Opty

David Sarda, Kymry Burwell, Michael Sammler

January 3, 2018

1 Introduction

Transactions are an important building block in distributed systems and are used for example in distributed databases. The Opty seminar introduces a transaction server using optimistic concurrency control based on the Paper 'On Optimistic Methods for Concurrency Control' by Kung and Robinson. For this report we implemented this server in Erlang using backward validation and forward validation and evaluated it based on different experiments.

2 Work done

For this seminar we implemented a transaction server using optimistic concurrency control with backward and forward validation together with an updatable data structure, which can be accessed by concurrent processes. The implementation was based on the provided templates in the assignment. The root folder contains the code with backward validation and the distributed variant in `opty_srv.erl`. The variant with forward validation for section 5 a) can be found in the `forward_validation` folder. The modifications are described in the Experiments sections.

3 Experiments

For the experimentation we have performed a series of experiments in order to test the impact of the modification of the different parameters suggested in the seminar. For each series of experiments we provide the mean between the percentage of success of all the clients, and also the maximum difference value between them.

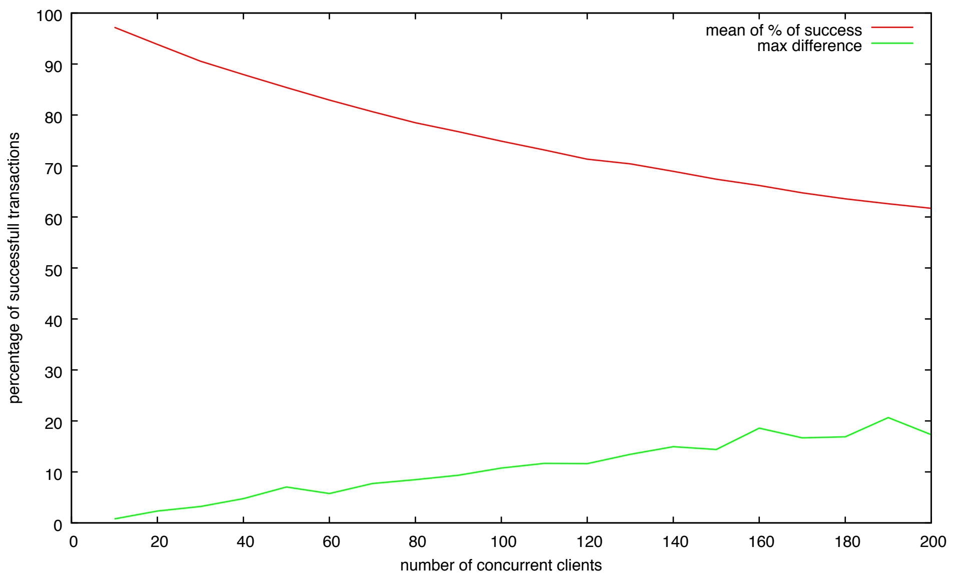


Figure 1: Percentage of successful transactions when increasing the number of concurrent clients, with 1000 entries, 2 read and write operations per transaction, and a wait time of 2 seconds.

