Seminar Report: Opty

Diego Garcia-Olano, Elsa Mullor

December 27, 2013

1 Introduction

This seminar is an example of an Optimistic Concurrency Control with backward validation. We implement in Erlang a transaction Server, an Updatable data structure and processes that will access the latter concurrently.

2 Work done

The source code is in the attached folder called "Opty_code".
It contains:

client.erl
enty.erl
handler.erl
opty.erl
optyclient.erl
optyserver.erl
server.erl
store.erl
validator.erl

When an experiment requires a particular part of the code, this one is left commented in the code. When it should be uncommented, it will be specified in the third part of this report.

3 Experiments & Open questions

(i) Different number of concurrent clients in the system.

We decided to fix the other parameters (number of entries=3, number of updates per transactions=2, duration=10s).

```
1> opty:start(1,3,2,10).
Starting: 1 CLIENTS, 3 ENTRIES, 2 UPDATES PER TRANSACTION,
DURATION 10 s
Stopping...
1: Transactions TOTAL:284464, OK:284464, -> 100.0 %
stop
```

With only one client in 10s there are more than 280 000 transactions and all of them are successful.

```
2> opty:start(5,3,2,10).
```

```
Starting: 5 CLIENTS, 3 ENTRIES, 2 UPDATES PER TRANSACTION,
DURATION 10 s
Stopping...
1: Transactions TOTAL:77359, OK:26230, -> 33.90684988172029 %
2: Transactions TOTAL:77444, OK:26503, -> 34.222147616342134 %
3: Transactions TOTAL:77498, OK:26417, -> 34.08733128596867 %
4: Transactions TOTAL:77390, OK:26567, -> 34.32872464142654 %
5: Transactions TOTAL:77602, OK:26540, -> 34.200149480683486 %
Stop
```

With 5 clients the number of transactions is nearly divided by 4 and only a third of them are successful. The performance of the system has decreased quickly.

```
3> opty:start(10,3,2,10).
Starting: 10 CLIENTS, 3 ENTRIES, 2 UPDATES PER TRANSACTION,
DURATION 10 s
Stopping...
1: Transactions TOTAL:44470, OK:10469, -> 23.541713514729032 %
2: Transactions TOTAL:43929, OK:9810, -> 22.331489448883424 %
3: Transactions TOTAL:44175, OK:10029, -> 22.70288624787776 %
4: Transactions TOTAL:44369, OK:10181, -> 22.946201176497105 %
5: Transactions TOTAL:44019, OK:9911, -> 22.515277493809492 %
6: Transactions TOTAL:44360, OK:10128, -> 22.831379621280433 %
7: Transactions TOTAL:44098, OK:9917, -> 22.488548233479975 %
8: Transactions TOTAL:44228, OK:10100, -> 22.836212354164783 %
9: Transactions TOTAL:44209, OK:10024, -> 22.67411613019973 %
10: Transactions TOTAL:43831, OK:9759, -> 22.265063539504006 %
stop
```

When at this point we double the number of clients (10) the performance keep decreasing but less sharply: around 44 000 of transactions (compared to 77000), a little bit more than the half, and 22% of success (10 points less).

```
4> opty:start(100,3,2,10).
Starting: 100 CLIENTS, 3 ENTRIES, 2 UPDATES PER TRANSACTION,
DURATION 10 s
Stopping...
1: Transactions TOTAL:3800, OK:99, -> 2.6052631578947367 %
2: Transactions TOTAL:3806, OK:104, -> 2.7325275880189177 %
3: Transactions TOTAL:3816, OK:104, -> 2.7253668763102725 %
...
98: Transactions TOTAL:3872, OK:116, -> 2.9958677685950414 %
99: Transactions TOTAL:3880, OK:107, -> 2.7577319587628866 %
100: Transactions TOTAL:3907, OK:141, -> 3.608907089838751 %
```

