Seminar Report: Opty

GOKTUG CENGIZ
ANAIS CAROLINA MARTINEZ MANZANILLA
JONNATAN MENDOZA ESCOBAR

June 2, 2019

1 Introduction

The second seminar is focused on implementing a transaction server using optimistic concurrency control with backward validation. We will also learn how to implement a updatable data structure that can be accessed by concurrent, possibly distributed, processes. Optimistic concurrency control is a method that can be applied to transactional systems such as relational database management systems and software transactional memory.

2 Experiments

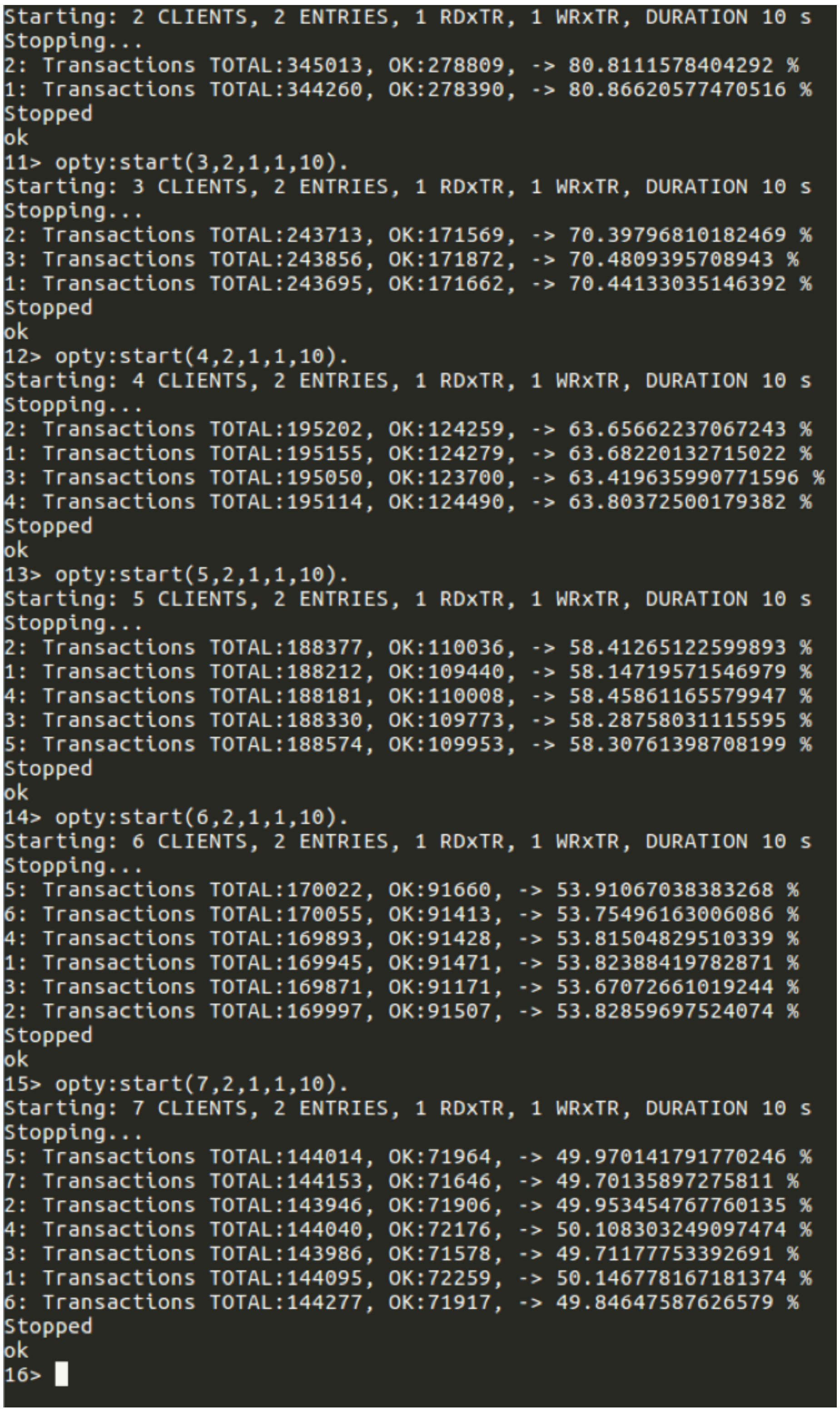
2.1 Experiment 1:

In the first experiment, we are going to test the performance of our transaction server, by updating the different parameters that our process has. For the start scenario, we are going to establish as basic parameters the following, for:

- Clients (Number of concurrent clients in the system): 2
- Entries (Number of entries in the store): 2
- Read (Number of read operations per transaction): 1
- Write (Number of write operations per transaction): 1
- Time (Duration of the experiment in seconds): 10

Now, we are going to have different scenarios changing the values one at a time that are defined in the process as follow:

I. Different number of concurrent clients in the system: in the first test we increment the number of clients from 2 to 50 (once we obtained the observation number 10, we increased from 10 to 10 in order to obtain a more detailed graph of the behavior of our process), to can see more clear the performance of our transaction server.



```
Starting: 2 CLIENTS, 2 ENTRIES, 1 RDxTR, 1 WRxTR, DURATION 10 s
Stopping...
2: Transactions TOTAL:345013, OK:278809, -> 80.8111578404292 %
1: Transactions TOTAL:344260, OK:278390, -> 80.86620577470516 %
Stopped
ok
11> opty:start(3,2,1,1,10).
Starting: 3 CLIENTS, 2 ENTRIES, 1 RDxTR, 1 WRxTR, DURATION 10 s
Stopping...
2: Transactions TOTAL:243713, OK:171569, -> 70.39796810182469 %
3: Transactions TOTAL:243856, OK:171872, -> 70.4809395708943 %
1: Transactions TOTAL:243695, OK:171662, -> 70.44133035146392 %
Stopped
ok
12> opty:start(4,2,1,1,10).
Starting: 4 CLIENTS, 2 ENTRIES, 1 RDxTR, 1 WRxTR, DURATION 10 s
Stopping...
2: Transactions TOTAL:195202, OK:124259, -> 63.65662237067243 %
1: Transactions TOTAL:195155, OK:124279, -> 63.68220132715022 %
3: Transactions TOTAL:195050, OK:123700, -> 63.419635990771596 %
4: Transactions TOTAL:195114, OK:124490, -> 63.80372500179382 %
Stopped
ok
13> opty:start(5,2,1,1,10).
Starting: 5 CLIENTS, 2 ENTRIES, 1 RDxTR, 1 WRxTR, DURATION 10 s
Stopping...
2: Transactions TOTAL:188377, OK:110036, -> 58.41265122599893 %
1: Transactions TOTAL:188212, OK:109440, -> 58.14719571546979 %
4: Transactions TOTAL:188181, OK:110008, -> 58.45861165579947 %
3: Transactions TOTAL:188330, OK:109773, -> 58.28758031115595 %
5: Transactions TOTAL:188574, OK:109953, -> 58.30761398708199 %
Stopped
ok
14> opty:start(6,2,1,1,10).
Starting: 6 CLIENTS, 2 ENTRIES, 1 RDxTR, 1 WRxTR, DURATION 10 s
Stopping...
5: Transactions TOTAL:170022, OK:91660, -> 53.91067038383268 %
6: Transactions TOTAL:170055, OK:91413, -> 53.75496163006086 %
4: Transactions TOTAL:169893, OK:91428, -> 53.81504829510339 %
1: Transactions TOTAL:169945, OK:91471, -> 53.82388419782871 %
3: Transactions TOTAL:169871, OK:91171, -> 53.67072661019244 %
2: Transactions TOTAL:169997, OK:91507, -> 53.82859697524074 %
Stopped
ok
15> opty:start(7,2,1,1,10).
Starting: 7 CLIENTS, 2 ENTRIES, 1 RDxTR, 1 WRxTR, DURATION 10 s
Stopping...
5: Transactions TOTAL:144014, OK:71964, -> 49.970141791770246 %
7: Transactions TOTAL:144153, OK:71646, -> 49.70135897275811 %
2: Transactions TOTAL:143946, OK:71906, -> 49.953454767760135 %
4: Transactions TOTAL:144040, OK:72176, -> 50.108303249097474 %
3: Transactions TOTAL:143986, OK:71578, -> 49.71177753392691 %
1: Transactions TOTAL:144095, OK:72259, -> 50.146778167181374 %
6: Transactions TOTAL:144277, OK:71917, -> 49.84647587626579 %
Stopped
ok
16> 
```

Figure 1: Execution of the process with the different values of clients

By plotting the values obtain from the previous execution, we could observed that an increment in the number of concurrent clients will reduce the performance of all.

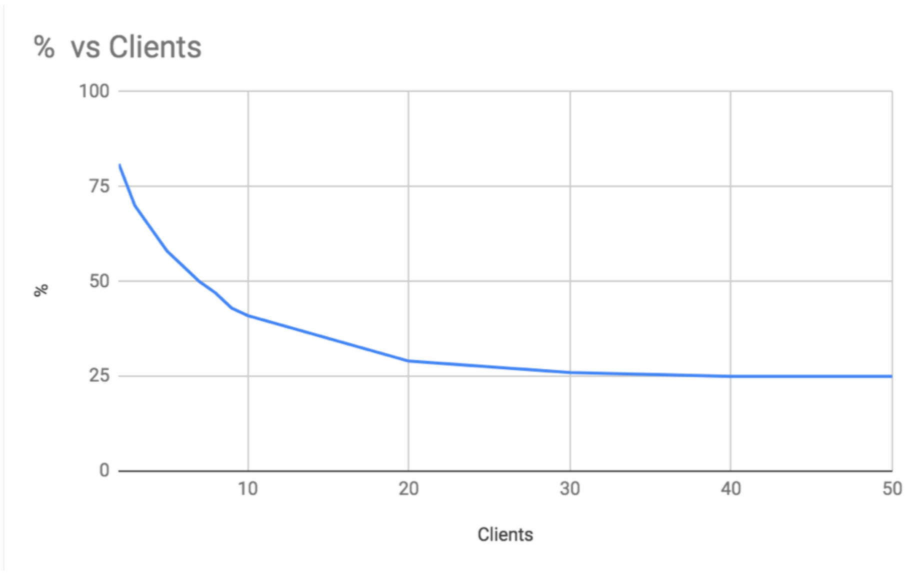
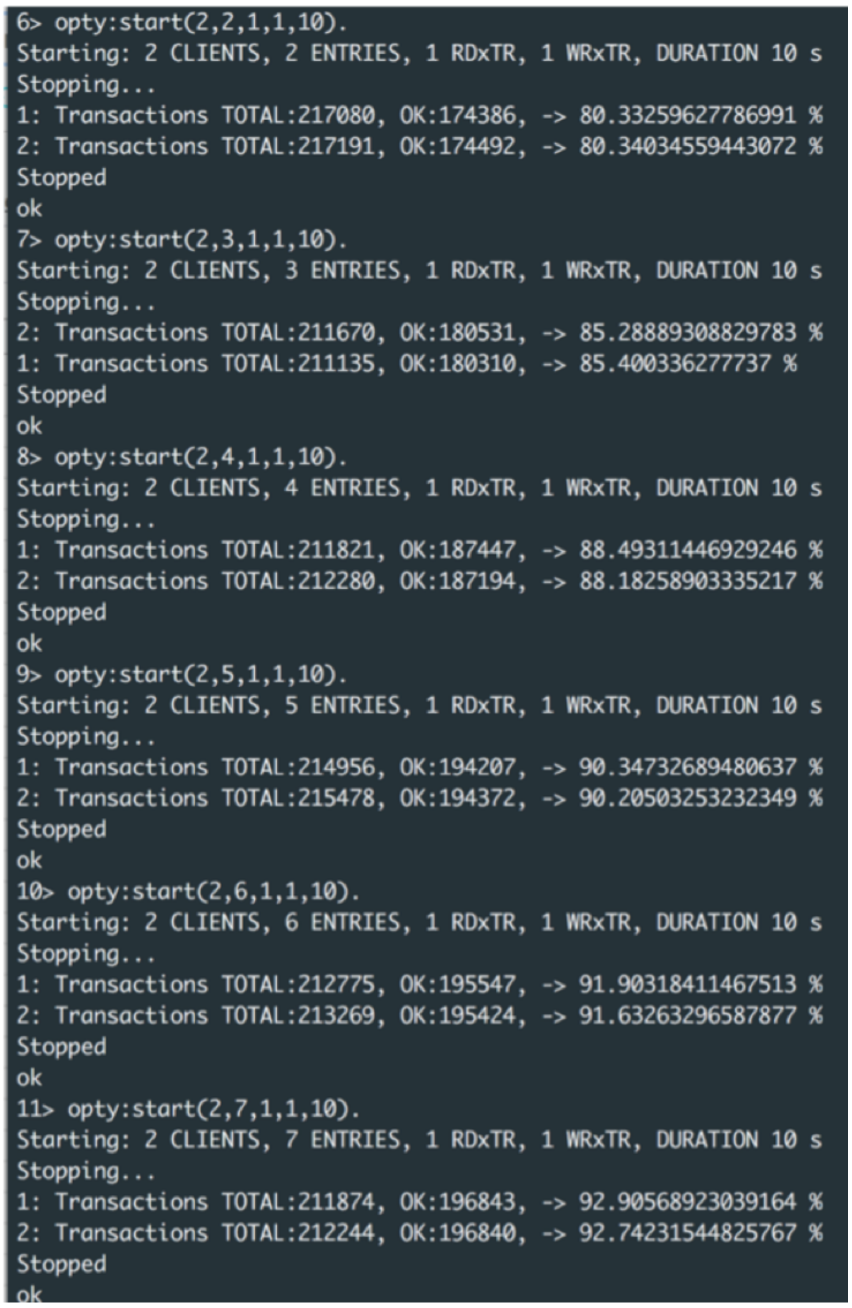