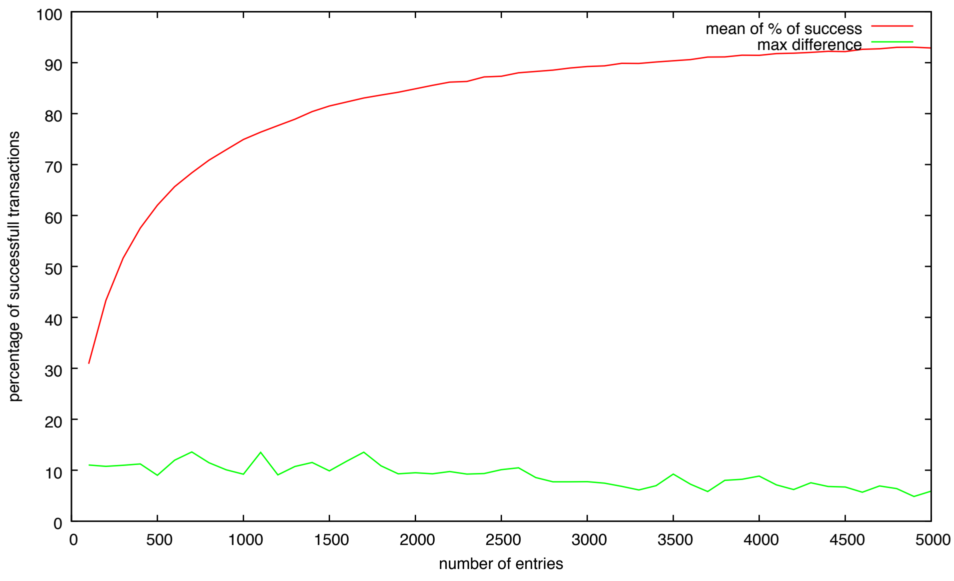


Figure 2: Percentage of successful transactions when increasing the number of entries, with 100 concurrent clients, 2 read and write operations per transaction, and a wait time of 2 seconds.

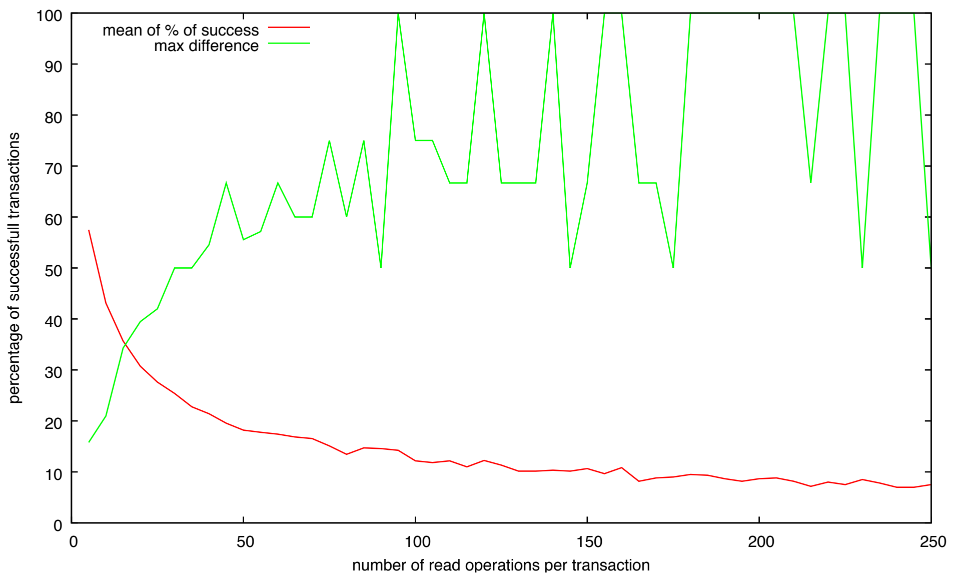


Figure 3: Percentage of successful transactions when increasing the number of read operations per transaction, with 100 concurrent cients, 1000 entries, 2 write operations per transaction, and a wait time of 2 seconds.

