Mert Gençtürk 22003506 – Project 2 Report

Note:

In my Project, I was not able to implement a fully functional DUMP request and QUITSERVER request. DUMP request prints out some corrupted values, probably because of a shift in the file offsets. However, I could not successfully solve the problem. Another thing is QUITSERVER request. I decided to use a condition variable that wakes up main thread and a global variable that all child threats loops; however, even though I tried different approaches, I could not manage to make it work properly. Only the receiving thread and main thread exists, but other child threads continue executing. Other than this issues, I have tested all functionalities in the interactive mode to see if the requests are executed correctly, and it does. Indexing works correctly I think, but there can be some shift in the file itself. QUIT method also works properly for client.

1-Experiments with Different Client Settings

In my trials I have observed that its more efficient to use multiple threads with multiple input files rather than using a single thread and single input file. For server program with following parameters, execution time of the client programs is given:

Parameters:

```
dcount: 2
tcount: 5
vsize: 32
```

Execution time for 10 threaded client with 40 requests in each file:

```
real    0m0,063s
user    0m0,006s
sys     0m0,022s
```

Execution time for 4 threaded client with 100 requests in each file:

```
real    0m0,066s
user    0m0,007s
sys     0m0,024s
```

Execution time for 1 threaded client with 400 requests in each file:

```
real    0m0,100s
user    0m0,009s
sys     0m0,015s
```

The reason for that looks like the waiting time for each response. For multiple threaded server, multiple threaded client programs executes faster. The buffering of the messages in the frontend threads also improves performance.

2-Experiments with Different Server Settings:

For a 4 threaded client program with 100 request in each input file, different threads of the server program tested, to see response time (in this context execution time of the client program).

For 1 worker in server:

```
real    0m0,084s
user    0m0,006s
sys     0m0,031s
```

For 3 worker in server:

```
real    0m0,073s
user    0m0,007s
sys     0m0,023s
```

For 5 worker in server:

```
real    0m0,066s
user    0m0,007s
sys     0m0,024s
```

From the experiment results it can be seen that for multiple threaded client program, multiple threaded server is more efficient. Even though one file is accessed by one thread at a given time, requests that requires access to the different files are executed concurrently. Therefore, response time of the server is decreased. In the context of the programs, response time is orthogonally related with the throughput, therefore, when response time is decreased, it can be sad that throughput is increased. For the context of the requests, client program ends after all requests are executed, therefore execution time is directly related to response time.

3-Experiment with Value Size:

In the experiments, it has observed that there is no relation with execution or response time with value size. Experiment results can be seen below:

Experiment Settings:

Server tcount : 3, dcount :2 ; Client clicount: 4, request per file : 100

vsize: 16

```
real    0m0,070s
user    0m0,012s
sys     0m0,007s
```

vsize: 32

```
real    0m0,073s
user    0m0,007s
sys     0m0,023s
```

vsize: 64

```
real    0m0,073s
user    0m0,010s
sys     0m0,014s
```