Figure 2: Percentage of transactions against the number of clients

II. Different number of entries in the store: in the second test we increment the number of entries from 2 to 50, to can see more clear the performance of our transaction server.



```
6> opty:start(2,2,1,1,10).
Starting: 2 CLIENTS, 2 ENTRIES, 1 RDxTR, 1 WRxTR, DURATION 10 s
Stopping...
1: Transactions TOTAL:217080, OK:174386, -> 80.33259627786991 %
2: Transactions TOTAL:217191, OK:174492, -> 80.34034559443072 %
Stopped
ok
7> opty:start(2,3,1,1,10).
Starting: 2 CLIENTS, 3 ENTRIES, 1 RDxTR, 1 WRxTR, DURATION 10 s
Stopping...
2: Transactions TOTAL:211670, OK:180531, -> 85.28889308829783 %
1: Transactions TOTAL:211135, OK:180310, -> 85.400336277737 %
Stopped
ok
8> opty:start(2,4,1,1,10).
Starting: 2 CLIENTS, 4 ENTRIES, 1 RDxTR, 1 WRxTR, DURATION 10 s
Stopping...
1: Transactions TOTAL:211821, OK:187447, -> 88.49311446929246 %
2: Transactions TOTAL:212280, OK:187194, -> 88.18258903335217 %
Stopped
ok
9> opty:start(2,5,1,1,10).
Starting: 2 CLIENTS, 5 ENTRIES, 1 RDxTR, 1 WRxTR, DURATION 10 s
Stopping...
1: Transactions TOTAL:214956, OK:194207, -> 90.34732689480637 %
2: Transactions TOTAL:215478, OK:194372, -> 90.20503253232349 %
Stopped
ok
10> opty:start(2,6,1,1,10).
Starting: 2 CLIENTS, 6 ENTRIES, 1 RDxTR, 1 WRxTR, DURATION 10 s
Stopping...
1: Transactions TOTAL:212775, OK:195547, -> 91.90318411467513 %
2: Transactions TOTAL:213269, OK:195424, -> 91.63263296587877 %
Stopped
ok
11> opty:start(2,7,1,1,10).
Starting: 2 CLIENTS, 7 ENTRIES, 1 RDxTR, 1 WRxTR, DURATION 10 s
Stopping...
1: Transactions TOTAL:211874, OK:196843, -> 92.90568923039164 %
2: Transactions TOTAL:212244, OK:196840, -> 92.74231544825767 %
Stopped
ok
```

Figure 3: Execution of the process with the different values of entries

By plotting the values obtain from the previous execution, we could observed that an increment in the number of entries while keeping the other values fixed will increment the performance of our server.

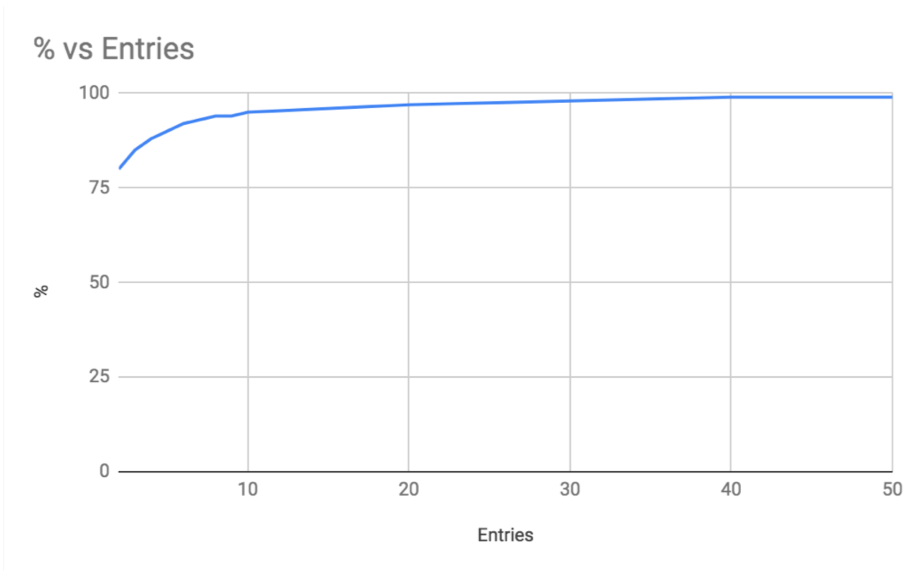
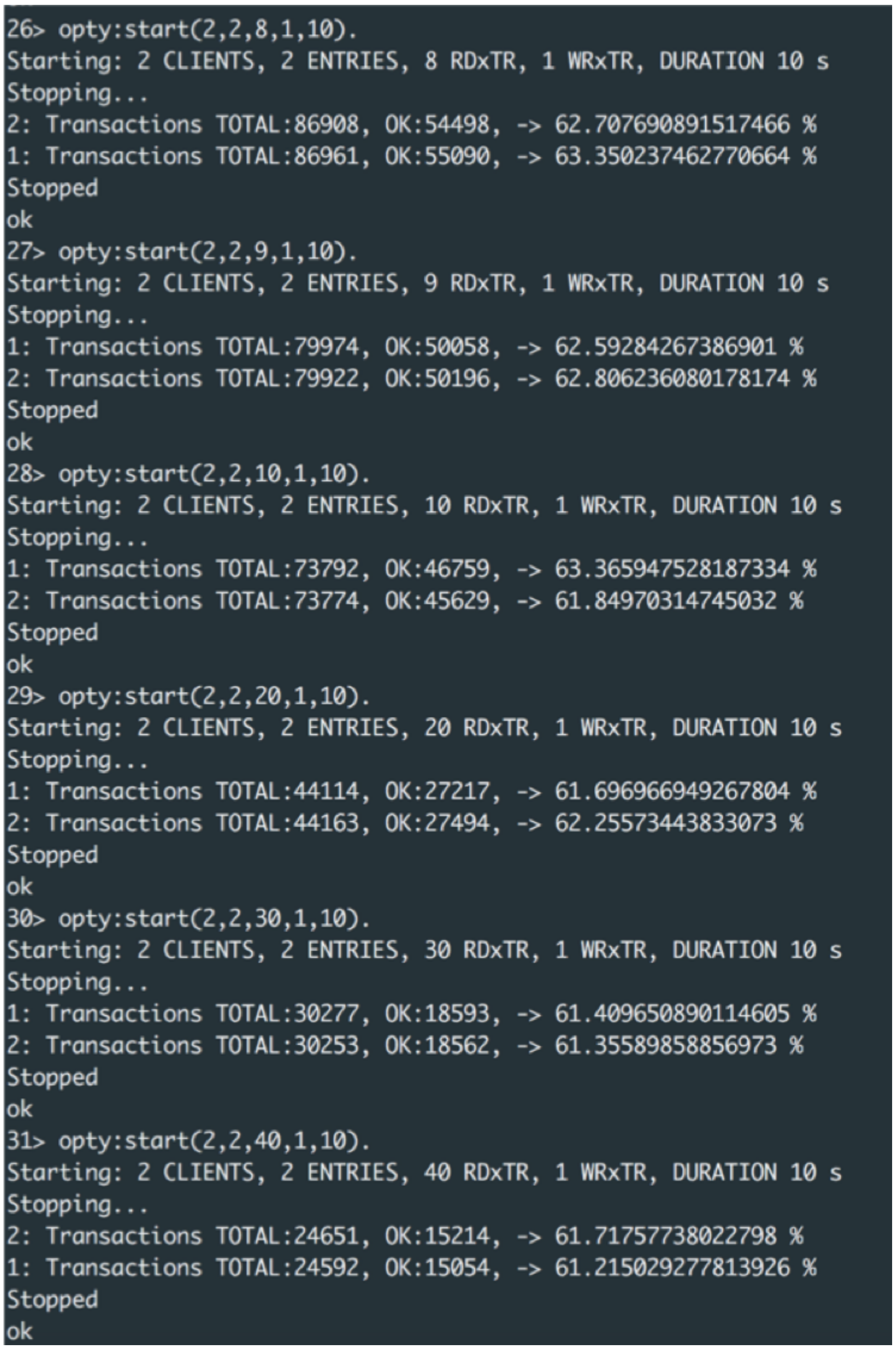


Figure 4: Percentage of transactions against the number of entries

III. Different number of read operations per transaction: in the third test we increment the number of read operations per transaction from 1 to 50, to can see more clear the performance of our transaction server.



```
26> opty:start(2,2,8,1,10).
Starting: 2 CLIENTS, 2 ENTRIES, 8 RDxTR, 1 WRxTR, DURATION 10 s
Stopping...
2: Transactions TOTAL:86908, OK:54498, -> 62.707690891517466 %
1: Transactions TOTAL:86961, OK:55090, -> 63.350237462770664 %
Stopped
ok
27> opty:start(2,2,9,1,10).
Starting: 2 CLIENTS, 2 ENTRIES, 9 RDxTR, 1 WRxTR, DURATION 10 s
Stopping...
1: Transactions TOTAL:79974, OK:50058, -> 62.59284267386901 %
2: Transactions TOTAL:79922, OK:50196, -> 62.806236080178174 %
Stopped
ok
28> opty:start(2,2,10,1,10).
Starting: 2 CLIENTS, 2 ENTRIES, 10 RDxTR, 1 WRxTR, DURATION 10 s
Stopping...
1: Transactions TOTAL:73792, OK:46759, -> 63.365947528187334 %
2: Transactions TOTAL:73774, OK:45629, -> 61.84970314745032 %
Stopped
ok
29> opty:start(2,2,20,1,10).
Starting: 2 CLIENTS, 2 ENTRIES, 20 RDxTR, 1 WRxTR, DURATION 10 s
Stopping...
1: Transactions TOTAL:44114, OK:27217, -> 61.696966949267804 %
2: Transactions TOTAL:44163, OK:27494, -> 62.25573443833073 %
Stopped
ok
30> opty:start(2,2,30,1,10).
Starting: 2 CLIENTS, 2 ENTRIES, 30 RDxTR, 1 WRxTR, DURATION 10 s
Stopping...
1: Transactions TOTAL:30277, OK:18593, -> 61.409650890114605 %
2: Transactions TOTAL:30253, OK:18562, -> 61.35589858856973 %
Stopped
ok
31> opty:start(2,2,40,1,10).
Starting: 2 CLIENTS, 2 ENTRIES, 40 RDxTR, 1 WRxTR, DURATION 10 s
Stopping...
2: Transactions TOTAL:24651, OK:15214, -> 61.71757738022798 %
1: Transactions TOTAL:24592, OK:15054, -> 61.215029277813926 %
Stopped
ok
```

Figure 5: Execution of the process with the different values of reads

By plotting the values obtain from the previous execution, we could observed that an increment in the number of reads operations per transaction while keeping the other values fixed will reduce the performance of all the clients.