When increasing a lot the number of clients (100) there are nearly 4000 transactions done in 10 s with less than 3% of success. The number of transaction is approximately divided by 10 so as the percentage of success.

```
5> opty:start(1000,3,2,10).
Starting: 1000 CLIENTS, 3 ENTRIES, 2 UPDATES PER TRANSACTION,
DURATION 10 s
Stopping...
1: Transactions TOTAL:158, OK:2, -> 1.2658227848101267 %
2: Transactions TOTAL:158, OK:1, -> 0.6329113924050633 %
3: Transactions TOTAL:160, OK:1, -> 0.625 %
4: Transactions TOTAL:159, OK:4, -> 2.5157232704402515 %
5: Transactions TOTAL:160, OK:0, -> 0.0 %
6: Transactions TOTAL:161, OK:2, -> 1.2422360248447204 %
7: Transactions TOTAL:162, OK:2, -> 1.2345679012345678 %
8: Transactions TOTAL:162, OK:7, -> 4.320987654320987 %
9: Transactions TOTAL:166, OK:1, -> 0.6024096385542169 %
10: Transactions TOTAL:163, OK:1, -> 0.6134969325153374 %
11: Transactions TOTAL:164, OK:0, -> 0.0 %
12: Transactions TOTAL:164, OK:1, -> 0.6097560975609756 %
13: Transactions TOTAL:167, OK:3, -> 1.7964071856287425 %
14: Transactions TOTAL:165, OK:0, -> 0.0 %
15: Transactions TOTAL:166, OK:0, -> 0.0 %
16: Transactions TOTAL:166, OK:0, -> 0.0 %
17: Transactions TOTAL:166, OK:0, -> 0.0 %
18: Transactions TOTAL:167, OK:1, -> 0.5988023952095808 %
19: Transactions TOTAL:168, OK:0, -> 0.0 %
20: Transactions TOTAL:168, OK:0, -> 0.0 %
21: Transactions TOTAL:168, OK:0, -> 0.0 %
22: Transactions TOTAL:169, OK:0, -> 0.0 %
23: Transactions TOTAL:169, OK:0, -> 0.0 %
24: Transactions TOTAL:170, OK:0, -> 0.0 %
25: Transactions TOTAL:170, OK:1, -> 0.5882352941176471 %
...
976: Transactions TOTAL:716, OK:21, -> 2.9329608938547485 %
```

```
977: Transactions TOTAL:709, OK:12, -> 1.692524682651622 %
978: Transactions TOTAL:690, OK:3, -> 0.43478260869565216 %
979: Transactions TOTAL:709, OK:5, -> 0.7052186177715092 %
980: Transactions TOTAL:698, OK:5, -> 0.7163323782234957 %
981: Transactions TOTAL:702, OK:13, -> 1.8518518518518519 %
982: Transactions TOTAL:696, OK:4, -> 0.5747126436781609 %
983: Transactions TOTAL:719, OK:4, -> 0.5563282336578581 %
984: Transactions TOTAL:700, OK:11, -> 1.5714285714285714 %
985: Transactions TOTAL:714, OK:8, -> 1.1204481792717087 %
986: Transactions TOTAL:711, OK:6, -> 0.8438818565400844 %
987: Transactions TOTAL:695, OK:5, -> 0.7194244604316546 %
988: Transactions TOTAL:717, OK:16, -> 2.2315202231520224 %
989: Transactions TOTAL:707, OK:9, -> 1.272984441301273 %
990: Transactions TOTAL:699, OK:9, -> 1.2875536480686696 %
991: Transactions TOTAL:711, OK:10, -> 1.4064697609001406 %
992: Transactions TOTAL:726, OK:16, -> 2.203856749311295 %
993: Transactions TOTAL:715, OK:14, -> 1.9580419580419581 %
994: Transactions TOTAL:698, OK:6, -> 0.8595988538681948 %
995: Transactions TOTAL:707, OK:12, -> 1.6973125884016973 %
996: Transactions TOTAL:713, OK:16, -> 2.244039270687237 %
997: Transactions TOTAL:717, OK:2, -> 0.2789400278940028 %
998: Transactions TOTAL:818, OK:45, -> 5.50122249388753 %
999: Transactions TOTAL:819, OK:65, -> 7.936507936507937 %
1000: Transactions TOTAL:846, OK:79, -> 9.33806146572104 %
```

With 1000 clients the number of transaction is very low so as the percentage of success. We can point out that the latest clients have more success than the first ones. The results change more from a client to another one, the number of transaction is between 100 and 900 and the rate of success between 0% and 10%. Compared to the experiments done before the system is less stable: there is more dispersion of the success rate for the different clients.

To conclude we can say that the number of clients (all other parameters being equals) impact the number of transaction and the success rate. Besides when multiplying the number of clients by a constant we can observe that the number of transactions is nearly divided by this constant so we could guess the relation follows:

Number of Transactions £ $\frac{1}{Number\ of\ Clients}$.

