed time - 59180066 Average latency - 4971.026123 Average bytes read - 340148.278706
		4	100	3.75	1.32	1231257297	16759332	**3,798.58**	
	Apriori	4	0.025	12.25	0.15	858249700	14289859	**4,114.56**	

A. *Performance Evaluation*

1) ***Time Window based Cluster::*** There was not much flexibility in our cluster processing, due to the options limitations in our Logs(User/Server). All the itemset mining algorithm works on some kinds of transactions. So Initially, We thought to create transactions based on file access time(seconds) present in Logs. We established "$TimeWindow$" like a sliding feature to extract different transactions from Log based on different user's input for this time window length. Then we fed these transactions into two algorithms we have explained in section 5 and generate clusters for different $minSupport$. In Table 1, presents the cluster quality and performance evaluation data of our time-window based approach. Our main target was to create small size high-quality cluster because hypothetically one infrequent file in CFFS could totally hamper/degrade the intended results. We observe that our cluster average size is very small. And after manual checking of those files in each cluster, we make sure that those files are actually frequent.

There is an inverse proportional relation between $minSupport$ and cluster construction time. Algorithms can do early pruning many candidates when the threshold is high, So it takes less time constructing cluster.

We also observe that total byte reads are almost half of the baseline and total elapsed time is also very less than the baseline. Yet, our average latency is higher due to the hard-link read fails using the second script. We will explain later more about it.

Note that in the baseline cluster, all files are un-clustered and tested same way as we tested our cluster using those 2 scripts and baseline information is always the same for each trace Log.

2) ***Time and Directory based Cluster::*** After time window based clustering, we were curious to incorporate the directory along with it. In this approach, first, we pre-filter all file accessed within the same time then apply the directory based grouping on the filtered result. More specifically, we took the pre-filtered list and group by with the directories. Note, If any directory contains only one file we discard them. Because of time constraint, we only tested this approach on FPGrowth with User Log. We have tested with different $minSupprot$ and witnessed that cluster processing time is higher than the time-window based clusters. We comprehend that average latency is better with this approach compared with only time window based approach. See the evaluation result of this approach in Table 2.

Table 2: Time & Directory based Cluster Quality (FPGrowth, timeWindow=1, UserLog)

minSupport	Avg Cluster Size	Cluster construction time(ms)	Total Bytes Read	Total Elapsed Time(ms)	Average Latency(ms)
20	2.54	6.73	223506490	36707157	1,100.69
50	2.55	6.38	249212227	24640016	1,060.03
100	2.54	6.51	241448727	12609848	**1,027.35**

3) ***Only Directory based Cluster::*** We have also tried only directory based clustering, In this approach, we did not engage any itemset mining algorithm. We wanted to try directory based consolidation with different file count threshold in each directory. In directory-based approach, average cluster size if larger than our others approach presented here. Due to the large cluster size, total bytes accessed is almost close to the

baseline(See table 1 for baseline) and it took more time to construct the cluster.

We also considered the directory based approach and performed K-Means on it.The baseline results is calculated.We considered two values for k.The latency value decreased for k=5 with respect to the baseline where as it increased for k=10.Hence, it is difficult to conclude the quality. The results are formulated in table 4.

Table 3: Only Directory based Cluster Quality(UserLog)

File Count Threshold	Avg Cluster Size	Cluster construction time(ms)	Total Bytes Read	Total Elapsed Time(ms)	Average Latency(ms)
1350	1,830.50	566.62	198809576	51813800	1,180.66
1320	1,334.97	2,395.17	217945792	82989900	1,890.63
1200	1,305.93	5,805.40	236863842	88303010	2,011.26

Table 4: K-Means based on directory

K value	Size of the data	Results	Baseline Results
k=10	10000	Total bytes read - 5812357 Total elapsed time - 2009 Average latency - 154.538462 Average bytes read - 447104.384615	Total bytes read - 1289434016 Total elapsed time - 308219 Average latency - 63.615893 Average bytes read - 266137.0518
k=5		Total bytes read - 1015261 Total elapsed time - 377 Average latency - 34.272727 Average bytes read - 92296.454545	

*4) **Session based:**:* As explained above, sessions were constructed by considering three values for time-window, 15,30 and 45 minutes.These sessions clustered using HAC algorithm and were given to the two scripts mentioned to test the quality.The baseline result were also established.When we compare the values with the baseline result, we see that the average latency is lesser than the average latency for baseline results.But for 45 minutes, we see a significant drop in the average latency parameter.This could be due to the comparatively less number of sessions and also could be due to caching as the experiments were carried out on one system. The results are given by table 5.