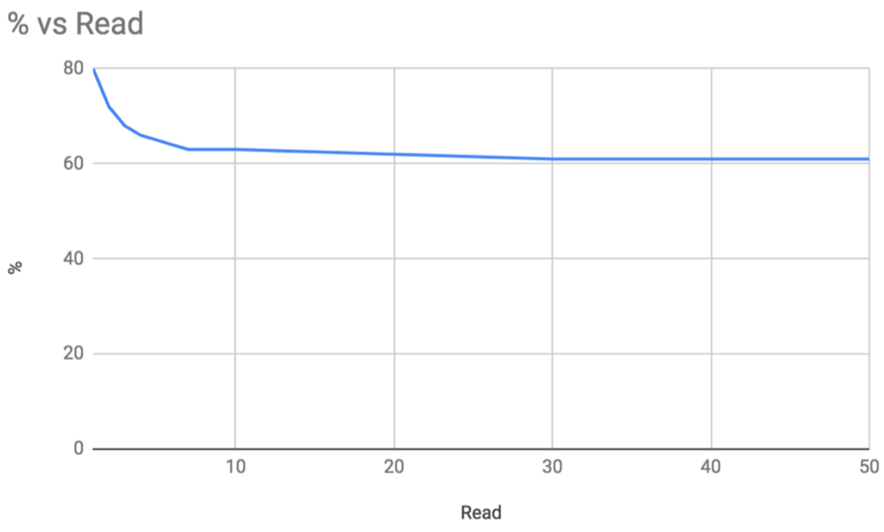
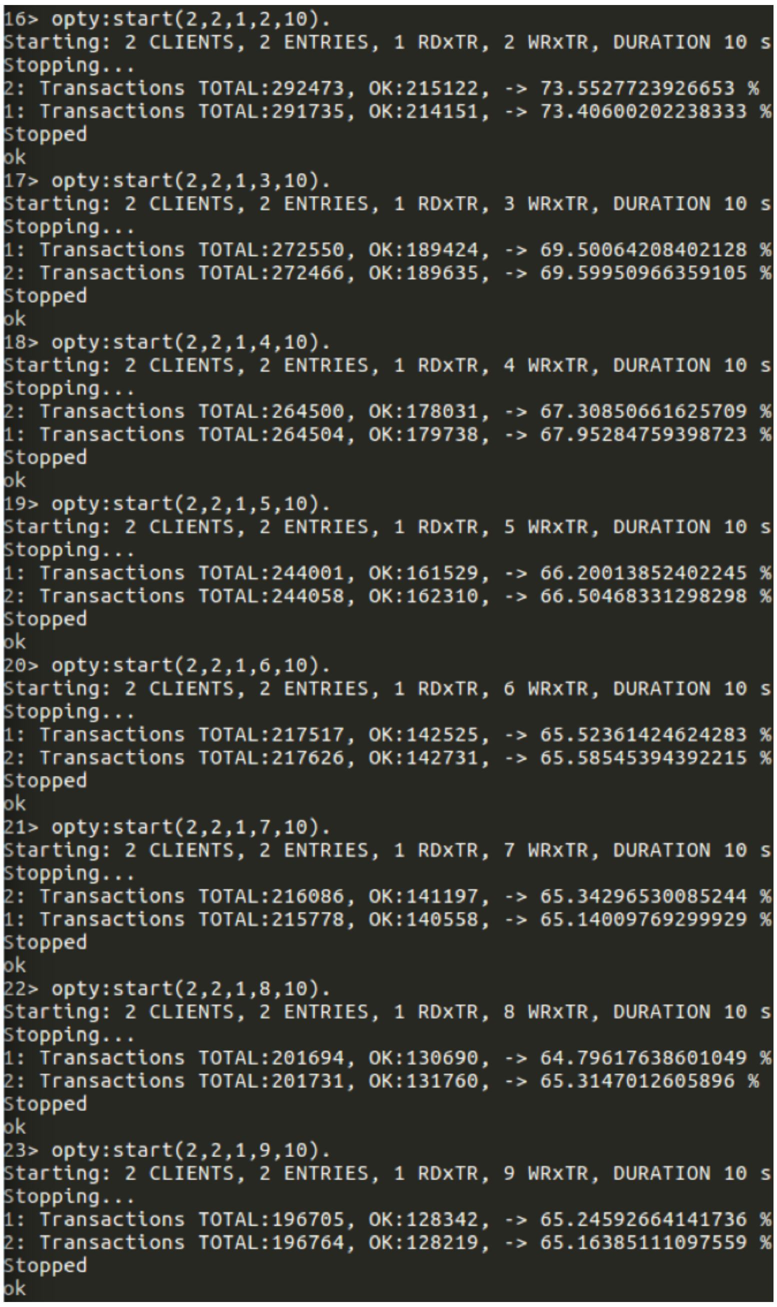


Figure 6: Percentage of transactions against the number of reads

IV. Different number of write operations per transaction: in the fourth test we increment the number of write operations per transaction from 1 to 50, to can see more clear the performance of our transaction server.



```
16> opty:start(2,2,1,2,10).
Starting: 2 CLIENTS, 2 ENTRIES, 1 RDxTR, 2 WRxTR, DURATION 10 s
Stopping...
2: Transactions TOTAL:292473, OK:215122, -> 73.5527723926653 %
1: Transactions TOTAL:291735, OK:214151, -> 73.4060020223833 %
Stopped
ok
17> opty:start(2,2,1,3,10).
Starting: 2 CLIENTS, 2 ENTRIES, 1 RDxTR, 3 WRxTR, DURATION 10 s
Stopping...
1: Transactions TOTAL:272550, OK:189424, -> 69.50064208402128 %
2: Transactions TOTAL:272466, OK:189635, -> 69.59950966359105 %
Stopped
ok
18> opty:start(2,2,1,4,10).
Starting: 2 CLIENTS, 2 ENTRIES, 1 RDxTR, 4 WRxTR, DURATION 10 s
Stopping...
2: Transactions TOTAL:264500, OK:178031, -> 67.30850661625709 %
1: Transactions TOTAL:264504, OK:179738, -> 67.95284759398723 %
Stopped
ok
19> opty:start(2,2,1,5,10).
Starting: 2 CLIENTS, 2 ENTRIES, 1 RDxTR, 5 WRxTR, DURATION 10 s
Stopping...
1: Transactions TOTAL:244001, OK:161529, -> 66.20013852402245 %
2: Transactions TOTAL:244058, OK:162310, -> 66.50468331298298 %
Stopped
ok
20> opty:start(2,2,1,6,10).
Starting: 2 CLIENTS, 2 ENTRIES, 1 RDxTR, 6 WRxTR, DURATION 10 s
Stopping...
1: Transactions TOTAL:217517, OK:142525, -> 65.52361424624283 %
2: Transactions TOTAL:217626, OK:142731, -> 65.58545394392215 %
Stopped
ok
21> opty:start(2,2,1,7,10).
Starting: 2 CLIENTS, 2 ENTRIES, 1 RDxTR, 7 WRxTR, DURATION 10 s
Stopping...
2: Transactions TOTAL:216086, OK:141197, -> 65.34296530085244 %
1: Transactions TOTAL:215778, OK:140558, -> 65.14009769299929 %
Stopped
ok
22> opty:start(2,2,1,8,10).
Starting: 2 CLIENTS, 2 ENTRIES, 1 RDxTR, 8 WRxTR, DURATION 10 s
Stopping...
1: Transactions TOTAL:201694, OK:130690, -> 64.79617638601049 %
2: Transactions TOTAL:201731, OK:131760, -> 65.3147012605896 %
Stopped
ok
23> opty:start(2,2,1,9,10).
Starting: 2 CLIENTS, 2 ENTRIES, 1 RDxTR, 9 WRxTR, DURATION 10 s
Stopping...
1: Transactions TOTAL:196705, OK:128342, -> 65.24592664141736 %
2: Transactions TOTAL:196764, OK:128219, -> 65.16385111097559 %
Stopped
ok
```

Figure 7: Execution of the process with the different values of writes

By plotting the values obtain from the previous execution, we could observed that an increment in the number of writes operations per transaction while keeping the other values fixed will reduce the performance of all the clients.

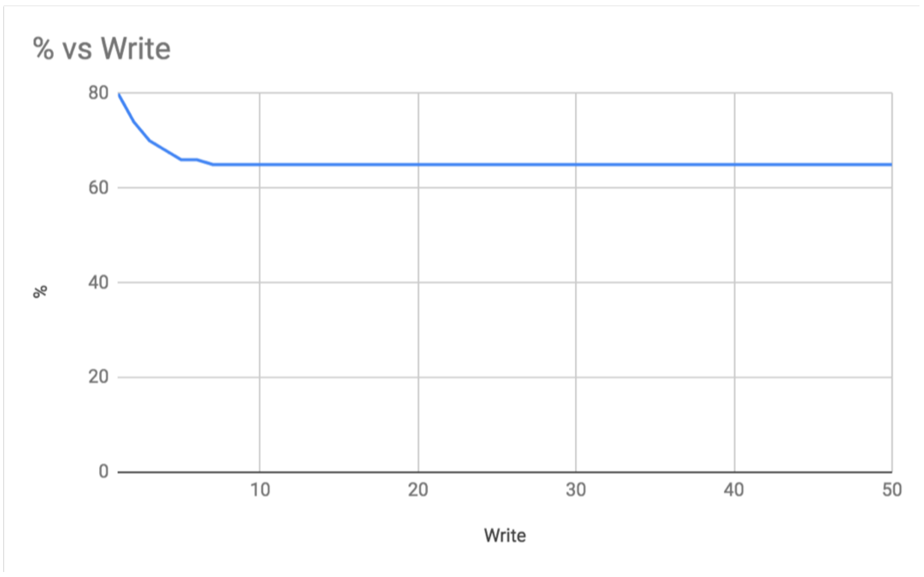


Figure 8: Percentage of transactions against the number of writes

V. Different ratio of read and write operations for a fixed amount of operations per transaction: for this experiment we chose the ratio of the operations as is shown in the following table, and the amount of read and writes match the ratio, also the table shows both special cases with no reads and no writes in all cases the amount of operations is the same,the number of clients and entries remains in two and the simulations is for 10 secs, the last colum of the table show the results of the experiment.

Reads	Ratio	Writes	Results C1 %	Results C2 %
10	10:0	0	100%	100%
9	9:1	1	62.45%	63.25%
8	8:2	2	60.77%	61.3%
7	7:3	3	60.2%	60.32%
6	6:4	4	59.33%	59.98%
5	5:5	5	59.19%	59.39%
4	4:6	6	59.07%	59.26%
3	3:7	7	59.29%	59.50%
2	2:8	8	61.51%	60.59%
1	1:9	9	65.01%	65.11%
0	0:10	10	100%	100%

Table 1: Diferent ratios for read and writes.

By plotting the values of the previous execution we can observed how reads and writes obstruct each other, this is reflected in the decreasing of the performance while the ratio approximates to be 1 (5:5).