(ii) Different number of entries in the store.

We try different number of entries. The other parameters are fixed.

```
6> opty:start(5,1,2,10).
Starting: 5 CLIENTS, 1 ENTRIES, 2 UPDATES PER TRANSACTION
DURATION 10 s
Stopping...
1: Transactions TOTAL:93945, OK:25642, -> 27.29469370376284 %
2: Transactions TOTAL:93224, OK:25096, -> 26.92010641036643 %
3: Transactions TOTAL:94456, OK:26178, -> 27.71449140340476 %
4: Transactions TOTAL:93343, OK:25338, -> 27.145045691696218 %
5: Transactions TOTAL:93774, OK:25204, -> 26.87738605583637 %
Stop
```

With 5 clients and only one entry there are more than 90 000 transactions and around 27% of success. With 5 clients and 3 entries we had 77 000 transactions and more than 34% of success.

```
7> opty:start(5,5,2,10).
Starting: 5 CLIENTS, 5 ENTRIES, 2 UPDATES PER TRANSACTION,
DURATION 10 s
Stopping...
1: Transactions TOTAL:73260, OK:31920, -> 43.57084357084357 %
2: Transactions TOTAL:73043, OK:31813, -> 43.55379707843325 %
3: Transactions TOTAL:73224, OK:31897, -> 43.56085436468917 %
4: Transactions TOTAL:73106, OK:32110, -> 43.922523459086804 %
```

```
5: Transactions TOTAL:73363, OK:31959, -> 43.562831400024535 %
Stop
```

If there are as much entries as clients (5 for each) the number of transactions becomes 73 000 which is not very different from the first experiment. But the success rate increases a lot (more than 43%).

```
8> opty:start(5,10,2,10).
Starting: 5 CLIENTS, 10 ENTRIES, 2 UPDATES PER TRANSACTION,
DURATION 10 s
Stopping...
1: Transactions TOTAL:68803, OK:41267, -> 59.97848931005915 %
2: Transactions TOTAL:68550, OK:40858, -> 59.60320933625091 %
3: Transactions TOTAL:68522, OK:41019, -> 59.862525904089196 %
4: Transactions TOTAL:67843, OK:40660, -> 59.93249119290126 %
5: Transactions TOTAL:68379, OK:40867, -> 59.76542505740066 %
Stop
```

With 10 entries the total number of transaction keeps decreasing slowly (now equal to 68 000) and the success rate keeps increasing (nearly 60%).

```
9> opty:start(5,100,2,10).
Starting: 5 CLIENTS, 100 ENTRIES, 2 UPDATES PER TRANSACTION,
DURATION 10 s
Stopping...
1: Transactions TOTAL:62920, OK:59116, -> 93.95422759059123 %
2: Transactions TOTAL:63128, OK:59301, -> 93.93771385122291 %
3: Transactions TOTAL:62853, OK:59143, -> 94.09733823365632 %
4: Transactions TOTAL:62786, OK:58926, -> 93.85213264103463 %
5: Transactions TOTAL:62966, OK:59155, -> 93.94752723692152 %
Stop
```

```
10> opty:start(5,1000,2,10).
Starting: 5 CLIENTS, 1000 ENTRIES, 2 UPDATES PER TRANSACTION,
DURATION 10 s
Stopping...
1: Transactions TOTAL:54807, OK:54574, -> 99.57487182294233 %
2: Transactions TOTAL:54861, OK:54602, -> 99.52789777802082 %
3: Transactions TOTAL:54834, OK:54594, -> 99.56231535178904 %
4: Transactions TOTAL:54880, OK:54637, -> 99.55721574344024 %
5: Transactions TOTAL:54792, OK:54543, -> 99.5455540954884 %
stop
```

When increasing dramatically the number of entries, we can notice that the number of transactions is slowly impacted (reduced to 62 000 with a hundred of entries and to 54 000 with a thousand). The success rate, on the other hand, changes a lot: with a hundred entries it is already equal to 93% and with a thousand entries to 99%.

To conclude we can say that the number of entries impacts a lot the success rate. The more entries there are, the better the rate will be. That can be explained, as the more entry there are, the less concurrency there will be between clients.
It also impacts a little the total number of transactions, as the store is bigger: the total number of transactions decrease as the number of entries increase.

