**CS 425 MP2: Simple Distributed File System (SDFS)**

*Dayue Bai (dayueb2), Yitan Ze(yitanze2)*

## 1. Design

We implemented a fully distributed system without using a master node. Each node maintains three lists: (1) full file list: list of all files stored in the file system, (2) local file list: list of all files (in the system) stored locally, (3) membership list: all members' information in the group. Besides, each node stores all file's information: name, local path, last modified timestamp, replica set, status. To tolerate up to 3 failures at a time, we store each file in the system on 4 different nodes. We show how we design each functionality below.

**put <localfilename> <sdfsfilename>:** When node A inserts its local file into the system, it decides the file's replica nodes set (may include itself) by hashing the file name to a node number. The following 3 nodes in the clockwise order will also be in the replica set (map node to a virtual ring). Then, A asks all replicas whether the file is being written or read. If all replicas say "yes", A tells all the replicas which local file it wants to upload. When any replica R receives this message, R adds the file to its full & local file list and builds a TCP connection with A to retrieve the file. After the file is replicated successfully, R sends an ACK message to A. When A gets quorum ACKs (majority), A adds the file to its full & local file list, multicasts new node's insertion to other nodes, and tells users insertion succeeds. Other nodes receiving the multicast will also add the file to its full file list. For updating, it simply gets the replica set from the file list (no need to hash or add the file to the local file list, just update the full file list).

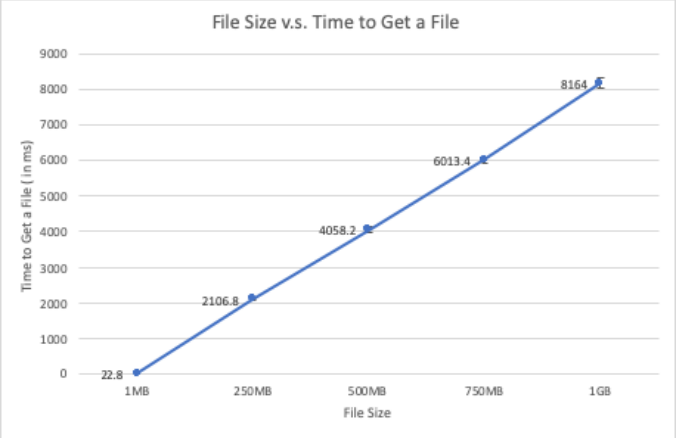
**get <sdfsfilename> <localfilename>**: We go to each replica node and check if the file is readable(not being written). If all replicas say the file is readable, we then send the read request to all replica nodes to get the file's latest timestamp and take only X number of responses. We only read from the node that contains the file having the latest timestamp. (In our implementation, X = quorum: majority)

**delete <sdfsfilename>**: We first check the status of all replica files and see if it's deletable, then delete the file in the current node and multicast the file deletion message to all other nodes.

**Re-replication process:** When a node fails, we re-replicate all files on that node. For each node, when it detects a failure, it sends the re-replication thread the failed node information and scans through its local file list. If it contains a file that's in the failed node and it is the smallest node in the replica set, then, this node is responsible for picking a random normal non-replica node as the new replica node and sending it the re-replicating request. When the new replica node gets the file, it multicasts the updated replica set info to all other nodes.