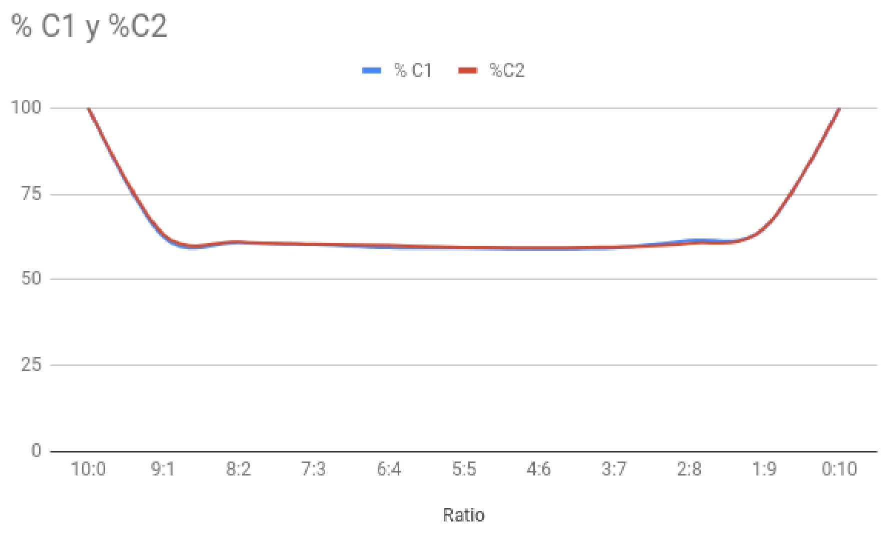
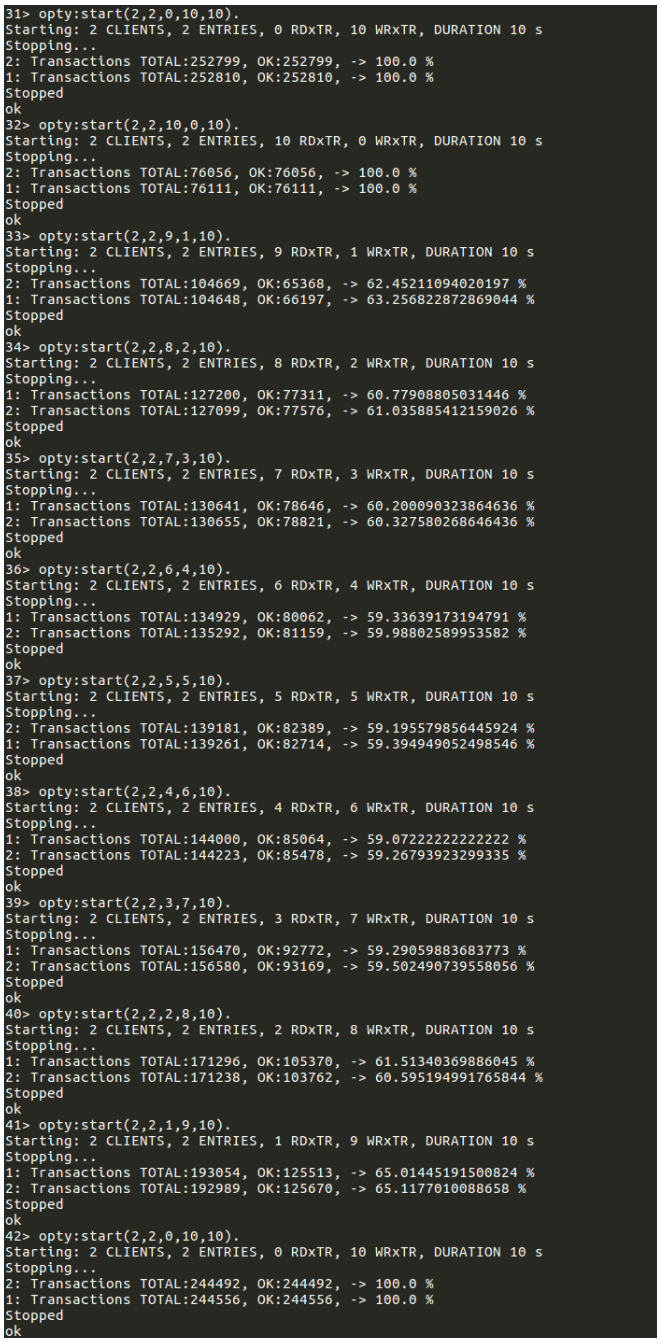


Figure 9: Performance plot with diferent ratios of read and write operations.



```
31> opty:start(2,2,0,10,10).
Starting: 2 CLIENTS, 2 ENTRIES, 0 RDxTR, 10 WRxTR, DURATION 10 s
Stopping...
2: Transactions TOTAL:252799, OK:252799, -> 100.0 %
1: Transactions TOTAL:252810, OK:252810, -> 100.0 %
Stopped
ok
32> opty:start(2,2,10,0,10).
Starting: 2 CLIENTS, 2 ENTRIES, 10 RDxTR, 0 WRxTR, DURATION 10 s
Stopping...
2: Transactions TOTAL:76056, OK:76056, -> 100.0 %
1: Transactions TOTAL:76111, OK:76111, -> 100.0 %
Stopped
ok
33> opty:start(2,2,9,1,10).
Starting: 2 CLIENTS, 2 ENTRIES, 9 RDxTR, 1 WRxTR, DURATION 10 s
Stopping...
2: Transactions TOTAL:104669, OK:65368, -> 62.45211094020197 %
1: Transactions TOTAL:104648, OK:66197, -> 63.256822872869044 %
Stopped
ok
34> opty:start(2,2,8,2,10).
Starting: 2 CLIENTS, 2 ENTRIES, 8 RDxTR, 2 WRxTR, DURATION 10 s
Stopping...
1: Transactions TOTAL:127200, OK:77311, -> 60.77908805031446 %
2: Transactions TOTAL:127099, OK:77576, -> 61.035885412159026 %
Stopped
ok
35> opty:start(2,2,7,3,10).
Starting: 2 CLIENTS, 2 ENTRIES, 7 RDxTR, 3 WRxTR, DURATION 10 s
Stopping...
1: Transactions TOTAL:130641, OK:78646, -> 60.200090323864636 %
2: Transactions TOTAL:130655, OK:78821, -> 60.327580268646436 %
Stopped
ok
36> opty:start(2,2,6,4,10).
Starting: 2 CLIENTS, 2 ENTRIES, 6 RDxTR, 4 WRxTR, DURATION 10 s
Stopping...
1: Transactions TOTAL:134929, OK:80062, -> 59.33639173194791 %
2: Transactions TOTAL:135292, OK:81159, -> 59.98802589953582 %
Stopped
ok
37> opty:start(2,2,5,5,10).
Starting: 2 CLIENTS, 2 ENTRIES, 5 RDxTR, 5 WRxTR, DURATION 10 s
Stopping...
2: Transactions TOTAL:139181, OK:82389, -> 59.195579856445924 %
1: Transactions TOTAL:139261, OK:82714, -> 59.394949052498546 %
Stopped
ok
38> opty:start(2,2,4,6,10).
Starting: 2 CLIENTS, 2 ENTRIES, 4 RDxTR, 6 WRxTR, DURATION 10 s
Stopping...
1: Transactions TOTAL:144000, OK:85064, -> 59.07222222222222 %
2: Transactions TOTAL:144223, OK:85478, -> 59.26793923299335 %
Stopped
ok
39> opty:start(2,2,3,7,10).
Starting: 2 CLIENTS, 2 ENTRIES, 3 RDxTR, 7 WRxTR, DURATION 10 s
Stopping...
1: Transactions TOTAL:156470, OK:92772, -> 59.29059883683773 %
2: Transactions TOTAL:156580, OK:93169, -> 59.502490739558056 %
Stopped
ok
40> opty:start(2,2,2,8,10).
Starting: 2 CLIENTS, 2 ENTRIES, 2 RDxTR, 8 WRxTR, DURATION 10 s
Stopping...
1: Transactions TOTAL:171296, OK:105370, -> 61.51340369886045 %
2: Transactions TOTAL:171238, OK:103762, -> 60.595194991765844 %
Stopped
ok
41> opty:start(2,2,1,9,10).
Starting: 2 CLIENTS, 2 ENTRIES, 1 RDxTR, 9 WRxTR, DURATION 10 s
Stopping...
1: Transactions TOTAL:193054, OK:125513, -> 65.01445191500824 %
2: Transactions TOTAL:192989, OK:125670, -> 65.1177010088658 %
Stopped
ok
42> opty:start(2,2,0,10,10).
Starting: 2 CLIENTS, 2 ENTRIES, 0 RDxTR, 10 WRxTR, DURATION 10 s
Stopping...
2: Transactions TOTAL:244492, OK:244492, -> 100.0 %
1: Transactions TOTAL:244556, OK:244556, -> 100.0 %
Stopped
ok
```

Figure 10: Execution of the process with the different ratios.

VI. Different percentage of accessed entries with respect to the total number of entries (i.e. each client accesses a randomly generated subset of the store: the number of entries in each subset should be the same for all the clients, but the subsets should be different per client and should not contain necessarily contiguous entries).

2.2 Experiment 2: Distributed execution Experiments

Adapt the opty module to ensure that the clients can connect correctly to the server, which must be created in a different remote Erlang instance (named opty-srv). Note that the server has to use a locally registered name. Remember how processes are created remotely, how names registered in remote nodes are referred, and how Erlang runtime should be started to run distributed programs.

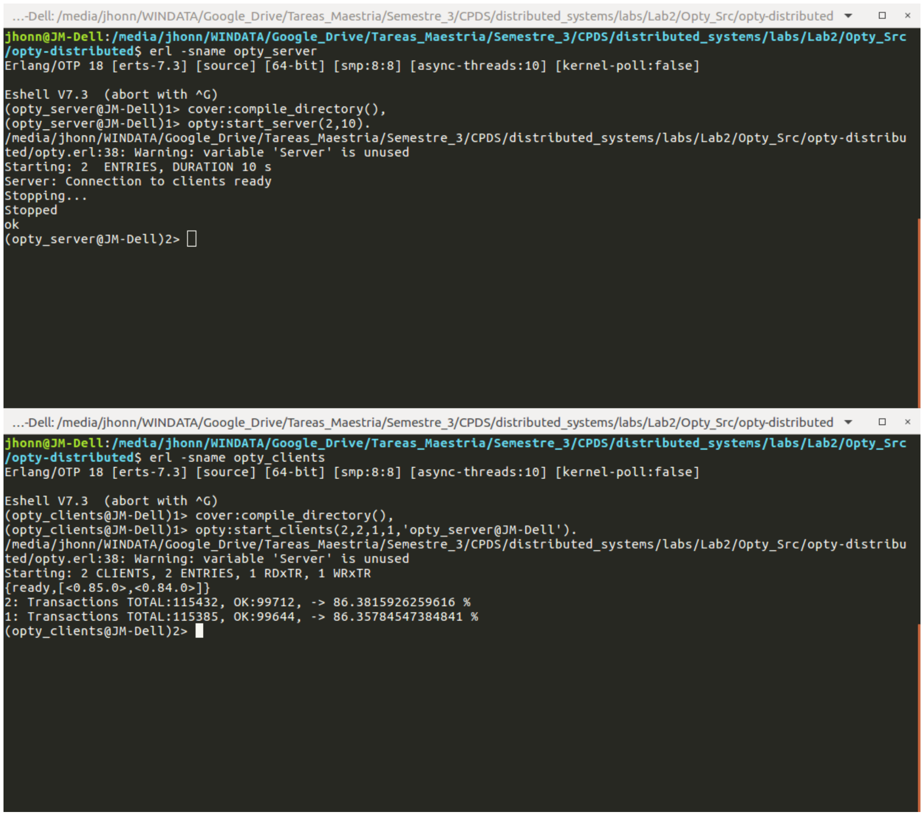
For this experiment, we create two procedures in the opty source code, one is the *opty* : *start_server* and the other is the *opty* : *start_clients*, the server will include the Store and the Validator processes, the server procedure needs to be called at first with the name of the node specified in the shell and requieres the number of entries and the time limit of the experiments, since this process will be in charge of crontrol the execution time. The server's node will wait until the node of the Clients had been ready, and then it will start the time countin, when it finish, it will send the stop messages to the Clients and the results will be displayed. The Client's node will require the identifier of the server's node. After all the clients were created the Client's node will send a ready message to the serve's node, in this message a list with all the PID's of the clients is included.

The following code shown correspons to the modifications to the opty source code.