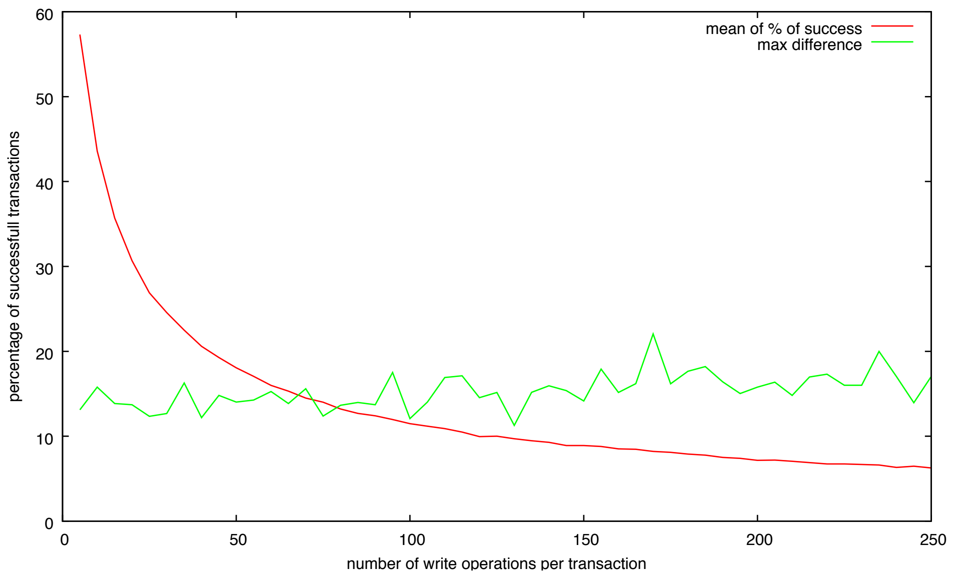


Figure 4: Percentage of successful transactions when increasing the number of write operations per transaction, with 100 concurrent clients, 1000 entries, 2 read operations per transaction, and a wait time of 2 seconds.

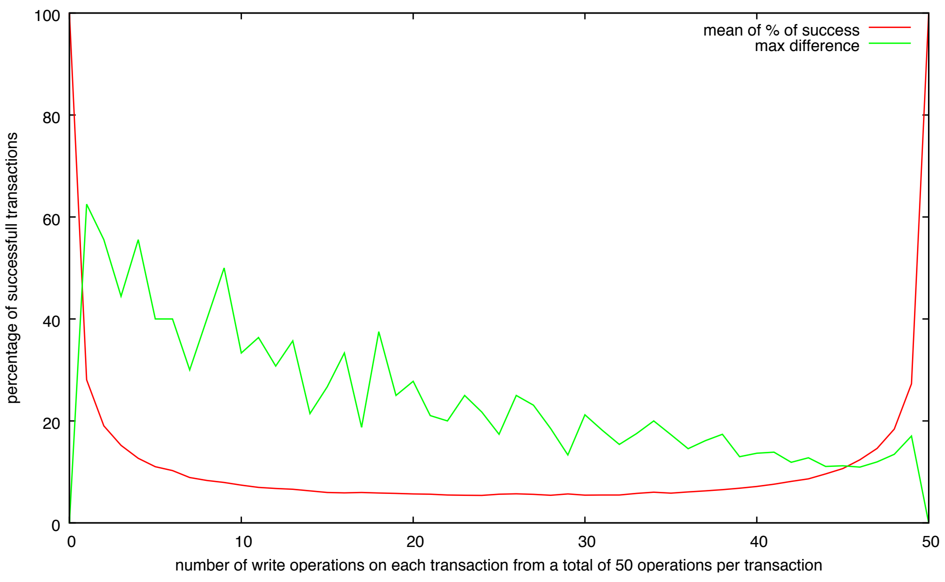


Figure 5: Percentage of successful transactions when modifying the ratio of read and write operations per transaction, with 100 concurrent clients, 1000 entries, and a wait time of 2 seconds.

