Design:

In this report we want to describe the design of each part of the application.

1. Insert():

The aim of this workflow is to upload files to the distributed dataset or cache. There are two types of upload in this application which use two different API (BlobStore API and Google Cloud Storage API). Google Cloud Storage API has 32 MB limitation on the size of each request. However, Blobstore does not have any limitation on the size of requests as the connection is directly to the google cloud storage.

- **Upload files using Google Cloud Storage API:**
  When the users press the "upload" button, the method of "upload Files" which is a java script code is called". This method uses the list of files which user wants to upload. First, it examines the size of each file, if it is less than 100KB it should insert to both distributed storage and cache. Otherwise the desired file should upload only in distributed dataset.

  - **Uploading in cache:**
    In this part in the file "index.jsp" one HTTP request is created. The URL of this request contains the name and the size of the files which should be sent. In server side, "doPost" method from "CacheServlet.java" receives the request. There is a record in Cache which the filename (key) of all records store there in order to know which files are in the Cache. This file contains the names of all files which each of them is separated from the other by "//" character. The reason which we choose this separator is that in Windows and Linux operating system it is not permitted to have a filename including "//". Therefore, the first step in to insert the name of the files in to the "cachedFiles". Indeed, keys of all records are the values of "cachedFiles" record. The second step is to insert the files into the Cache. The name of files is the key and their content is the value of each record.
  - **Uploading in distributed dataset:**
    In this part one HTTP Post Request is sent to the server. This request contains a URL which includes the name of file and bucket name. At the server side "doPost" method from "UploadDownloadServlet.java" file receive the request. In this method we should create a "GCSFilename" using the filename which is in the URL of request packet. Then, the files store to the URL of request packet. Content of the file is sent to the server through a bytestream over the network.

- **Upload Files using Blobestore API:**
  "Blobestore" API creates a ULR automatically in order to download files.

  - **Uploading in the Cache:**
    The process of inserting files in the cache is similar to the Google Cloud Storage API. We do not use the automatic URL which is created by Blobstore API.

- **Uploading in distributed dataset:**
  In "index.jsp" the URL is created by Blobestore API and the "doPost" method from "UploadLargeFileServlet.java" is called. In this method we get the name of files and their content from blobstore and pass them to the Google Cloud Storage API in order to store them into the bucket.

2. Check():

In this process, the time which users press the "check" button. The check method in "index.jsp" is called. This method has the responsibility to check the availability of file in both cache and distributed dataset and alert it to the user.

- **Check() in distributed dataset:**
  Google Cloud Storage API has a method which lists name of all files which is available in bucket. Then the method search in this list to find the desired file.
- **Check() in cache:**
  As it is described in insert() part. There is a file in cache which contains the name of all cached files. This method search in this file to find the desired file.

3. Find(key):

This method downloads file from cache or distributed data set. During this process we use lists of file which is created in "listing()" method (is explained it later). Users can download files from cache or distributed dataset.

- **Download from Cache:**
  The filename is sent to the responsible servlet. The servlet gets the contents of the file from the cache and send it back to the client using a byte array.
- **Download from distributed dataset:**
  Server provides a link for the client to download the file directly from the google cloud storage.

4. Remove(key):

The goal of this method is to remove file from cache and distributed dataset. When the user presses the "delete" button, an HTTP Get request is sent to the "doGet" method of servlets.

- **Remove from cache:**
  In this part, "doGet" from "CacheDeleteServlet.java" is called. In this method the name of selected file is removed from the reference file in cache ("cachedFiles"). With this process the pointer to the file is removed and it would be impossible for us to have access to the file.
- **Remove From distributed dataset**
  In this part, "doGet" from "DeleteServlet.java" is called. In order to remove file from distributed dataset, Google Cloud Storage API has a method that we need to use.

5. Listing()

The goal of this method is to list the available files in cache and distributed dataset.

- **Listing available files in cache:**
  In this part we used the "cachedFiles" which has the names of all stored record. Through this file we can easily list available files.

- **Listing available files in distributed dataset:**
  Google Cloud Storage has an API for listing the available files in distributed dataset. We use this method in this part.

Function of extra credit briefly:

Key[] = Listing(string)
Javascript function of Listing(string) is called on the click of a button 'Search Files" and a String is passed to the function entered by the user in the search text box.

Function body:

1. Get a list of files of a specified bucket by using an interface object gcsService
2. Iterate through the list and retrieve file key for each file to match with the string pattern provided by the user.
3. If a match is found store the key in an array and pass the array to a new window listing all the files with the matching user search.

Boolean = RemoveAll():
Javascript function of RemoveAll () is called on the click of a button 'Remove All Files"

Function body:

1. Get a list of files of a specified bucket by using an interface object gcsService
2. Iterate through the list and delete files if exist.

Double = StorageSizeMB()
Javascript function of StorageSizeMB() is called on the click of a button 'Files Storage Size"

Function body:

1. Get a list of files of a specified bucket by using an interface object gcsService
2. Iterate through the list and get the size of each file.
3. Calculate total size by adding all file sizes followed by assigned to a variable storing total size.

Double = StorageSizeElem()
avascript function of StorageSizeElem() is called on the click of a button 'count All Files"

Function body:

1. Get a list of files of a specified bucket by using an interface object gcsService
2. Iterate through the list and count total number of files in the bucket.

Boolean = FindInFile(key, string)
javascript function of FindInFile(key, string) is called on the click of a button 'FindInFile" and the file key along with the String entered by the user in the search text box is passed to the function

Function body:

1. If a string pattern match is found then user receives an alert of "Match found" else "No Match found"

Boolean = RemoveAllCache():
Javascript function of RemoveAllCache () is called on the click of a button 'Remove All Files (Cache)"

Function body:

1. Get a list of files stored in cache by using JSP
2. Iterate through the list and delete files if exist.

Double = cacheSizeMB()
Javascript function of cacheSizeMB() is called on the click of a button 'Files Storage Size (Cache)"

Function body:

1. Get a list of files stored in cache by using JSP
2. Iterate through the list and get the size of each file.
3. Calculate total size by adding all file sizes followed by assigned to a variable storing total size.

Double = cachsSizeElem()
avascript function of StorageSizeElem() is called on the click of a button 'count All Files"

Function body:

1. Get a list of files stored in cache by using JSP
2. Store them in an array an return the length of the array

Creating workload design:

Objective: This program is to create a dataset comprising of 411 files spanning 311MB of data with certain rules.

Design:

1. A “random number generator” function is defined which creates a random number.
2. This random number then points to a character in a string which is appended to a new random string. Following this method a random string of a defined size is generated.
3. As we are supposed to copy 100 bytes per line in a file therefore random strings of 100 bytes are generated by the “generateRandomString” function and copied into newly created files.
4. Number of bytes copied to files varies as per the file sizes and different numbers of files are generated as mentioned for different file sizes.
5. Also, the files are given alpha-numerical names of 10 characters long by calling “generateRandomString” and passing the required string length.

*Note: 2 bytes of line separator is considered while generating and copying 100 bytes per line in a file.