Listing 1: Code Modfications for Distributed execution.

```
-module(opty).
-export([stop/1, start_server/2, start_clients/5]).

%% Clients: Number of concurrent clients in the system
%% Entries: Number of entries in the store
%% Reads: Number of read operations per transaction
%% Writes: Number of write operations per transaction
%% Time: Duration of the experiment (in secs)

start_server(Entries, Time) ->
    register(s, server:start(Entries)),
    register(opty, self()),
    io:format("Starting: ~w  ENTRIES, DURATION ~w s~n",
        [Entries, Time]),
    receive
        {ready, L} ->
            io:format("Server: Connection to clients ready~n")
    end,
    timer:sleep(Time*1000),
    stop(L).

start_clients(Clients, Entries, Reads, Writes, ServerNode) ->
    Server = {s, ServerNode},%%Server PID
    L = startClients(Clients, [], Entries, Reads, Writes, Server),
    io:format("Starting: ~w CLIENTS, ~w ENTRIES, ~w RDxTR, ~w WRxTR~n",
        [Clients, Entries, Reads, Writes]),
    {opty ,ServerNode} ! {ready, L}.


stop(L) ->
    io:format("Stopping...~n"),
    stopClients(L),
    waitClients(L),
    s ! stop,
    unregister(opty),
    io:format("Stopped~n").

startClients(0, L, _, _, _,Server) -> L;
startClients(Clients, L, Entries, Reads, Writes,Server) ->
    Pid = client:start(Clients, Entries, Reads, Writes, Server),
    startClients(Clients-1, [Pid|L], Entries, Reads, Writes, Server).

stopClients([]) ->
    ok;
stopClients([Pid|L]) ->
    Pid ! {stop, self()},
    stopClients(L).

waitClients([]) ->
    ok;
waitClients(L) ->
    receive
        {done, Pid} ->
            waitClients(lists:delete(Pid, L))
    end.
```

The following image shows the distributed execution of the algorithm in two diferent shells.



```
…-Dell: /media/jhonn/WINDATA/Google_Drive/Tareas_Maestria/Semestre_3/CPDS/distributed_systems/labs/Lab2/Opty_Src/opty-distributed
jhonn@JM-Dell:/media/jhonn/WINDATA/Google_Drive/Tareas_Maestria/Semestre_3/CPDS/distributed_systems/labs/Lab2/Opty_Src
/opty-distributed$ erl -sname opty_server
Erlang/OTP 18 [erts-7.3] [source] [64-bit] [smp:8:8] [async-threads:10] [kernel-poll:false]

Eshell V7.3  (abort with ^G)
(opty_server@JM-Dell)1> cover:compile_directory(),
(opty_server@JM-Dell)1> opty:start_server(2,10).
/media/jhonn/WINDATA/Google_Drive/Tareas_Maestria/Semestre_3/CPDS/distributed_systems/labs/Lab2/Opty_Src/opty-distribu
ted/opty.erl:38: Warning: variable 'Server' is unused
Starting: 2  ENTRIES, DURATION 10 s
Server: Connection to clients ready
Stopping...
Stopped
ok
(opty_server@JM-Dell)2>
```

```
…-Dell: /media/jhonn/WINDATA/Google_Drive/Tareas_Maestria/Semestre_3/CPDS/distributed_systems/labs/Lab2/Opty_Src/opty-distributed
jhonn@JM-Dell:/media/jhonn/WINDATA/Google_Drive/Tareas_Maestria/Semestre_3/CPDS/distributed_systems/labs/Lab2/Opty_Src
/opty-distributed$ erl -sname opty_clients
Erlang/OTP 18 [erts-7.3] [source] [64-bit] [smp:8:8] [async-threads:10] [kernel-poll:false]

Eshell V7.3  (abort with ^G)
(opty_clients@JM-Dell)1> cover:compile_directory(),
(opty_clients@JM-Dell)1> opty:start_clients(2,2,1,1,'opty_server@JM-Dell').
/media/jhonn/WINDATA/Google_Drive/Tareas_Maestria/Semestre_3/CPDS/distributed_systems/labs/Lab2/Opty_Src/opty-distribu
ted/opty.erl:38: Warning: variable 'Server' is unused
Starting: 2 CLIENTS, 2 ENTRIES, 1 RDxTR, 1 WRxTR
{ready,[<0.85.0>,<0.84.0>]}
2: Transactions TOTAL:115432, OK:99712, -> 86.3815926259616 %
1: Transactions TOTAL:115385, OK:99644, -> 86.35784547384841 %
(opty_clients@JM-Dell)2>
```

Figure 11: Execution of $opty : strat\_server$ and $opty : start\_clients$.

2.3 Experiment 3: Concurrency control techniques

We decided to compare the timey with the opty.

2.3.1 Changing no of clients

As we can see in the figure 12 below, initial performance of timey is better than opty. When we change the number of clients, decrements of success rate is almost similar between timey and opty.

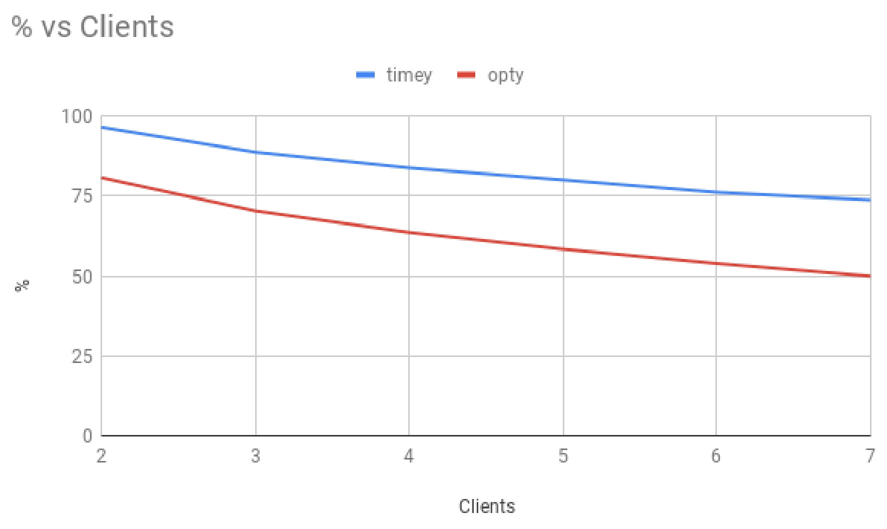
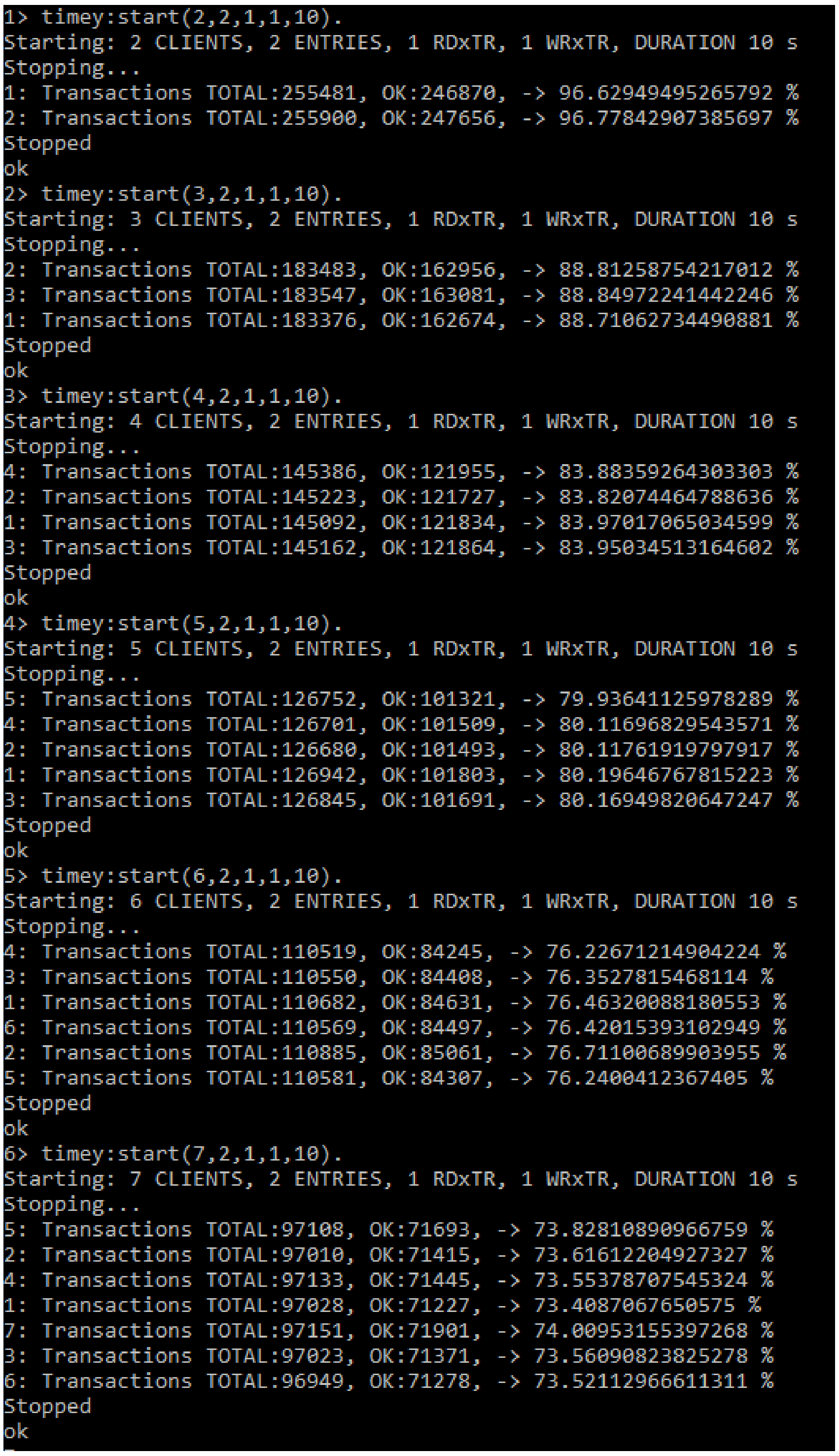


Figure 12: Different no of clients on timey vs opty



```
1> timey:start(2,2,1,1,10).
Starting: 2 CLIENTS, 2 ENTRIES, 1 RDxTR, 1 WRxTR, DURATION 10 s
Stopping...
1: Transactions TOTAL:255481, OK:246870, -> 96.62949495265792 %
2: Transactions TOTAL:255900, OK:247656, -> 96.77842907385697 %
Stopped
ok
2> timey:start(3,2,1,1,10).
Starting: 3 CLIENTS, 2 ENTRIES, 1 RDxTR, 1 WRxTR, DURATION 10 s
Stopping...
2: Transactions TOTAL:183483, OK:162956, -> 88.81258754217012 %
3: Transactions TOTAL:183547, OK:163081, -> 88.84972241442246 %
1: Transactions TOTAL:183376, OK:162674, -> 88.71062734490881 %
Stopped
ok
3> timey:start(4,2,1,1,10).
Starting: 4 CLIENTS, 2 ENTRIES, 1 RDxTR, 1 WRxTR, DURATION 10 s
Stopping...
4: Transactions TOTAL:145386, OK:121955, -> 83.88359264303303 %
2: Transactions TOTAL:145223, OK:121727, -> 83.82074464788636 %
1: Transactions TOTAL:145092, OK:121834, -> 83.97017065034599 %
3: Transactions TOTAL:145162, OK:121864, -> 83.95034513164602 %
Stopped
ok
4> timey:start(5,2,1,1,10).
Starting: 5 CLIENTS, 2 ENTRIES, 1 RDxTR, 1 WRxTR, DURATION 10 s
Stopping...
5: Transactions TOTAL:126752, OK:101321, -> 79.93641125978289 %
4: Transactions TOTAL:126701, OK:101509, -> 80.11696829543571 %
2: Transactions TOTAL:126680, OK:101493, -> 80.11761919797917 %
1: Transactions TOTAL:126942, OK:101803, -> 80.19646767815223 %
3: Transactions TOTAL:126845, OK:101691, -> 80.16949820647247 %
Stopped
ok
5> timey:start(6,2,1,1,10).
Starting: 6 CLIENTS, 2 ENTRIES, 1 RDxTR, 1 WRxTR, DURATION 10 s
Stopping...
4: Transactions TOTAL:110519, OK:84245, -> 76.22671214904224 %
3: Transactions TOTAL:110550, OK:84408, -> 76.3527815468114 %
1: Transactions TOTAL:110682, OK:84631, -> 76.46320088180553 %
6: Transactions TOTAL:110569, OK:84497, -> 76.42015393102949 %
2: Transactions TOTAL:110885, OK:85061, -> 76.71100689903955 %
5: Transactions TOTAL:110581, OK:84307, -> 76.2400412367405 %
Stopped
ok
6> timey:start(7,2,1,1,10).
Starting: 7 CLIENTS, 2 ENTRIES, 1 RDxTR, 1 WRxTR, DURATION 10 s
Stopping...
5: Transactions TOTAL:97108, OK:71693, -> 73.82810890966759 %
2: Transactions TOTAL:97010, OK:71415, -> 73.61612204927327 %
4: Transactions TOTAL:97133, OK:71445, -> 73.55378707545324 %
1: Transactions TOTAL:97028, OK:71227, -> 73.4087067650575 %
7: Transactions TOTAL:97151, OK:71901, -> 74.00953155397268 %
3: Transactions TOTAL:97023, OK:71371, -> 73.56090823825278 %
6: Transactions TOTAL:96949, OK:71278, -> 73.52112966611311 %
Stopped
ok
```