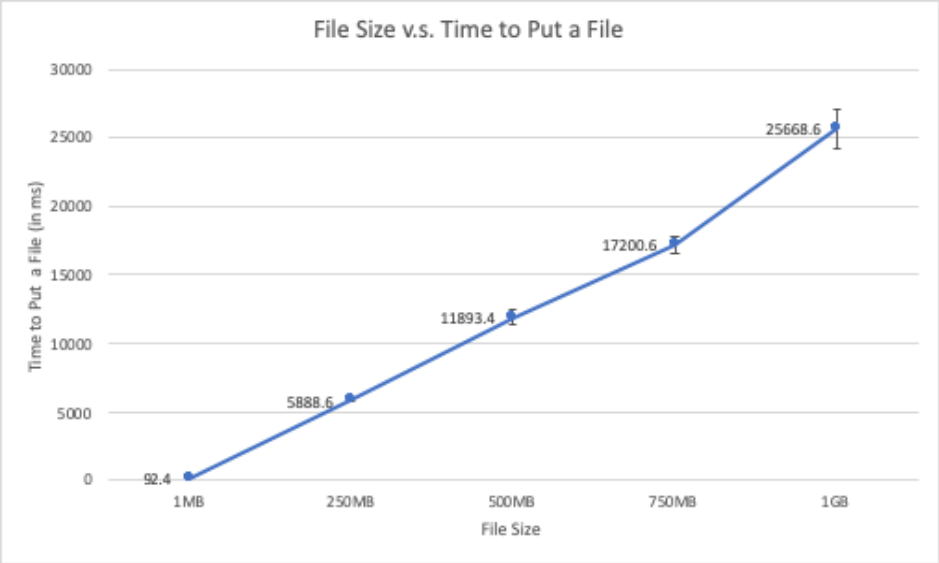
## 2. Measurements

**(i) Time to get a file vs. File size** (File size ranging from 1 MB to 1 GB)



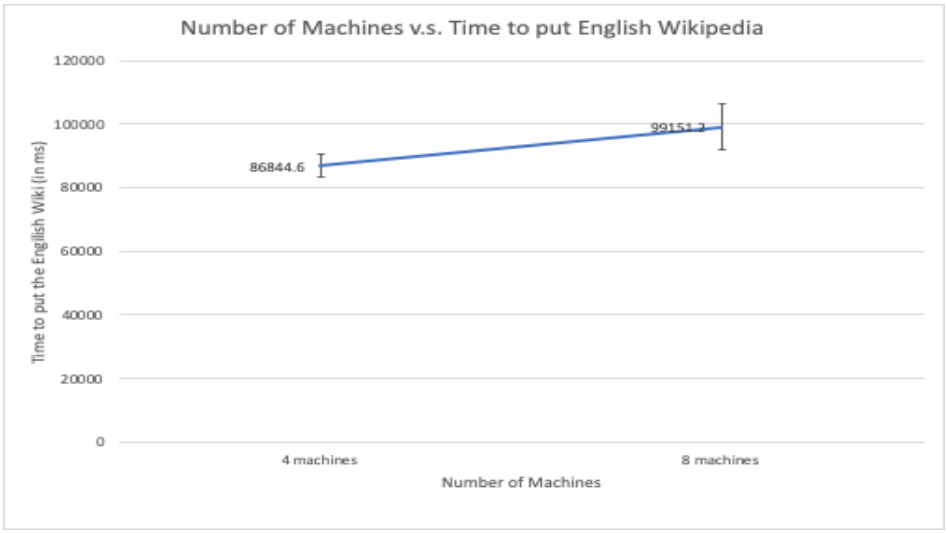

We can see that the time to get a file increases as file sizes increase, which meets our expectations. As file size increases, the bandwidth increases, so it takes longer to fetch a file.
(The standard deviation bars may seem vague due to picture size, please zoom in to view it.)

**(ii) Time to put a file vs. File size** (file size ranging from 1 MB to 1 GB)




We can see that the time to put a file increases as file sizes increase, which meets our expectations. Similar to the reasoning in (i), as file size increases, it takes longer to send the file. Writing is slower than reading because a file is transmitted at least 3 times in writing while it is only transmitted once in reading.

**(iii) time to store the entire English Wikipedia corpus into SDFS with 4 machines and with 8 machines** (not counting the master).




From the graph, we can see that the time to put English Wikipedia is slightly longer in 8 machines than in 4 machines. This meets our expectations. In 4 machines, the node that contains the file and initiates the put request is guaranteed to be chosen as the replica, so we only need to send the file to 3 other nodes. However, in 8 machines, the node that initiates the request may not be the replica node, so it may need to send the file to 4 other nodes. This is why it takes longer to put the file in 8 machines than in 4 machines. This also explains why the standard deviation is higher when we use 8 machines. In 8 machines, we send the file to 3 or 4 other nodes, while in 4 machines, we always send the file to 3 other nodes.

(P.S. Our detailed test data is stored in this link.)