(iii) Different number of write operations per transaction.

We change the number of updates per transactions, all other parameters being equals.

```
14> opty:start(5,10,1,10).
Starting: 5 CLIENTS, 10 ENTRIES, 1 UPDATES PER TRANSACTION,
DURATION 10 s
Stopping...
```

```
1: Transactions TOTAL:97195, OK:80283, -> 82.59992797983435 %
2: Transactions TOTAL:97245, OK:80340, -> 82.61607280579979 %
3: Transactions TOTAL:97299, OK:80381, -> 82.61235983925837 %
4: Transactions TOTAL:97128, OK:80197, -> 82.5683633967548 %
5: Transactions TOTAL:97202, OK:80308, -> 82.61969918314438 %
Stop
```

With one update per transaction the total number of transactions is around 97 000 and the success rate around 83%

```
15> opty:start(5,10,10,10).
Starting: 5 CLIENTS, 10 ENTRIES, 10 UPDATES PER TRANSACTION,
DURATION 10 s
Stopping...
1: Transactions TOTAL:24535, OK:5524, -> 22.514774811493783 %
2: Transactions TOTAL:24651, OK:5551, -> 22.518356253296012 %
3: Transactions TOTAL:24604, OK:5636, -> 22.90684441554219 %
4: Transactions TOTAL:24300, OK:5112, -> 21.037037037037038 %
5: Transactions TOTAL:24331, OK:5312, -> 21.832230487855 %
```

If we multiply this number by 10 we now have less than 25 000 of transactions with a success rate of 22%. Both have decreased a lot.

```
16> opty:start(5,10,100,10).
Starting: 5 CLIENTS, 10 ENTRIES, 100 UPDATES PER TRANSACTION,
DURATION 10 s
Stopping...
1: Transactions TOTAL:5878, OK:1542, -> 26.23341272541681 %
2: Transactions TOTAL:5797, OK:1449, -> 24.99568742452993 %
3: Transactions TOTAL:5765, OK:1252, -> 21.717259323503903 %
4: Transactions TOTAL:5871, OK:1611, -> 27.43995912110373 %
5: Transactions TOTAL:5795, OK:1165, -> 20.103537532355478 %
Stop
```

With 100 of updates per transaction the total number of transactions keeps decreasing significantly (less than 6000 this time) but the success rate isn't much impacted (between 20% and 28%). But we can point out that this number varies more from a client to another one.
The fact that the more updates per transaction there are, the less transactions are done in 10s can be explained as every transaction will require more time.

We want to observe how the number of updates per transaction impacts the system along with the number of entries.

```
18> opty:start(5,5,1,10).
Starting: 5 CLIENTS, 5 ENTRIES, 1 UPDATES PER TRANSACTION,
DURATION 10 s
Stopping...
1: Transactions TOTAL:100732, OK:54075, -> 53.6820474129373 %
2: Transactions TOTAL:100727, OK:53500, -> 53.11386222164862 %
3: Transactions TOTAL:101397, OK:54083, -> 53.337869956704836 %
4: Transactions TOTAL:101075, OK:54246, -> 53.66905763047242 %
5: Transactions TOTAL:101401, OK:54674, -> 53.91860040828 %
stop
19> opty:start(5,10,2,10).
Starting: 5 CLIENTS, 10 ENTRIES, 2 UPDATES PER TRANSACTION,
DURATION 10 s
Stopping...
1: Transactions TOTAL:68609, OK:40745, -> 59.387252401288464 %
2: Transactions TOTAL:68730, OK:40985, -> 59.63189291430234 %
3: Transactions TOTAL:68828, OK:41184, -> 59.83611320974022 %
4: Transactions TOTAL:68780, OK:41079, -> 59.725210817097995 %
5: Transactions TOTAL:68807, OK:40914, -> 59.461973345735174 %
stop
20> opty:start(5,100,20,10).
Starting: 5 CLIENTS, 100 ENTRIES, 20 UPDATES PER TRANSACTION,
DURATION 10 s
Stopping...
1: Transactions TOTAL:10131, OK:2572, -> 25.387424735958938 %
2: Transactions TOTAL:10114, OK:2439, -> 24.11508799683607 %
```

```
3: Transactions TOTAL:10107, OK:2419, -> 23.933907193034532 %
4: Transactions TOTAL:10122, OK:2438, -> 24.086148982414542 %
5: Transactions TOTAL:10141, OK:2476, -> 24.415738092890248 %
stop
```

If we increase the number of entries while increasing the number of updates per transactions, the success rate is less impacted than before. Still even if we maintain a factor of 1/5 between them, the performance finally decreases when there are many entries (100).