Figure 13: Execution of timey with different no of client

2.3.2 Changing no of entries

Increasing the number of entries increases the success rate for opty; but initial success rate is already very high for timey and increasing the number of entries does not have any significant impact on success rate.

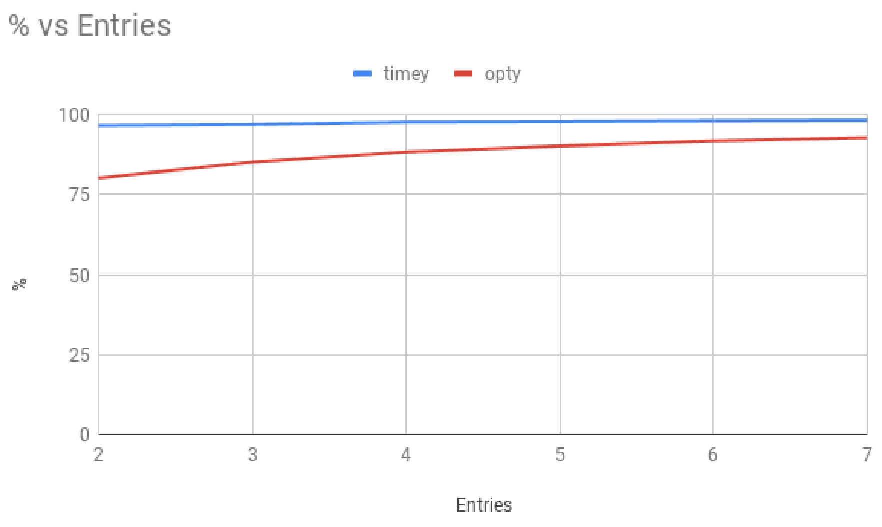
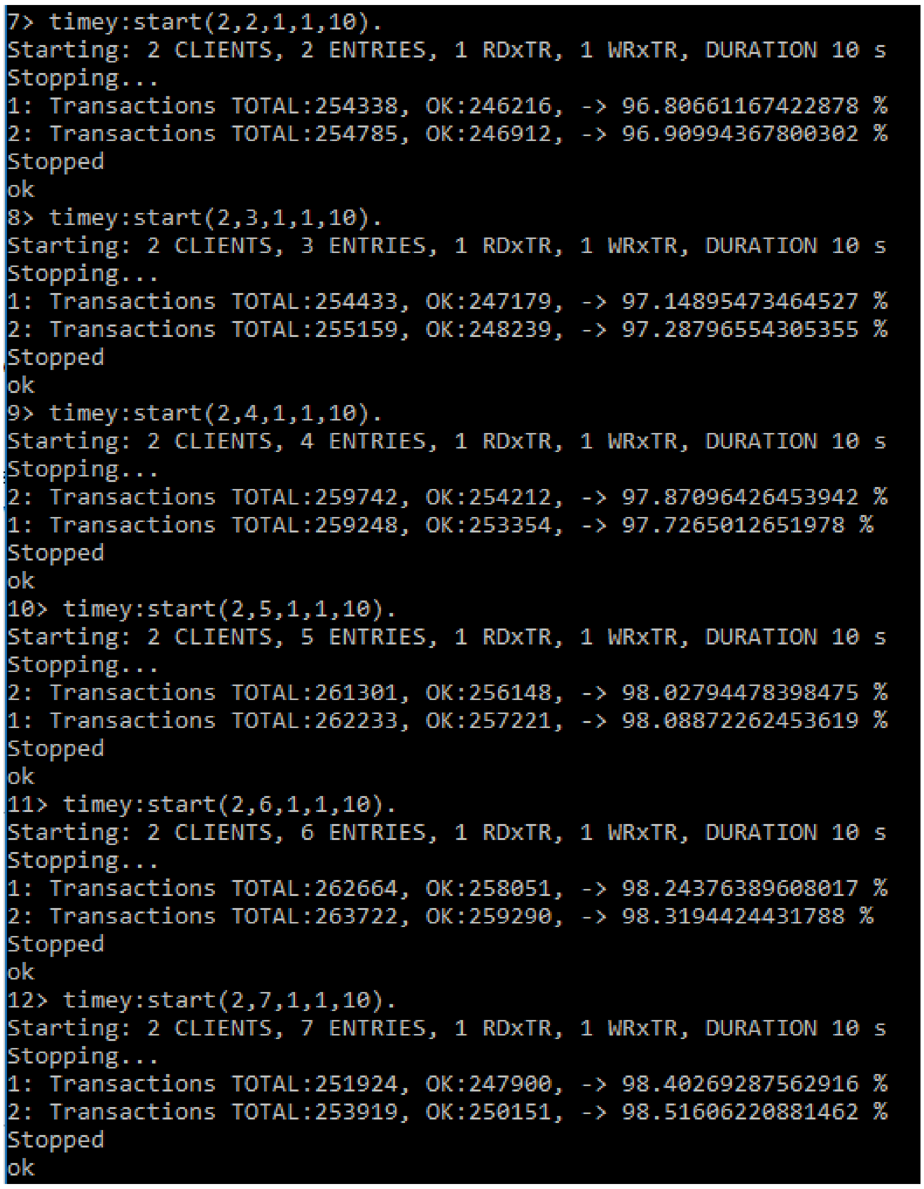


Figure 14: Different no of entries on timey vs opty



```
7> timey:start(2,2,1,1,10).
Starting: 2 CLIENTS, 2 ENTRIES, 1 RDxTR, 1 WRxTR, DURATION 10 s
Stopping...
1: Transactions TOTAL:254338, OK:246216, -> 96.80661167422878 %
2: Transactions TOTAL:254785, OK:246912, -> 96.90994367800302 %
Stopped
ok
8> timey:start(2,3,1,1,10).
Starting: 2 CLIENTS, 3 ENTRIES, 1 RDxTR, 1 WRxTR, DURATION 10 s
Stopping...
1: Transactions TOTAL:254433, OK:247179, -> 97.14895473464527 %
2: Transactions TOTAL:255159, OK:248239, -> 97.28796554305355 %
Stopped
ok
9> timey:start(2,4,1,1,10).
Starting: 2 CLIENTS, 4 ENTRIES, 1 RDxTR, 1 WRxTR, DURATION 10 s
Stopping...
2: Transactions TOTAL:259742, OK:254212, -> 97.87096426453942 %
1: Transactions TOTAL:259248, OK:253354, -> 97.7265012651978 %
Stopped
ok
10> timey:start(2,5,1,1,10).
Starting: 2 CLIENTS, 5 ENTRIES, 1 RDxTR, 1 WRxTR, DURATION 10 s
Stopping...
2: Transactions TOTAL:261301, OK:256148, -> 98.02794478398475 %
1: Transactions TOTAL:262233, OK:257221, -> 98.08872262453619 %
Stopped
ok
11> timey:start(2,6,1,1,10).
Starting: 2 CLIENTS, 6 ENTRIES, 1 RDxTR, 1 WRxTR, DURATION 10 s
Stopping...
1: Transactions TOTAL:262664, OK:258051, -> 98.24376389608017 %
2: Transactions TOTAL:263722, OK:259290, -> 98.3194424431788 %
Stopped
ok
12> timey:start(2,7,1,1,10).
Starting: 2 CLIENTS, 7 ENTRIES, 1 RDxTR, 1 WRxTR, DURATION 10 s
Stopping...
1: Transactions TOTAL:251924, OK:247900, -> 98.40269287562916 %
2: Transactions TOTAL:253919, OK:250151, -> 98.51606220881462 %
Stopped
ok
```

Figure 15: Execution of timey with different no of entries

2.3.3 Changing no of reads

Behavior of changing the number of reads is almost identical between timey and opty, it does not effect the overall performance at all.

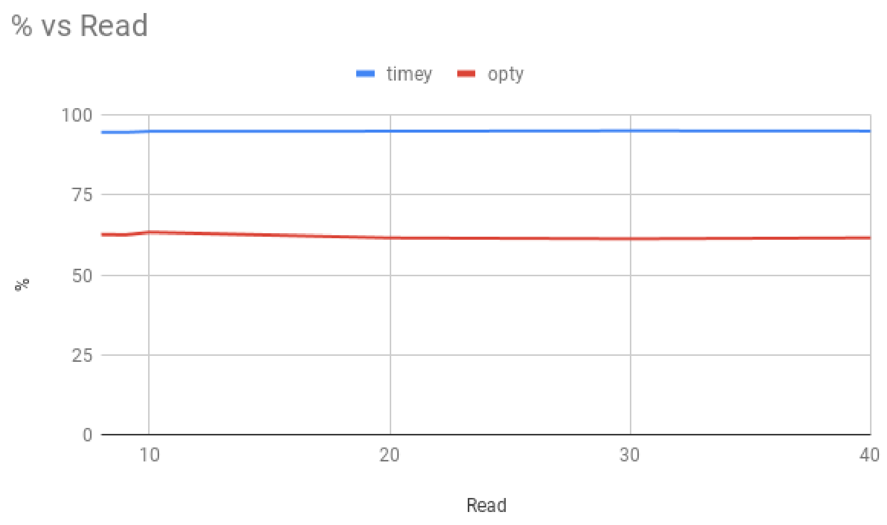
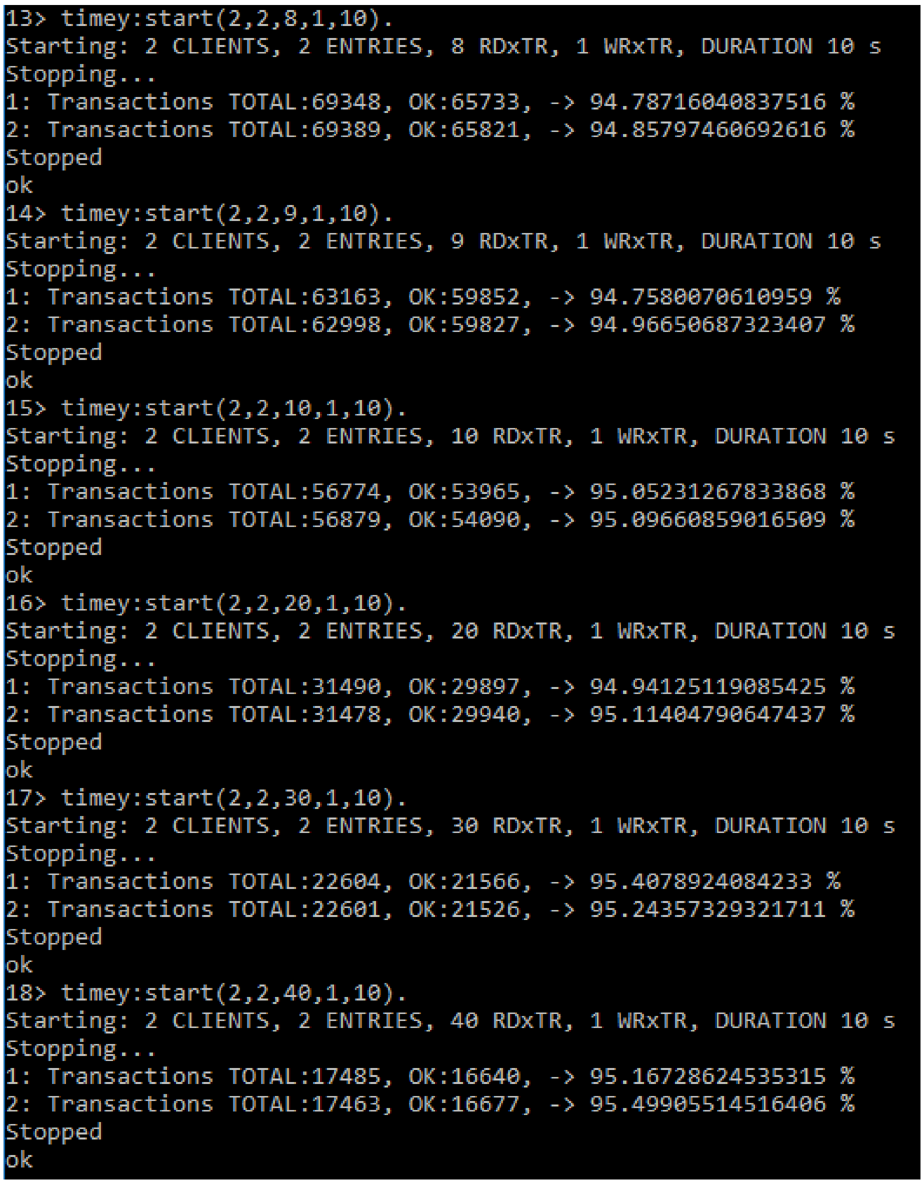