Q. Why the Average latency is high? To answer this question, we need to understand how the average latency is calculated by the second script. It calculates the total elapsed time and the number of reads during replaying the original log on the dummy file system with our cluster generated hard-links. Average latency is the result generated by this two data using below equation:

$$\text{Average latency} = \frac{\text{total elapsed time}}{\text{number of reads}} \tag{3}$$

As we can see our Table 1, Table 2 and Table 3, for all test cases(with user and server log), Our total elapsed time is less than the baseline. So something must be wrong with the number of reads. We have investigated in backward fashion and found that during replay, the second script got lots of $ERRNO$ (in some cases, half of the total baseline read). The Error message was ".. File size is too small to be read". These errors depicts that read operation failed for certain files, consequently, we got less read count, which result in higher average latency. Then we dive into it to expose the reason, why it fails to read files while exact file(with mentioned size) is present in our dummy file system. We discovered that during hard-link creation using the first script, there are many hard-link creation failures. Due to this hard-link failure, the second script could not find the meta-data of our clusters, so it settles with an error.

Table 5: HAC based on sessions (Server Log)

Session Window	Total number of sessions	Size of the data	Results	Baseline Results
15 minutes	998	10000	Total bytes read - 50482225 Total elapsed time - 6278 Average latency - 25.520325 Average bytes read - 205212.296748	Total bytes read - 1289434016 Total elapsed time - 308219 Average latency - 63.615893 Average bytes read - 266137.0518
30 minutes	933		Total bytes read - 768385 Total elapsed time - 395 Average latency - 26.333333 Average bytes read - 51225.666667	
45 minutes	909		Total bytes read - 221803 Total elapsed time - 74 Average latency - 8.222222 Average bytes read - 24644.777778	

B. Challenges faced

The server log contains 1 million lines of system call logs of file access and accessing such huge file in a local machine was a challenge. Often our systems became unresponsive to run some simple preprocessing task over that log. Hence we realized the datasets by considering only a part of server log. We tried splitting the server log based on date ranges and we could get around 10 dataframes. However, we could not consider those single dataframes because, while running the replay script we had to pass the filepaths from the dataframe and the server log containing exactly those filepaths. Splitting dataframe using date categorization leaves us with very sparse data. While some date contains only a few system calls, others have more than 100,000 system calls in a single day. We ran our code on these dataframes but still our machines would freeze with this huge load. In that scenario, we split the server log into several dataframes maintaining the order of system calls where each dataframe contains about 30000 lines of log. We considered two such dataframes for our experiment and proceeded in creating transactions.

Though we did not have enough features to explore different ways to cluster, yet we tried to form transaction based on "$userID$" from Server Log. But the results we observed was not good enough to do more experiment on it. Both of our algorithms (Apriori/ FPGrowth) returned some frequent itemsets those were not actually frequent. We figured out the reason behind it. Since each of the server Log filepath already contains the "$userID$" as the root node, transaction processing using this field($userID$) will not give us our intended results.

We faced the difficulty of recognizing and separating the files and directories within logs. We talked to Bobby to understand about the method that he had used. He only considered the files in the innermost directory. We found this method trivial since it's a very common practice to keep the files and

directories under another directory. So we came up with our own idea- to do DOT(".") processing, we considered those log filepath as files if they contain DOT in their filepath. Initially, we faced difficulty to implement that in python pandas data frame. Our implementation was taking too much time. Later, we found a faster approach to complete the DOT processing.

For clustering methods both for Hierarchical Agglomerative clustering as well as K-Means using the whole data was extremely difficult. In HAC, we were trying to generate the similarity matrix which takes n^2 computations. Since the original data contained 1 million lines, similarity matrix generation proved to be a futile effort. Similarly even for K-Means since we were normalizing the data and reducing the higher dimensions to lower dimensions, the size of our data posed challenges. Hence we decided to only use the first 10,000 lines. We initially had also considered 50000 lines for generating clusters but it was impossible to derive any result. Also since we know that the web log data maintains an order in its entry we simply cannot choose the data randomly. Hence we had to use the data from its starting point.

We had also read papers which clustered files based on processID parameter as it is unique to each process. But unfortunately since the data generated by Bobby did not contain processID, we could not use them. The most important challenge related to using sklearn algorithms is that fact that the functions do not allow the users to write either their own metrics for computation(like euclidean distance) or the user-defined functions which help to improve the algorithms. The only way to use a new technique altogether is by writing the new algorithm from scratch.

The most tiring and challenging task we have faced was with replay scripts during our quality testing of clusters. Our code is faster and we could easily generate a lot of clusters using different parameters and logs(Server / User). But each of the cluster testing on user log took us around 1 hour (half hour for dummy file system and hard link generation and half hour for replay). It was kind of spoon feeding, We had to be alert to do the next step with an interval of 30 minutes. For server log, the same experiment took around 9 hours for each case. That's why we could not generate more results on server log.

VIII. Conclusion and Future work

We have presented the work on composite-file file system along with their design, implementation, and experimentation by exploring different strategies on the data. The CFFS system is used to identify the files accessed together and to consolidate their metadata. It could significantly reduce the number of I/O's for both read-dominant and write-dominant workload. Our work extended the work on the composite-file file system and included new techniques with different algorithms used for analyzing the files such as FP-Growth, K-Means, Hierarchical Clustering, and Apriori. Even though we recorded ambiguous results, we can easily conclude that the many-to-one mapping is beneficial and can lead to better optimization techniques.

