CS 421

Computer Networks

Programming Assignment

Report

Maryam Shahid 21801344

December 25, 2021

The user is able to run our ParallelFileDownloader program using the following command:

```
python3 ParallelFileDownloader.py <index_file> <connection_count>
```

Once the index file is received, the URL is parsed to extract the path. Alongside, a socket is created with the host's IP address and port number 80.
After this, a GET REQUEST is sent to the index_file as shown in the following code segment:

```
def getRequest(path, ipAddr):
    getReq = "GET " + path + " HTTP/1.0\r\nHost: " + ipAddr + "\r\n\r\n"
    getReq = str.encode(getReq)
    sock.send(getReq)
```

Once the request's content is received, the number of files along with their respective URLs is calculated and stored in an array.

Next for each file, in a sequential manner, a socket is initialized and a HEAD request is sent as shown in the following snippet:

```
def headRequest(path, ipAddr):
    headReq = "HEAD " + path + " HTTP/1.0\r\nHost: " + ipAddr + "\r\n\r\n"
    headReq = str.encode(headReq)
    sock.send(headReq)
```

The response of the HEAD request is checked and if it is '200 OK', the next part is carried out, else an error message is printed.

Next, the byte count for each thread is calculated according to the file's size. If file size is divisible with the number of threads, the respective number of threads are initialized with equal number of byte size as shown in the following code snippet:

```
if n % threadCount == 0:
    byteCount = int(n / threadCount)

    for threadNumber in range(threadCount):
        start = byteCount * threadNumber
        end = start + byteCount - 1

        t = threading.Thread(target=thread_function, args=(start, end,
nextIPAddress, nextPath))
        t.start()
```

In case it is not divisible, (⌊n/k⌋+1) bytes are downloaded through the first (n−⌊n/k⌋*k) connections and ⌊n/k⌋ bytes are downloaded through the remaining connections. Where n is the file size and k are the thread count. This can be seen in the following code segment:

```
else:
    modThreads = n % threadCount
```

```
    remainingThreads = threadCount - modThreads
    floorBytes = math.floor(n / threadCount)
    floorBytesAddOne = floorBytes + 1
    modEndByte = 0

    # modThreads will download floorBytesAddOne
    for threadNumber in range(modThreads):
        start = floorBytesAddOne * threadNumber
        end = start + floorBytesAddOne - 1
        modEndByte = end

        t = threading.Thread(target=thread_function, args=(start, end,
nextIPAddress, nextPath))
        t.start()

    secondStart = modEndByte + 1
    secondEnd = secondStart + floorBytes - 1
    # remainingThreads will download floorBytes
    for threadNumber in range(remainingThreads):
        t = threading.Thread(target=thread_function, args=(secondStart,
secondEnd, nextIPAddress, nextPath))
        t.start()

        secondStart = secondEnd + 1
        secondEnd = secondStart + floorBytes - 1
```

Once the byte size is allotted for each thread, a RANGE GET request is called in each thread's respective function. The RANGE GET request is as followed:

```
def getRangeRequest(lowerBound, upperBound, path, ipAddr):
    lowerBound = str(lowerBound)
    upperBound = str(upperBound)
    getReq = "GET " + path + " HTTP/1.1\r\nHost: " + ipAddr + "\r\n"
"Range: bytes=  " + lowerBound + '-' + upperBound + "\r\n\r\n"
    getReq = str.encode(getReq)
    sock.send(getReq)
```

Once the data is received from each thread, it is written in the respective text file according to its start byte and the files are saved in the user's directory as shown below: