DSCI 551 Project, Fall 2022

100 points

The theme of this semester's project is developing and utilizing an **emulation**-based system for distributed file storage and parallel computation. The project consists of three major tasks: (1) building an emulated distributed file system (EDFS); (2) implementing partition-based map and reduce (PMR) on data stored on EDFS for search/analytics function in task 3; (3) creating an app that uses the implemented PMR methods.

1. [20 points] Building an emulated distributed file system (EDFS)
   - EDFS should support the following commands, similar to that in HDFS:
     - mkdir: create a directory in file system, e.g., mkdir /user/john
     - ls: listing content of a given directory, e.g., ls /user
     - cat: display content of a file, e.g., cat /user/john/hello.txt
     - rm: remove a file from the file system, e.g., rm /user/john/hello.txt
     - put: uploading a file to file system, e.g., put(cars.csv, /user/john, k = # partitions) will upload a file cars.csv to the directory /user/john in EDFS. But note that the file should be stored in k partitions, and the file system should remember where the partitions are stored. You should design a method to partition the data. You may also have the user indicate the method, e.g., hashing on certain car attribute, in the put method.
     - getPartitionLocations(file): this method will return the locations of partitions of the file.
     - readPartition(file, partition#): this method will return the content of partition # of the specified file. The portioned data will be needed in the second task for parallel processing.

     Note that EDFS should store the metadata about the file system (similar to that in name node of HDFS, but much simplified). Metadata include file system structure, attributes of files, and location of partitions storing the content of the files. You can limit the type of files stored in the file system to certain format, e.g., csv or JSON.

   - Sample implementations: you can think of the above commands of EDFS as API calls. It is up to you to implement these calls. Note that your implementation does not have to create actual files on some file system (so emulation-based). Instead, you can use SQL or NoSQL database to implement these calls (and thus support the commands).
     - Firebase-based emulation: you can use an JSON object to emulate the directory structure of the file system.

```
e.g., / -> user, home, etc
"root": {
        "user": {
                "john": {
                        "cars.json":  {"p1": <location>, "p2": <location>,
                        "boats.json": …
                        }
                }
```

}
Where <location> may refer to URL of partition of a file, e.g., the first partition of cars.json is stored as: https://dsci551a-1234.firebaseio.com/data/cars1.json". Note that the partition of cars.json may be stored in a different database than one used to store the directory structure of file system. You can think of the database storing the actual data of file as data node, while the database storing the metadata as name node.

- MySQL-based emulation: Here you can store file system information in tables. For example, one table for storing metadata of files, including id of files (similar to inumbers), file names, etc. ; another table for the directory structure of the file system in a table, e.g., with attribute (parent, child), e.g., rows in the table may be: (id for 'root', id for 'user'), (id for 'user', id for 'john'), (id for 'john', id for cars.csv); table for storing the partition information about a file, e.g., (id for cars.csv, partition 1), (id for cars.csv, partition 2), where partition 1 is a location which may refers to another table in the database, if you partition cars.csv horizontally and store them in separate tables (one for each partition). For example, all cars with ids < 100 go to table 1, 100 <= id < 200 go to table 2, etc. The partition may also base on the hashing of some car attribute, e.g., all cars with odd car ID go to partition 1, etc.

  You may find the partitioning method in MySQL useful:
  https://dev.mysql.com/doc/mysql-partitioning-excerpt/8.0/en/partitioning-types.html

- MongoDB-based emulation: This can be similar to that using Firebase, but instead of database, you may store directory structure in a MongoDB collection; and may store different partitions of the same file (e.g., cars.json) in different MongoDB collections (which may be in different MongoDB databases).

2. [20 points] Implementing partition-based map and reduce (PMR) on data stored on EDFS, for the search or analytics functions in your application developed in the next task.
   - Partition-based map, also called mapPartition, is similar to mapPartitions() in Spark (https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.RDD.mapPartitions.html.)
     mapPartition(p) takes a partition p as the input and transforms (or reduces) data in p.
   - Reduce function then takes the output from each partition produced by mapPartition, combine/reduce them into the final output. Note that combine function might not be necessary, depending on the function you are implementing.
   - Examples:
     - Consider again the cars.csv file you have seen in homework 1. Suppose your application needs to search IDs of cars by price range:
       Select ID
       From Cars
       Where 15000 <= price <= 16000

And suppose Cars.csv is stored in two partitions: p1 = cars1.csv and p2 = cars2.csv. Then you can implement mapPartititon(p) that takes partition p, for each car in the partition, check if its price is in the range, and output its ID if yes.

- Suppose your analytics function is:
  Select carBody, count(*)
  From Cars
  Where fueltype = 'gas'
  Group by carBody

  In this case, mapPartition(p) may takes cars in partition p, output the number of gas cars for each different carBody (sedan, convertible, etc.)
  Reduce function then combines the local counts in each partition for each type of car body, and sums them up to produce final count.

3. [30 points] Creating an app for searching and analyzing the data stored in EDFS using the functions implemented using PMR.
   - Your app should have an interface to support the EDFS commands as described in task 1.
   - Your application should also have a function that allows users to interactively **navigate** through the EDFS, exploring its directory structure and files. Note that the UI for the navigation should be intuitive, and you should not expect users to navigate through the structure or content of database that is used to emulate the EDFS.
   - For each search and analytics function, you should implement an **explanation** facility that explains to the users how the search and analytics results were obtained. For example, show the input and output of mapPartition (for each partition) and reduce functions.

4. Other deliverables [30 points]
   - Reports [15 points]: proposal, midterm, and final report, 5 points each.
     i. Your proposal should describe in detail your plan on how to implement tasks 1, 2, and 3; your team members, backgrounds, and responsibilities; and timelines.
     ii. Your midterm report should show satisfactory progress based on your timelines.
     iii. Final report should be a comprehensive report, detailing your project design and implementations. It should also include a Google drive link to your codes.
     iv. Final report should have a separate section discussing your learning experience.
   - Peer evaluation [10 points]
     i. A google Form will be used to obtain your ratings on peer projects, based on how well the projects have implemented the required functions.
     ii. It will be conducted in the last week of the semester.
   - In-class presentation and recorded 20-min video [5 points]
     i. The presentation will be held in the last week, online.
     ii. The 20-min video (and final report) are due in the first day of last week of semester.
   - Bonus points [2 points] (maximum point possible of project remains to be 100 points):

  i. beginning of semester background survey
  ii. end of semester survey

5. Additional requirements
   - You are allowed to form a group of up to 5 people. However, if your group has more than 3 people, we require you to also set up a Spark cluster to implement your search and/or analytics functions in the actual cluster with multiple worker nodes, each residing on a different machine. See this for more information:
     https://spark.apache.org/docs/latest/cluster-overview.html

   - Detailed requirements vary depending on your group size:

Group size	Acceptable UI	Minimal number of data sets	Number of EDFS implementations	Search	Analytics	Spark cluster
1	Command line, Jupyter notebook	1	1	X		
2	Jupyter notebook	2	1	X	X	
3	Web browser-based	3	2	X	X	
4	Web browser-based	4	3	X	X	X
5	Web browser-based	5	3	X	X	X

Note x means required.
If you are required to implement more than one EDFS emulation (e.g., one using Firebase, the other MySQL), your final report should have a separate section discussing the pros and cons of different method, and your development experience. If your group is required to implement Spark cluster, please address the differences between and learning experience of developing emulation-based solution and using actual Spark cluster.

6. Deadlines
   - Proposal: 9/19, Monday
   - Midterm report: 10/31, Monday
   - Final report, 20-min project video: 11/28, Monday
   - Peer evaluations: 12/2, Friday