Figure 16: Different no of reads on timey vs opty



```
13> timey:start(2,2,8,1,10).
Starting: 2 CLIENTS, 2 ENTRIES, 8 RDxTR, 1 WRxTR, DURATION 10 s
Stopping...
1: Transactions TOTAL:69348, OK:65733, -> 94.78716040837516 %
2: Transactions TOTAL:69389, OK:65821, -> 94.85797460692616 %
Stopped
ok
14> timey:start(2,2,9,1,10).
Starting: 2 CLIENTS, 2 ENTRIES, 9 RDxTR, 1 WRxTR, DURATION 10 s
Stopping...
1: Transactions TOTAL:63163, OK:59852, -> 94.7580070610959 %
2: Transactions TOTAL:62998, OK:59827, -> 94.96650687323407 %
Stopped
ok
15> timey:start(2,2,10,1,10).
Starting: 2 CLIENTS, 2 ENTRIES, 10 RDxTR, 1 WRxTR, DURATION 10 s
Stopping...
1: Transactions TOTAL:56774, OK:53965, -> 95.05231267833868 %
2: Transactions TOTAL:56879, OK:54090, -> 95.09660859016509 %
Stopped
ok
16> timey:start(2,2,20,1,10).
Starting: 2 CLIENTS, 2 ENTRIES, 20 RDxTR, 1 WRxTR, DURATION 10 s
Stopping...
1: Transactions TOTAL:31490, OK:29897, -> 94.94125119085425 %
2: Transactions TOTAL:31478, OK:29940, -> 95.11404790647437 %
Stopped
ok
17> timey:start(2,2,30,1,10).
Starting: 2 CLIENTS, 2 ENTRIES, 30 RDxTR, 1 WRxTR, DURATION 10 s
Stopping...
1: Transactions TOTAL:22604, OK:21566, -> 95.4078924084233 %
2: Transactions TOTAL:22601, OK:21526, -> 95.24357329321711 %
Stopped
ok
18> timey:start(2,2,40,1,10).
Starting: 2 CLIENTS, 2 ENTRIES, 40 RDxTR, 1 WRxTR, DURATION 10 s
Stopping...
1: Transactions TOTAL:17485, OK:16640, -> 95.16728624535315 %
2: Transactions TOTAL:17463, OK:16677, -> 95.49905514516406 %
Stopped
ok
```

Figure 17: Execution of timey with different no of reads

2.3.4 Changing no of writes

Changing the number of write transactions decrease the success rate of timey, with a similar rate to the opty.

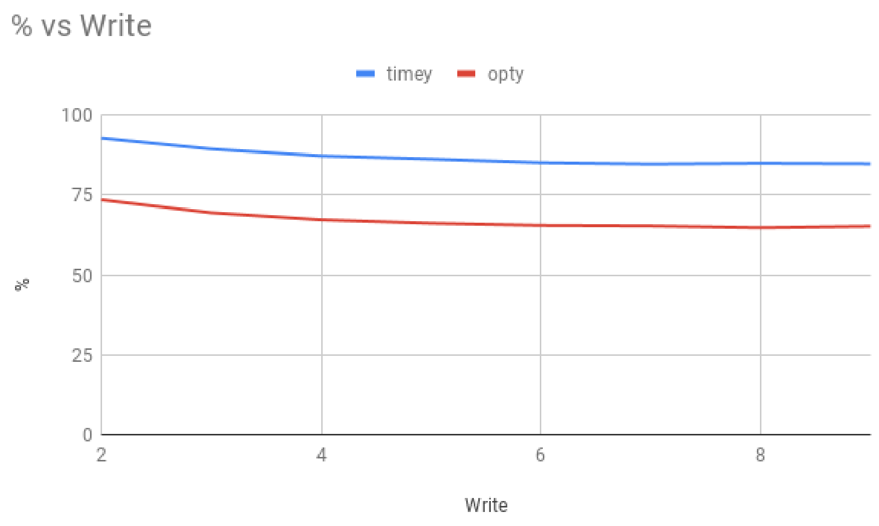
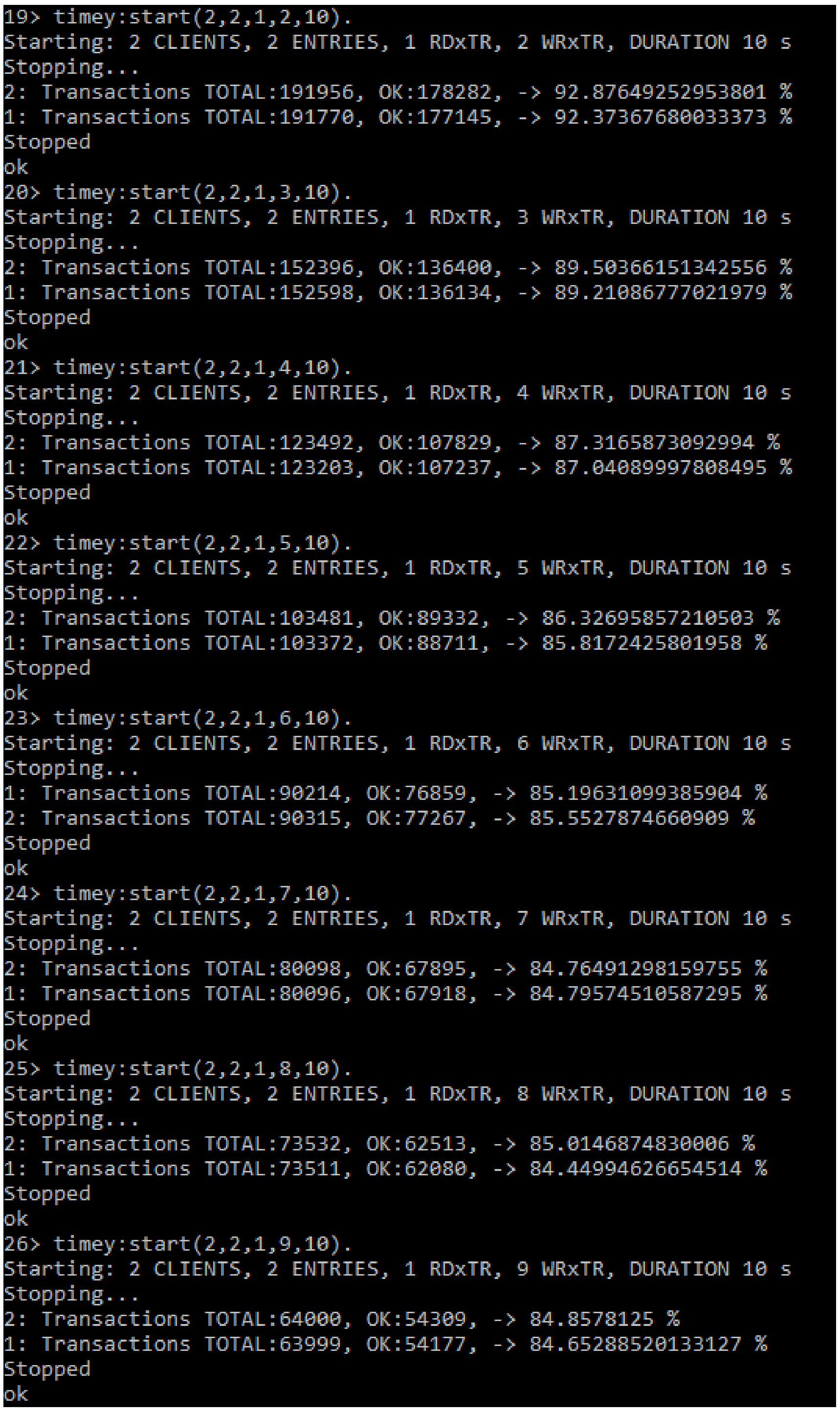


Figure 18: Different no of writes timey vs opty



```
19> timey:start(2,2,1,2,10).
Starting: 2 CLIENTS, 2 ENTRIES, 1 RDxTR, 2 WRxTR, DURATION 10 s
Stopping...
2: Transactions TOTAL:191956, OK:178282, -> 92.87649252953801 %
1: Transactions TOTAL:191770, OK:177145, -> 92.37367680033373 %
Stopped
ok
20> timey:start(2,2,1,3,10).
Starting: 2 CLIENTS, 2 ENTRIES, 1 RDxTR, 3 WRxTR, DURATION 10 s
Stopping...
2: Transactions TOTAL:152396, OK:136400, -> 89.50366151342556 %
1: Transactions TOTAL:152598, OK:136134, -> 89.21086777021979 %
Stopped
ok
21> timey:start(2,2,1,4,10).
Starting: 2 CLIENTS, 2 ENTRIES, 1 RDxTR, 4 WRxTR, DURATION 10 s
Stopping...
2: Transactions TOTAL:123492, OK:107829, -> 87.3165873092994 %
1: Transactions TOTAL:123203, OK:107237, -> 87.04089997808495 %
Stopped
ok
22> timey:start(2,2,1,5,10).
Starting: 2 CLIENTS, 2 ENTRIES, 1 RDxTR, 5 WRxTR, DURATION 10 s
Stopping...
2: Transactions TOTAL:103481, OK:89332, -> 86.32695857210503 %
1: Transactions TOTAL:103372, OK:88711, -> 85.8172425801958 %
Stopped
ok
23> timey:start(2,2,1,6,10).
Starting: 2 CLIENTS, 2 ENTRIES, 1 RDxTR, 6 WRxTR, DURATION 10 s
Stopping...
1: Transactions TOTAL:90214, OK:76859, -> 85.19631099385904 %
2: Transactions TOTAL:90315, OK:77267, -> 85.5527874660909 %
Stopped
ok
24> timey:start(2,2,1,7,10).
Starting: 2 CLIENTS, 2 ENTRIES, 1 RDxTR, 7 WRxTR, DURATION 10 s
Stopping...
2: Transactions TOTAL:80098, OK:67895, -> 84.76491298159755 %
1: Transactions TOTAL:80096, OK:67918, -> 84.79574510587295 %
Stopped
ok
25> timey:start(2,2,1,8,10).
Starting: 2 CLIENTS, 2 ENTRIES, 1 RDxTR, 8 WRxTR, DURATION 10 s
Stopping...
2: Transactions TOTAL:73532, OK:62513, -> 85.0146874830006 %
1: Transactions TOTAL:73511, OK:62080, -> 84.44994626654514 %
Stopped
ok
26> timey:start(2,2,1,9,10).
Starting: 2 CLIENTS, 2 ENTRIES, 1 RDxTR, 9 WRxTR, DURATION 10 s
Stopping...
2: Transactions TOTAL:64000, OK:54309, -> 84.8578125 %
1: Transactions TOTAL:63999, OK:54177, -> 84.65288520133127 %
Stopped
ok
```

Figure 19: Execution of timey with different no of writes

2.3.5 Diferent ratio between reads and writes

Following the same ratios that we select for the first test we obtain the resulst shown in the next table:

Reads	Ratio	Writes	Results C1 %	Results C2 %
10	10:0	0	100%	100%
9	9:1	1	94.56%	94.99%
8	8:2	2	89.98%	90.08%
7	7:3	3	86.84%	86.81%
6	6:4	4	84.89%	85.06%
5	5:5	5	83.76%	83.98%
4	4:6	6	81.87%	82%
3	3:7	7	77.49%	80.05%
2	2:8	8	77.87%	78.01%
1	1:9	9	84.02%	83.78%
0	0:10	10	99.99%	99.99%

Table 2: Diferent ratios for read and writes.

