In this project, we designed and implemented an application to index and search large documents. The implementation is done in two ways: 1) Java-Based Implementation, and 2) Hadoop-based implementation.

**Java-Based Implementation**

This is a server-client based implementation where we have a master server that handles all the connections and request and clients issue an index or a search request. The main components of this system are described below:

**Master Server**

The master server two concurrent threads: 1) MasterServerThread that waits for connection from either a client or a worker node, and 2)Main thread which waits for the arrival of a request in the WorkQueue (this queue holds incoming index/search request).

If the request is an index request, the master server checks if the document has been indexed before, then it skips the indexing, else it proceeds with the indexing request.

It then creates a Job Coordinator thread to verify if the request type of the received request is equal to the previous request. This is done so that no two same requests are handled at the same time (e.g, search request is not handled as the time as the index request).

**Job Coordinator**

The job coordinator creates jobs and sends them to the registered workers and mappers. This class defines number of mapper and number of reducer. These number depends on the size of the file.

The Job Coordinator class sends job to all the mapper/reducer and wait for an acknowledgement that they received the job. This acknowledgement is to avoid deadlock between two workers. The workers are shuffled so that all the works are not passed to the same worker.

[add any communication details that is missed]

[will draw the server-client diagram tomorrow in the board and add it here]

**Indexing:**

To perform the indexing operation, when we receive an indexing request, we divide the document and send to multiple mappers to do a wordcount on the file. The division of the document depends on the length of the input file [add anything else?]. For each document, the wordcount object contains the term and total count. The document is preprocessed so all the punctuations and stopwords are removed from the document before sending it off to the mappers. The mappers go through the document and converts each word into a wordcount object. Once done, they save the result to AFS space. Reducers dedicated to a range of alphabets then go through the wordcount and do a merge of the terms (total count of each item). A documentIndexer class is created to check if an input file is already indexed or not based on the filepath, if the index is present we return the id of the file. [what else?]

**Search**: [explain the search based on the code structure]

**Hadoop-based implementation**

For the hadoop-based implementation, we implemented an index mapper and index reducer to perform index operation. We also implemented a query mapper and query indexer to do a search. We used StringTokenizer to token the input file and the reducer aggregated and displayed the result. For querying, the query element is passed through argument. If a match is found, the term with filename and its number of occurrence is printed out.

In order to perform search, first index has to be done on the input file. The output of this index is given as an input path to the search query.

**Metric Used:** We decided to use completion time of each task to be a metric to determine how well the index/query operations work. Before the starting of each task, we measured the time, and upon completion measured the time again. The difference indicated the total time to complete the task.

**Test of Java-Based Implementation:**

**Test of Hadoop-Based Implementation:**

We tested our implementation in Hadoop server (had6110.cs.pitt.edu).

For a test data with 7 lines taken from ‘war and peace.txt’ files, it too 19140 ms to index the file. When we searched for two terms ‘freedom Hello’ it returned results within 111740 ms.

[add data for bigger inputs]

[read me is on a separate doc, I will polish it and upload it in git tomorrow]

**Conclusion:**

Through this experiment, it can be seen that for large dataset, MapReduce and Hadoop gives better results. Amount of code that needs to be written is less as well. Socket implementation and managing server-client-worker communication was complicated.