|  |  |  |
| --- | --- | --- |
| Actor | Use Case | Description |
| User | Analyse Data | Analyse data is the first main feature that should be provided. A user should choose as a first step which algorithm to run. Then as a second step, the user should configure algorithm parameters before running it. This use case must be preceded by an authentication. |
| Manage Data | Mange large scale of data is the second main feature that should be provided. User should be enabled to load data and explore its emplacement. This use case must be preceded by an authentication. |
| Visualize Results | Visualize results is also a main feature of the application. Result are sort of graphs and charts. Therefore, user should configure parameters (e.g. choosing axes) and then load chart. This use case must be preceded by an authentication. |
| Manage Account | The application must enable users to manage their own information in their accounts (e.g. password, user name ..). This use case must be preceded by an authentication. |
| Administrator | Manage list of users. | The Administrator has exclusively the right to confirm new users’ right to access to their accounts after verifying their identity. As being a critical use case, this operation must be preceded by authentication. |
| Use Application Statistics | Each operation processed must be logged in a way that enables the administrator to control application operation. This use case is very important in order to register traceability of the processed operations which make reporting issues easier and more efficient. This use case must be preceded by an authentication. |
| Client | Request Registration | As a first step to interact with our application, a user should fill a form on which he describes his identity. After this step, he waits for the administrator confirmation to access the application. |

When the user sits in front of his computer, open the application on his browser he receives, first, the authentication page. He types his login name and password. If it matches together (login + password), the user access to the application. Otherwise, he will redirected to the login page again. After getting the access the application, the user receives the main menu, from which he can choose either \textbf{Manage Data}, \textbf{Analyse Data} or \textbf{Visualize Data}.

If the user chooses the \textbf{Manage Data} option, he will we asked to choose if he wants to load new data or to explore existing data through a file manager.\\

If the user chooses the \textbf{Analyse Data} option, then he is asked which is the appropriate algorithm to run on selected data, configure the algorithm parameters and process it by the end.\\

If the user chooses the \textbf{Visualize Data} option, then he will be asked to fill the configuration form (select result to visualize, select axes …). This step will make loading charts possible.\\

Consult application statistics

Queries are the main way to make the application server and service server communicate. Queries are sent through TCP/IP socket. In order to avoid server overload, if a query is lost the receiver will not wait it infinitely. After exceeding a predefined timeout, the application layer will sent a query to disconnect from the service and then kill the attached service thread.

In Order to grantee a multi-user application, we have adopted a multi-thread solution. For every operation running in the service, we create a thread. As we are using a socket solution to send communication queries between the application server and the service server, there is a threat that a query send from one thread disturb another one. In order to avoid the risk of disturbing the application, every query sent or received should be preceded by a user id in order to identify every query to whom user’s thread belongs. If a thread receive a query that does not mind him, it will reject it and then inform the application server to send a query again. For every query sent, the service thread make the same job, until it receive the correct query or reaches a maximum number of iterations; if it overcome it, it will reject the whole operation and kill the thread in order to avoid server overload and sudden shutdown.

The delivered application consists on designing and implementing Client / Server application that should satisfy the specific functionalities listed below:

The system must offer to the user the possibility to load big data from different sources into HDFS.

The user is not coming from An IT background; all command lines should be hidden behind a Graphical User Interface.

The system allows user to apply on large volume of data several data mining algorithms such as:

\begin{itemize}

\item Principal components analysis

\item Linear Model

\item K-Means

\end{itemize}

The User should be able to visualize algorithm results in different interactive forms (Charts, Graphics, data visualizations, grids …)

The application should provide a users management feature in order to affect different roles for every user category.

The application should provide for the administrator user the ability to log every operation lunched by the client. This functionality enables reporting issues.

This application should offer the ability for every client to ask for a registration. After administrator permission, he would be able to access and process algorithm on his loaded data.

When the administrator open the application on his web browser, he receives first the login page. He types his login name and password; if they (login + password) are correct, he will have access to the application. Otherwise, his request will be rejected and he will redirected again to the login page again.\\After getting the access to the application, the administrator will main menu. The provided options are \textbf{Manipulate Data}, \textbf{Choose the Manage list of users} and \textbf{Choose the Consult Application Log option}.