By plotting the values we can observe that the behavaior of this implementations have better performance, and also that the handle of the writes has more impact in the performace, in other words when there are more writes the performance is slightly lower compared with the same amount of reads, but even with that the performance is much better in this implementation.

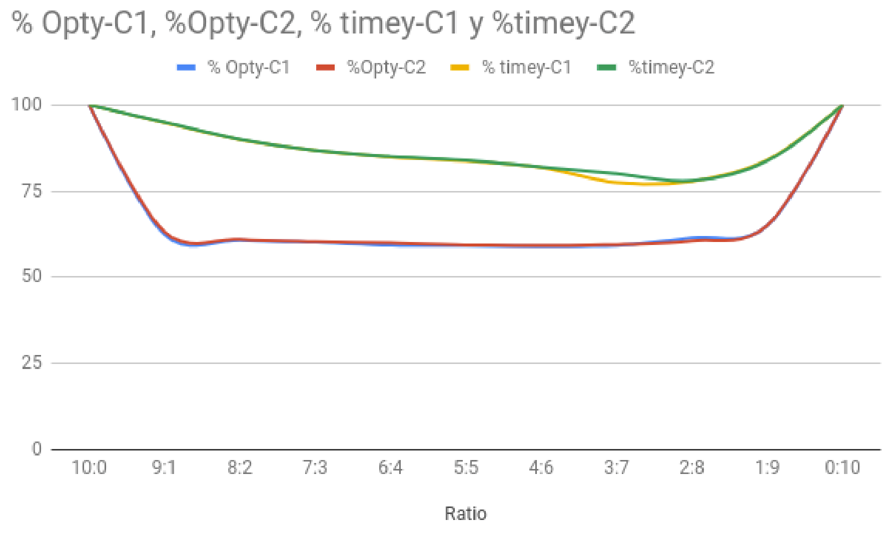
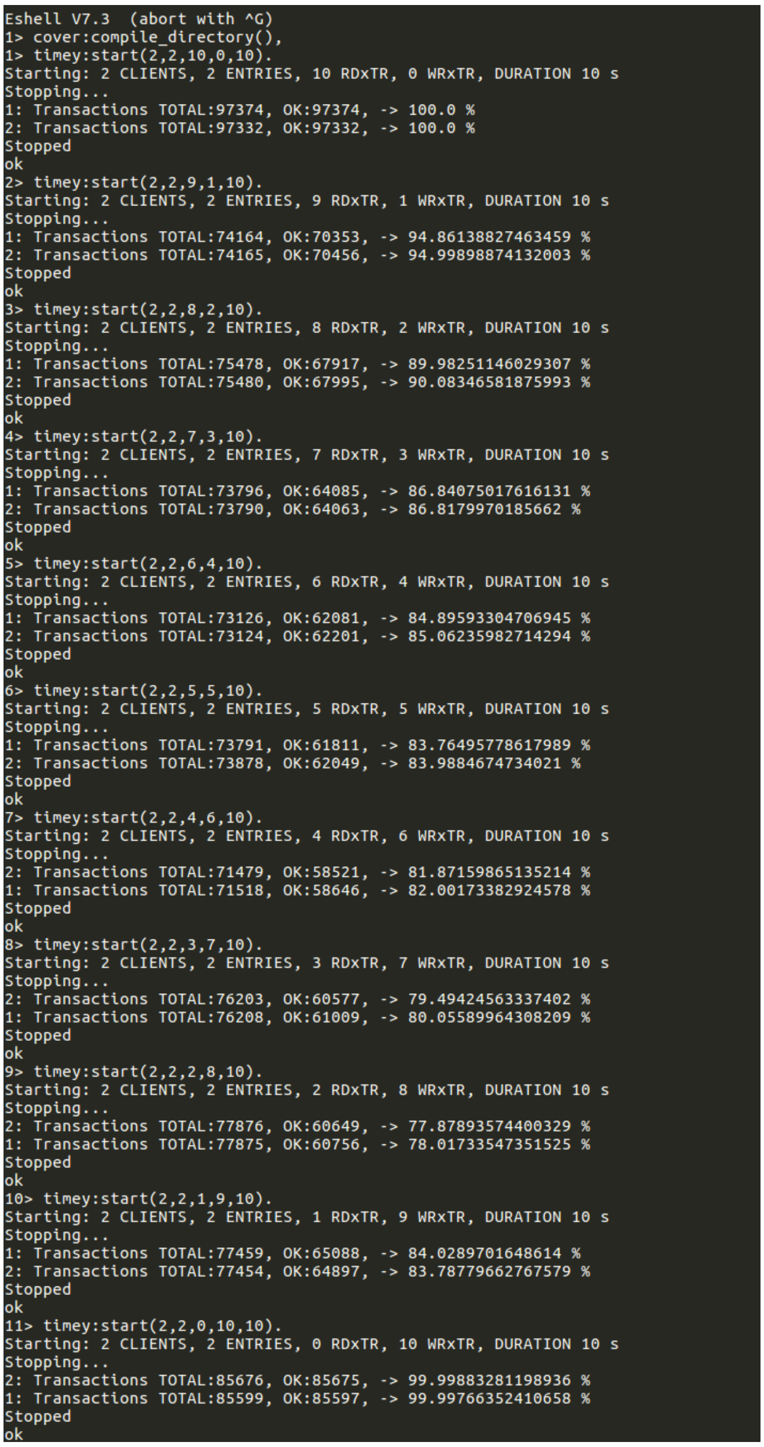


Figure 20: Different ratios timey vs opty graph.



```
Eshell V7.3  (abort with ^G)
1> cover:compile_directory(),
1> timey:start(2,2,10,0,10).
Starting: 2 CLIENTS, 2 ENTRIES, 10 RDxTR, 0 WRxTR, DURATION 10 s
Stopping...
1: Transactions TOTAL:97374, OK:97374, -> 100.0 %
2: Transactions TOTAL:97332, OK:97332, -> 100.0 %
Stopped
ok
2> timey:start(2,2,9,1,10).
Starting: 2 CLIENTS, 2 ENTRIES, 9 RDxTR, 1 WRxTR, DURATION 10 s
Stopping...
1: Transactions TOTAL:74164, OK:70353, -> 94.86138827463459 %
2: Transactions TOTAL:74165, OK:70456, -> 94.99898874132003 %
Stopped
ok
3> timey:start(2,2,8,2,10).
Starting: 2 CLIENTS, 2 ENTRIES, 8 RDxTR, 2 WRxTR, DURATION 10 s
Stopping...
1: Transactions TOTAL:75478, OK:67917, -> 89.98251146029307 %
2: Transactions TOTAL:75480, OK:67995, -> 90.08346581875993 %
Stopped
ok
4> timey:start(2,2,7,3,10).
Starting: 2 CLIENTS, 2 ENTRIES, 7 RDxTR, 3 WRxTR, DURATION 10 s
Stopping...
1: Transactions TOTAL:73796, OK:64085, -> 86.84075017616131 %
2: Transactions TOTAL:73790, OK:64063, -> 86.8179970185662 %
Stopped
ok
5> timey:start(2,2,6,4,10).
Starting: 2 CLIENTS, 2 ENTRIES, 6 RDxTR, 4 WRxTR, DURATION 10 s
Stopping...
1: Transactions TOTAL:73126, OK:62081, -> 84.89593304706945 %
2: Transactions TOTAL:73124, OK:62201, -> 85.06235982714294 %
Stopped
ok
6> timey:start(2,2,5,5,10).
Starting: 2 CLIENTS, 2 ENTRIES, 5 RDxTR, 5 WRxTR, DURATION 10 s
Stopping...
1: Transactions TOTAL:73791, OK:61811, -> 83.76495778617989 %
2: Transactions TOTAL:73878, OK:62049, -> 83.9884674734021 %
Stopped
ok
7> timey:start(2,2,4,6,10).
Starting: 2 CLIENTS, 2 ENTRIES, 4 RDxTR, 6 WRxTR, DURATION 10 s
Stopping...
2: Transactions TOTAL:71479, OK:58521, -> 81.87159865135214 %
1: Transactions TOTAL:71518, OK:58646, -> 82.00173382924578 %
Stopped
ok
8> timey:start(2,2,3,7,10).
Starting: 2 CLIENTS, 2 ENTRIES, 3 RDxTR, 7 WRxTR, DURATION 10 s
Stopping...
2: Transactions TOTAL:76203, OK:60577, -> 79.49424563337402 %
1: Transactions TOTAL:76208, OK:61009, -> 80.05589964308209 %
Stopped
ok
9> timey:start(2,2,2,8,10).
Starting: 2 CLIENTS, 2 ENTRIES, 2 RDxTR, 8 WRxTR, DURATION 10 s
Stopping...
2: Transactions TOTAL:77876, OK:60649, -> 77.87893574400329 %
1: Transactions TOTAL:77875, OK:60756, -> 78.01733547351525 %
Stopped
ok
10> timey:start(2,2,1,9,10).
Starting: 2 CLIENTS, 2 ENTRIES, 1 RDxTR, 9 WRxTR, DURATION 10 s
Stopping...
1: Transactions TOTAL:77459, OK:65088, -> 84.0289701648614 %
2: Transactions TOTAL:77454, OK:64897, -> 83.78779662767579 %
Stopped
ok
11> timey:start(2,2,0,10,10).
Starting: 2 CLIENTS, 2 ENTRIES, 0 RDxTR, 10 WRxTR, DURATION 10 s
Stopping...
2: Transactions TOTAL:85676, OK:85675, -> 99.99883281198936 %
1: Transactions TOTAL:85599, OK:85597, -> 99.99766352410658 %
Stopped
ok
```

Figure 21: Execution of timey with different ratios.

3 Open questions

I. What is the impact of each of these parameters on the success rate (i.e. percentage of committed transactions with respect to the total)? (hint: the impact can be better appreciated if represented graphically)

   This question is better answered along the experiments, and also the impact of each parameter shown in the respective graphs.

II. Is the success rate the same for the different clients?

   In general no, the difference is no too much to be considered, but each particular case is better seen on each experiment.

III. If we run this in a distributed Erlang network, where is the handler running?

   The handler process runs in the client machine, also when using timestamp ordering.

4 Personal opinion

Jonnatan Mendoza Escobar: This lab gives a good idea of the different methods that be used to implement transactional operations to database systems, is great to undestant a little beat of what is under the hood of concurrent databases engains, the lab is complete and certaly difficult, relate the tehory with the implementation was the hardest part for me.