To conclude we can say that the number of write operations per transactions impacts the success rate but depending on the number of entries present in the store. But globally when the number of write operations per transactions increase, the success rate decrease.

(iv) Different duration of the transactions.

We uncomment out the initial code in do_transactions client.erl to add delay between the read/write commits.

```
timer:sleep(1),
With Read/write delay of 1ms
21> opty:start(5,3,2,10).
Starting: 5 CLIENTS, 3 ENTRIES, 2 UPDATES PER TRANSACTION,
DURATION 10 s
Stopping...
1: Transactions TOTAL:4542, OK:1326, -> 29.194187582562748 %
2: Transactions TOTAL:4540, OK:1358, -> 29.911894273127754 %
3: Transactions TOTAL:4543, OK:1256, -> 27.646929341844597 %
4: Transactions TOTAL:4543, OK:1240, -> 27.294739159145937 %
5: Transactions TOTAL:4541, OK:1276, -> 28.09953754679586 %
stop
```

With a delay of 1ms the performance of the system is already decreased. In 10 s there are 400 transactions (compared to 77 000 with no delay) and around 29% of success only (compared to 34%)

```
timer:sleep(10),
With Read/write delay of 10ms
22> opty:start(5,3,2,10).
Starting: 5 CLIENTS, 3 ENTRIES, 2 UPDATES PER TRANSACTION,
DURATION 10 s
Stopping...
1: Transactions TOTAL:888, OK:305, -> 34.346846846846844 %
2: Transactions TOTAL:888, OK:337, -> 37.950450450450454 %
3: Transactions TOTAL:889, OK:245, -> 27.559055118110237 %
4: Transactions TOTAL:890, OK:270, -> 30.337078651685392 %
5: Transactions TOTAL:890, OK:279, -> 31.348314606741575 %
stop
```

With 10ms of delay the total number of transaction is dramatically decreased but the success rate not. On the contrary this time it has slightly been improved.

```
timer:sleep(100),
With Read/write delay of 100ms
23> opty:start(5,3,2,10).
Starting: 5 CLIENTS, 3 ENTRIES, 2 UPDATES PER TRANSACTION,
DURATION 10 s
Stopping...
1: Transactions TOTAL:99, OK:10, -> 10.1010101010101 %
2: Transactions TOTAL:100, OK:34, -> 34.0 %
3: Transactions TOTAL:101, OK:41, -> 40.59405940594059 %
4: Transactions TOTAL:102, OK:47, -> 46.07843137254902 %
5: Transactions TOTAL:103, OK:27, -> 26.21359223300971 %
stop
```

```
timer:sleep(200),
With Read/write delay of 200ms
24> opty:start(5,3,2,10).
Starting: 5 CLIENTS, 3 ENTRIES, 2 UPDATES PER TRANSACTION,
DURATION 10 s
Stopping...
1: Transactions TOTAL:50, OK:12, -> 24.0 %
2: Transactions TOTAL:51, OK:11, -> 21.568627450980394 %
3: Transactions TOTAL:52, OK:20, -> 38.46153846153846 %
4: Transactions TOTAL:53, OK:18, -> 33.9622641509434 %
5: Transactions TOTAL:53, OK:17, -> 32.075471698113205 %
```

```
timer:sleep(1000),
With Read/write delay of 1000ms
25> opty:start(5,3,2,10).
Starting: 5 CLIENTS, 3 ENTRIES, 2 UPDATES PER TRANSACTION,
DURATION 10 s
Stopping...
1: Transactions TOTAL:10, OK:3, -> 30.0 %
2: Transactions TOTAL:10, OK:1, -> 10.0 %
3: Transactions TOTAL:11, OK:6, -> 54.54545454545455 %
4: Transactions TOTAL:12, OK:4, -> 33.333333333333336 %
5: Transactions TOTAL:13, OK:6, -> 46.15384615384615 %
stop
```

If we keep increasing the delay (to 100ms, 200ms and 1000ms) the number of transactions becomes very low (100, 50 and 10 transactions in 10s). The success rate, on the other hand, does not decrease significantly. As the total number of transaction is very low, the rate is not very robust (it changes a lot from a client to another one), but it stays around 30% in most cases.