If the user chooses the \textbf{Manage List of Users} option, he will be asked either to \textbf{Add new user}, \textbf{Delete existing User} or \textbf{Edit user features}.

As a first step, a client should fill the registration form, providing his personal information. Filling this form is considered as a request to get access to the application. After this step, the client will wait for a confirmation from the administrator; if he get it (the confirmation), he will be able to access after. Otherwise, he will be rejected to fill a request again or quit the system.\\

Getting the confirmation will give the client the ability to login the system, manipulate his own data as much as he wants and then leave by logging out.

In this chapter, we have presented the requirements and the needs of the system and its users. We have first identified the actors of our system and explicit the functional and non-functional requirements. Then, we have detailed these needs by specifying, in more details, the features expected through a detailed use case and activity diagram. Thus, we can begin the design phase, which will be the purpose of the next chapter.

The placement of replicas is critical to HDFS reliability and performance. Optimizing replica placement distinguishes HDFS from most other distributed file systems. The purpose of a \textbf{rack-aware} replica placement policy is to improve data reliability, availability, and network bandwidth utilization.\\

Large HDFS instances run on a cluster of computers that commonly spread across many racks. Communication between two nodes in different racks has to go through switches. In most cases, network bandwidth between machines in the same rack is greater than network bandwidth between machines in different racks.\\

The NameNode determines the rack id each DataNode belongs to. A simple policy is to place replicas on unique racks. This prevents losing data when an entire rack. This policy evenly distributes replicas in the cluster, which makes it easy to balance load on component failure. However, this policy increases the cost of writes because a write needs to transfer blocks to multiple racks.\\

The HDFS namespace is stored by the NameNode. The NameNode uses a transaction log called the EditLog to persistently record every change that occurs to file system metadata. For example, creating a new file in HDFS causes the NameNode to insert a record into the EditLog indicating this. Similarly, changing the replication factor of a file causes a new record to be inserted into the EditLog. The NameNode uses a file in its local host OS file system to store the EditLog. The entire file system namespace, including the mapping of blocks to files and file system properties, is stored in a file called the FsImage. The FsImage is stored as a file in the NameNode’s local file system too.

The NameNode keeps an image of the entire file system namespace and file Blockmap in memory. This key metadata item is designed to be compact, such that a NameNode with 4 GB of RAM is plenty to support a huge number of files and directories. When the NameNode starts up, it reads the FsImage and EditLog from disk, applies all the transactions from the EditLog to the in-memory representation of the FsImage, and flushes out this new version into a new FsImage on disk. It can then truncate the old EditLog because its transactions have been applied to the persistent FsImage. This process is called a checkpoint. In the current implementation, a checkpoint only occurs when the NameNode starts up. Work is in progress to support periodic checkpointing in the near future.

The DataNode stores HDFS data in files in its local file system. The DataNode has no knowledge about HDFS files. It stores each block of HDFS data in a separate file in its local file system. The DataNode does not create all files in the same directory. Instead, it uses a heuristic to determine the optimal number of files per directory and creates subdirectories appropriately. It is not optimal to create all local files in the same directory because the local file system might not be able to efficiently support a huge number of files in a single directory. When a DataNode starts up, it scans through its local file system, generates a list of all HDFS data blocks that correspond to each of these local files and sends this report to the NameNode: this is the Blockreport.

Ref: 2017/II3/ June 2017

Ministry of Higher Education  
for Scientific Research and Technology  
University of Manouba   
National School of Computer Sciences



FINAL GRADUATION PROJECT REPORT

A DISSERETATION SUBMITTED IN FULFILMENT OF THE REQUIREMENT FOR

COMPUTER SCIENCE ENGINEER DIPLOMA

**Big Data Management for Manufacturing Intelligence Application**

**Performed by:**

Mèhdi Ben Hamida

**Supervised by**:

Mr. Mohamed Anis Mlaouah

**Advisor:**

Mrs. Lobna Jribi