Because of the time constraint we were not able to test our experiment completely since the scripts which were used ran for long hours. Also, the scripts we used for testing failed to read many files which resulted in low latency for our clusters. Our goal will be to write our own script to seek correct latency and compare it with baseline clusters. Also, we will try to come up with a solution to run our scripts faster. Our work considered the subset of the data for infusing different techniques. However, we could not integrate our methods on the entire dataset and test the results. Given proper resources, we would like to apply our implementations on the entire dataset to generate better clusters. Our current implementations could help us understand the working of the entire system which will help us to establish certain parameters like the performance parameter over the existing work etc. To summarize, as a future work we can do the following:

- Write our own scripts to test our cluster quality with accurate latency.
- Try to make the script run faster as this was a huge overhead.
- We could also study the performance of the system on SSDs and further understand the performance gap between HDDs and SSDs.

References

[1] S. Zhang, H. Catanese, and A.-I. A. Wang, "The composite-file file system: Decoupling the one-to-one mapping of files and metadata for better performance." in *FAST*, 2016, pp. 15–22.

[2] N. K. Edel, D. Tuteja, E. L. Miller, and S. A. Brandt, "Mramfs: A compressing file system for non-volatile ram," in *Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, 2004.(MASCOTS 2004). Proceedings. The IEEE Computer Society's 12th Annual International Symposium on*. IEEE, 2004, pp. 596–603.

[3] S. Jiang, X. Ding, Y. Xu, and K. Davis, "A prefetching scheme exploiting both data layout and access history on disk," *ACM Transactions on Storage (TOS)*, vol. 9, no. 3, p. 10, 2013.

[4] X. Ding, S. Jiang, F. Chen, K. Davis, and X. Zhang, "Diskseen: Exploiting disk layout and access history to enhance i/o prefetch." in *USENIX Annual Technical Conference*, vol. 7, 2007, pp. 261–274.

[5] Z. Li, Z. Chen, S. M. Srinivasan, and Y. Zhou, "C-miner: Mining block correlations in storage systems." in *FAST*, vol. 4, 2004, pp. 173–186.

[6] T. M. Kroeger and D. D. Long, "Design and implementation of a predictive file prefetching algorithm." in *USENIX Annual Technical Conference, General Track*, 2001, pp. 105–118.

[7] K. S. U. N. M. A. G. Shamila Nasreen, Muhammad Awais Azamb, "Frequent pattern mining algorithms for finding associated frequent patterns for data streams: A survey," in *EUSPN*, 2014.

[8] I. V. Renáta Iváncsy, "Frequent pattern mining in web log data," in *Acta Polytechnica Hungarica*, 2006.

[9] X. J. Z. Liping Sun, "Efficient frequent pattern mining on web log data," in *Asia-Pacific Web Conference*, 2004.

[10] A. S. Chen Luo, "Ssh (sketch, shingle, hash) for indexing massive-scale time series," in *arXiv*, 2016.

[11] A. M. R. Heidar Mamosian and M. A. Dezfouli, "A new clustering approach based on page's path similarity for navigation patterns mining," in *arXiv*, 2010.

[12] M. H. A. Elhiber and A. Abraham*, "Access patterns in web log data: A review," *Journal of Network and Innovative Computing*, vol. 1, pp. 348–355, 2013.

[13] M. Agosti and G. M. D. Nunzio, "Web log mining:a study of user sessions," 2007, pp. 1–5.

[14] M. P. Priyanka Patel, "A review on user session identification through web server log," in *(IJCSIT) International Journal of Computer Science and Information Technologies*, vol. 5, 2014, pp. 1–3.

[15] S. D. Anupama D.S, "Clustering of web user sessions to maintain occurrence of sequence in navigation pattern," 2015, pp. 558–564.

[16] A. S. T. V.Chitraa, "Web log data analysis by enhanced fuzzy c means clustering," in *(IJCSIT) International Journal of Computer Science and Information Technologies*, vol. 4, 2014, pp. 1–15.
[17] A. K. Ruchika R. Patil, "Bisecting k-means for clustering web log data," in *International Journal of Computer Applications*, vol. 116, 2015, pp. 36–41.
[18] P. S. R. G. Poornalatha, "Web user session clustering using modified k-means algorithm," 2011, pp. 243–252.
[19] R. Agrawal, R. Srikant *et al.*, "Fast algorithms for mining association rules," in *Proc. 20th int. conf. very large data bases, VLDB*, vol. 1215, 1994, pp. 487–499.
[20] F. A. El-Mouadib, "The performance of the apriori-dhp algorithm with some alternative measures."
[21] T. id Items, "Association analysis: Basic concepts and algorithms."
[22] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," in *ACM sigmod record*, vol. 29, no. 2. ACM, 2000, pp. 1–12.
[23] "https://newonlinecourses.science.psu.edu/stat555/node/86/."