To conclude, adding a delay before Read and Write commits reduce the total number of transactions, but it doesn't impact the success rate of transactions.

(v) Different ratio of read and write operations per transaction.

We need to modify the code in clients.erl to change the ratio.

In do_transactions(Name, Entries, Updates, Server, Handler, Total, Ok, N),

```
Ref = make_ref(),
Num = random:uniform(Entries),
Handler ! {read,Ref, Num},
Value = receiveValue(Ref),
Handler ! {write, Num, Value+1},

Ref2 = make_ref(),
Num2 = random:uniform(Entries),
Handler ! {read, Ref2, Num2},
```

We have 1 write per 2 reads.

```
26> opty:start(5,3,2,10).
Starting: 5 CLIENTS, 3 ENTRIES, 2 UPDATES PER TRANSACTION,
DURATION 10 s
Stopping...
1: Transactions TOTAL:4101, OK:1787, -> 43.574737868812484 %
2: Transactions TOTAL:4213, OK:1748, -> 41.49062425824828 %
3: Transactions TOTAL:4187, OK:1692, -> 40.41079531884404 %
4: Transactions TOTAL:4248, OK:1853, -> 43.620527306967986 %
5: Transactions TOTAL:4102, OK:1654, -> 40.321794246708926 %
Stop
```

Compared with the normal case (2>) where we had 77 000 transactions with a 34% of success. In this case the total number of transactions is reduced a lot (nearly divided by 20).

The success rate (around 40%) is better than in the normal case. We can therefore conclude that read operations are more successful than write operations.

```
Ref = make_ref(),
Num = random:uniform(Entries),
Handler ! {read, Ref, Num},
Value = receiveValue(Ref),
Handler ! {write, Num, Value+1},
Num2 = random:uniform(Entries),
Handler ! {write, Num2, Value+1},
```

We have 2 writes per 1 read:

```
27> opty:start(5,3,2,10).
Starting: 5 CLIENTS, 3 ENTRIES, 2 UPDATES PER TRANSACTION,
DURATION 10 s
Stopping...
1: Transactions TOTAL:48319, OK:18808, -> 38.92464661934229 %
2: Transactions TOTAL:48657, OK:19005, -> 39.05912818299525 %
3: Transactions TOTAL:48706, OK:18995, -> 38.9993019340533 %
4: Transactions TOTAL:47710, OK:18665, -> 39.12177740515615 %
5: Transactions TOTAL:48701, OK:19282, -> 39.59261616804583 %
stop
```

In this case the total number of transaction (48 000) is much higher than in the previous experiment (4 000) even if smaller than the normal case (77 000).
The success rate is this time around 39% (2 points less than the previous case).

We can conclude that increasing the number of read operations over writes operations reduce a lot the total number of transactions while the contrary is not true. The read operations slow the system more than the write operations.

The success rate is comparable in both case and with the normal case, even if slightly higher when there are more read operations.

(vi) Different percentage of accessed entries with respect to the total number of entries.

For this experiment, when a Client does a transaction we need to allow him to only access a randomly generate subset of the available entries.
To do so we modify the code in client.erl:

```
do_transactions(Name, Entries, Updates, Server, Handler, Total, Ok, N) ->
%io:format("~w: R/W: TOTAL ~w, OK ~w, N ~w~n", [Name, Total, Ok, N]),
Ref = make_ref(),

%only access random subset of entries to transaction
Percentage = 0.5,
RandomSubset = random:uniform(round(Percentage * Entries)),
Num = RandomSubset,
Handler ! {read, Ref, Num},
Value = receiveValue(Ref),
Handler ! {write, Num, Value+1},
```

If we run the following without any change as a baseline (ie, percentage = 1), we get

```
16> opty:start(5,10,2,10).
Starting: 5 CLIENTS, 10 ENTRIES, 2 UPDATES PER TRANSACTION,
DURATION 10 s
Stopping...
1: Transactions TOTAL:47301, OK:28758, -> 60.797868966829455 %
2: Transactions TOTAL:47472, OK:28831, -> 60.732642399730366 %
```

```
3: Transactions TOTAL:47112, OK:28788, -> 61.105450840550176 %
4: Transactions TOTAL:47980, OK:29396, -> 61.267194664443515 %
5: Transactions TOTAL:47537, OK:28948, -> 60.89572333129983 %
Stop
```

With percentage = .1,

```
20> opty:start(5,10,2,10).
Starting: 5 CLIENTS, 10 ENTRIES, 2 UPDATES PER TRANSACTION,
DURATION 10 s
Stopping...
1: Transactions TOTAL:60959, OK:24132, -> 39.58726357059663 %
2: Transactions TOTAL:60620, OK:23796, -> 39.25437149455625 %
3: Transactions TOTAL:59942, OK:24556, -> 40.96626739181208 %
4: Transactions TOTAL:61726, OK:25680, -> 41.60321420471114 %
5: Transactions TOTAL:60790, OK:22141, -> 36.42210889949005 %
```

When the client can only access 10% of the entries, the total number of transactions increase as a transaction is faster, but the success rate decreases to 60% to around 39%. Besides there is more heterogeneity in the success rate of the different clients.

With percentage = .3,

```
22> opty:start(5,10,2,10).
Starting: 5 CLIENTS, 10 ENTRIES, 2 UPDATES PER TRANSACTION,
DURATION 10 s
Stopping...
1: Transactions TOTAL:51357, OK:22728, -> 44.254921432326654 %
2: Transactions TOTAL:51761, OK:21975, -> 42.45474391916694 %
3: Transactions TOTAL:52539, OK:23359, -> 44.46030567768705 %
4: Transactions TOTAL:51891, OK:22582, -> 43.51814380142992 %
5: Transactions TOTAL:52035, OK:22771, -> 43.760930143172864 %
```

With 30% of the entries, the total number of transaction is less than in the previous case, but the success rate of transaction is slightly better. The dispersion of the success rate is still higher than in the normal case but less remarkable than the previous experiment.

With percentage = .5,

```
24> opty:start(5,10,2,10).
Starting: 5 CLIENTS, 10 ENTRIES, 2 UPDATES PER TRANSACTION,
DURATION 10 s
Stopping...
1: Transactions TOTAL:51056, OK:25919, -> 50.765825759949855 %
2: Transactions TOTAL:51026, OK:25392, -> 49.76286598988751 %
3: Transactions TOTAL:49882, OK:24950, -> 50.018042580489954 %
4: Transactions TOTAL:51844, OK:26226, -> 50.586374508139805 %
5: Transactions TOTAL:49462, OK:24667, -> 49.8706077392746 %
```

With 50% of the entries the total number of transactions hasn’t changed compared to 30%. The success rate keeps increasing and the dispersion of results for clients keeps decreasing.

with percentage = .8,

```
26> opty:start(5,10,2,10).
Starting: 5 CLIENTS, 10 ENTRIES, 2 UPDATES PER TRANSACTION,
DURATION 10 s
Stopping...
1: Transactions TOTAL:49146, OK:29339, -> 59.69763561632686 %
2: Transactions TOTAL:48956, OK:29311, -> 59.8721300759866 %
3: Transactions TOTAL:48770, OK:29239, -> 59.95283986057002 %
4: Transactions TOTAL:48226, OK:29101, -> 60.342968523203254 %
5: Transactions TOTAL:48490, OK:29280, -> 60.38358424417406 %
```

With 60% of the entries the success rate is close to the normal case. The dispersion of the results (0.7 pts) for each client is also close to the normal case (0.6 pts) and less than the previous case (0.8 pts).

The total number of transactions has also decreased.

We can conclude that when the clients only have access to a ratio of the store, the transactions are faster (so the total number of transaction in 10s is higher).
But the success rate is smaller and varies more between clients. The smaller the percentage of accessible entries is, the smaller the success rate will be.

Experiment: Split the opty module in two parts. If we run this in a distributed Erlang network, where is the handler running?

For this section we create **optyclient.erl** and **optyserver.erl** modules which contain the functionality for each. The server will set the number of entries, updates and time and then wait for connections from clients for a given period of time, sending its internal Entries to the clients, before timing out and telling the clients to stop.

The clients which are started in **optyclient.erl** will need a reference to the **optyserver** which itself must be instantiated first and additionally to the internal server within the optyserver class. Then when the clients are ready, they'll send an "open" message as usual to the Server, and the functionality will proceed as normal, except that the clients will be terminated once they receive a message from the server telling them to stop (ie, that the time of the experiment is done).

To do this we'll instantiate the server in its own window via the following command:

```
erl -name optys@127.0.0.1 -setcookie secret
```

And then we'll start it as follows with 5 entries, 2 updates per transaction and 10 seconds:

```
optyserver:start( 5, 2, 10 ).
```

Then in a different window, we'll instantiate the clients.erl process with the following:

```
erl -name client@127.0.0.1 -setcookie secret
```

And then we'll need to pass in the running server PID to the optyclient's start function, along with number of clients we want (10).

```
optyclient:start(10, {optys,'optys@127.0.0.1'},{s,'optys@127.0.0.1'}).
```

In action:

```
$ erl -name optys@127.0.0.1 -setcookie secret
(optys@127.0.0.1)1> c(optyserver).
{ok,optyserver}
(optys@127.0.0.1)2> optyserver:start( 5, 2, 10 ).
```

```
Opty Server started with 5 Entries, 2 Updates for Time: 10 s
optys In WaitForClients with Entries: 5, Updates: 2, Time: 10, Clients: []
Received Client request from <9427.38.0>
optys In WaitForClients with Entries: 5, Updates: 2, Time: 10, Clients: [<9427.38.0>]
Time up so return [<9427.38.0>]
Stopping...send [<9427.38.0>] to stopClients
```

```
$ erl -name client@127.0.0.1 -setcookie secret
(client@127.0.0.1)1> optyclient:start(10, {optys,'optys@127.0.0.1'},{s,'optys@127.0.0.1'}).
Client Sending clientrequest to Server: {optys,'optys@127.0.0.1'}
Client waiting for response from Server
Server sent ready message so create clients and proceed
Starting: 10 CLIENTS, 5 ENTRIES, 2 UPDATES PER TRANSACTION
Client receives Server Stop notice...
1: Transactions TOTAL:7566, OK:2370, -> 31.324345757335447 %
2: Transactions TOTAL:7538, OK:2389, -> 31.692756699389758 %
3: Transactions TOTAL:7576, OK:2363, -> 31.190601900739175 %
4: Transactions TOTAL:7567, OK:2466, -> 32.588872736883836 %
5: Transactions TOTAL:7551, OK:2359, -> 31.240895245662827 %
6: Transactions TOTAL:7503, OK:2328, -> 31.027588964414235 %
7: Transactions TOTAL:7583, OK:2430, -> 32.04536463141237 %
8: Transactions TOTAL:7589, OK:2450, -> 32.283568322572144 %
9: Transactions TOTAL:7549, OK:2357, -> 31.222678500463637 %
10: Transactions TOTAL:7562, OK:2365, -> 31.27479502777043 %
ok
```

In our non-distributed version of opty this simulation returns the following:

```
2> opty:start(10,5,2,10).
Starting: 10 CLIENTS, 5 ENTRIES, 2 UPDATES PER TRANSACTION,
DURATION 10 s
Stopping...
1: Transactions TOTAL:25674, OK:9312, -> 36.27015657863987 %
2: Transactions TOTAL:27051, OK:9846, -> 36.39791504935123 %
3: Transactions TOTAL:26808, OK:9687, -> 36.134735899731425 %
4: Transactions TOTAL:26776, OK:9880, -> 36.898715267403645 %
5: Transactions TOTAL:26124, OK:9498, -> 36.35737253100597 %
6: Transactions TOTAL:26341, OK:9533, -> 36.19072928134847 %
7: Transactions TOTAL:26049, OK:9436, -> 36.22403931053015 %
8: Transactions TOTAL:26749, OK:9855, -> 36.84249878500131 %
9: Transactions TOTAL:26251, OK:9960, -> 37.941411755742635 %
10: Transactions TOTAL:26244, OK:9355, -> 35.6462429507697 %
stop
```

In a distributed version, the optyserver process kicks off first and creates the server, store, and validator, and then once the optyclient tells the optyserver its around, and the server sends a ready request back, the optyclient then creates the clients themselves which each have their own handlers that interact with the clients themselves and the store (entries) and validator of the optyserver process. So the handler runs on the client side.

4 Personal Opinion

The seminar was interesting and the support was giving enough help to implement experiments. It should be in next year’s course.