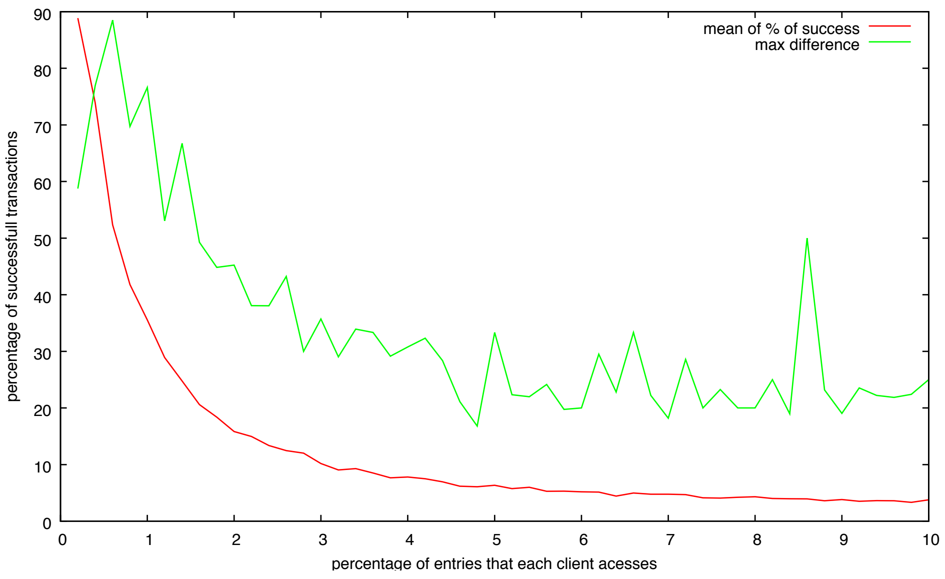


Figure 6: Percentage of successful transactions when modifying the percentage of entries accessed by each client, with 100 concurrent clients, 1000 entries, 2 read and write operations per transaction, and a wait time of 2 seconds.

On figure 1, the impact when modifying the number of concurrent clients is shown. As is expected, with more clients, more conflicts may arise, as more clients will be trying to access the same entries. We can also observe that the difference between the success of different clients increases as the number of clients increases. This is both because the number of clients to consider for this difference will be larger and that some of those clients will have more conflicts than the others.

On figure 2, the impact when modifying the number of entries is shown. As is logical, when increasing the number of entries the percentage of success will increase as well, as with more entries less conflicts will arise. We can also see that the difference in success between clients remains low and stable. This is because if only the number of entries is modified the conflicts that may arise will tend to be evenly distributed between all clients.

On figure 3, the impact when modifying the number of read operations per transaction is shown. As we can see, the percentage of success decreases as the number of read operations increases. This is because when increasing the number of read operations, we increase the chance of having a conflict on every transaction. Also, we can observe that as there are more conflicts, there will be some clients that can't perform their transactions, while some other clients can be lucky and perform most of theirs, increasing the difference as we increase the number of read operations.

On figure 4, the impact when modifying the number of write operations per transaction is shown. As on the previous figure, increasing the write operations per transaction will incur in more conflicts, thus decreasing the success rate. Nevertheless the difference here remains stable and somewhat low, as even if we increase the write operations, they won't generate more conflicts by themselves.

On figure 5, the impact when modifying the ratio between read and write operations is shown. As is expected, when only having one of the two kinds of operations no conflicts will be possible, and thus the success rate will be perfect for every client. On the other hand we can also see that the success rate is lower when the ratio is as equal as possible. As we have seen on the previous two figures, the difference tends to be higher when there are more read operations than when there are more write operations.

On figure 6, the impact when modifying the percentage of entries accessed by each client is shown. We can see that when this percentage is very small, the success rate is very high. This is because when each client can only access a very small percentage of entries, the probability that this same entries are shared with other clients is very small. As expected, this success rate decreases rapidly when this percentage increases, because then it will be more likely than different clients share the same entries. We can also see that the difference is higher when the percentage is very small. That is because when two clients do share entries, the probability of conflict between them will be very high, as they have a very small pool of entries to choose from.

3.1 Distributed execution

For this experiment we altered the original Opty code to run the server on a remote node. This updated version can be found in the `opty_srv.erl` file. In order to run the experiment, the following steps must be taken:

1. Start the server node by executing

   ```
   erl -sname opty-srv -setcookie secret
   ```

   and then loading the server in it by executing `c(server).`

2. Start the main node, which runs the clients, by executing

   ```
   erl -sname main -setcookie secret
   ```

   and then loading the `opty_srv` file in it by executing `c(opty_srv).`

3. Choose the parameters and execute on the main node

   ```
   opty_srv:start(Clients, Entries, Reads, Writes, Time).
   ```

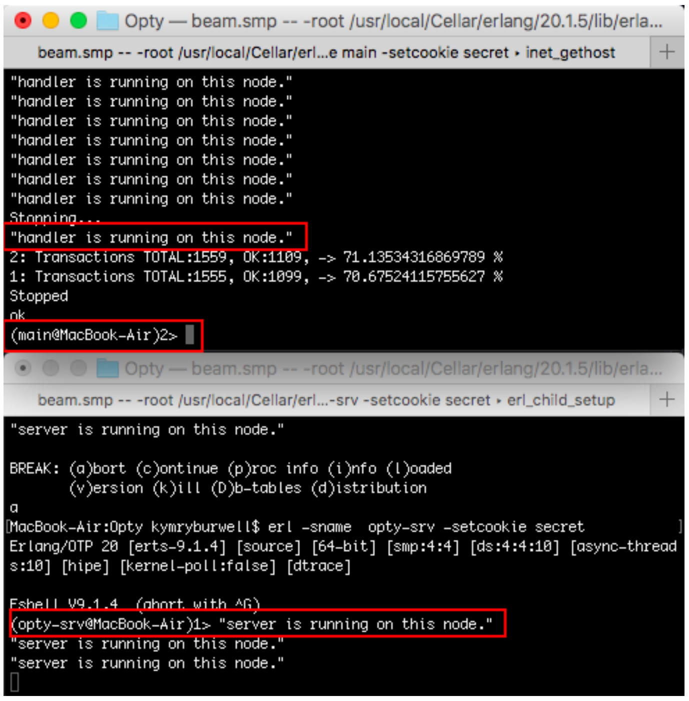
If we run this in a distributed Erlang network, the handler is running in the Erlang instance where the program was initiated. In our experiment, this would be in the *main* node. In order to test this we placed the following code in the handler.erl file:

```
rpc:call( node(), erlang, display, [ "handler is running on this node." ]);
```

We placed very similar code in the server.erl file:

```
rpc:call( node(), erlang, display, [ "server is running on this node." ]);
```

Ther results, shown in figure 7, confirm that the handler is running in the *main* node and the server is running in the *opty-srv* node.


Opty — beam.smp -- -root /usr/local/Cellar/erlang/20.1.5/lib/erla...
beam.smp -- -root /usr/local/Cellar/erl...e main -setcookie secret ▸ inet_gethost

```
"handler is running on this node."
"handler is running on this node."
"handler is running on this node."
"handler is running on this node."
"handler is running on this node."
"handler is running on this node."
"handler is running on this node."
Stopping...
"handler is running on this node."
2: Transactions TOTAL:1559, OK:1109, -> 71.13534316869789 %
1: Transactions TOTAL:1555, OK:1099, -> 70.67524115755627 %
Stopped
ok
(main@MacBook-Air)2>
```

Opty — beam.smp -- -root /usr/local/Cellar/erlang/20.1.5/lib/erla...
beam.smp -- -root /usr/local/Cellar/erl...-srv -setcookie secret ▸ erl_child_setup

```
"server is running on this node."

BREAK: (a)bort (c)ontinue (p)roc info (i)nfo (l)oaded
       (v)ersion (k)ill (D)b-tables (d)istribution
a
MacBook-Air:Opty kymryburwell$ erl -sname  opty-srv -setcookie secret
Erlang/OTP 20 [erts-9.1.4] [source] [64-bit] [smp:4:4] [ds:4:4:10] [async-thread
s:10] [hipe] [kernel-poll:false] [dtrace]

Eshell V9.1.4  (abort with ^G)
(opty-srv@MacBook-Air)1> "server is running on this node."
"server is running on this node."
"server is running on this node."
```

Figure 7: Nodes running the Handler and Server

3.2 Forward validation

For the last experiment about other concurrency control techniques, we chose to implement forward validation. The implementation can be found in the forward_validation subfolder. The following adjustments were made to the original code:

- Each transaction is assigned a transaction id at the start of the handler. This TId is sent to the entry on a read to keep track, which transaction are currently active.
- Each entry keeps track of the transactions, which have read the entry and are still active.
- Before a transaction is validated, it is removed from the active lists of all entries.
- After this the validator asks all entries, which were written by the transaction, whether another transaction read the value and is still active.
- If yes, the transaction, which is currently validated, is aborted.
- If no, the entries are updated.

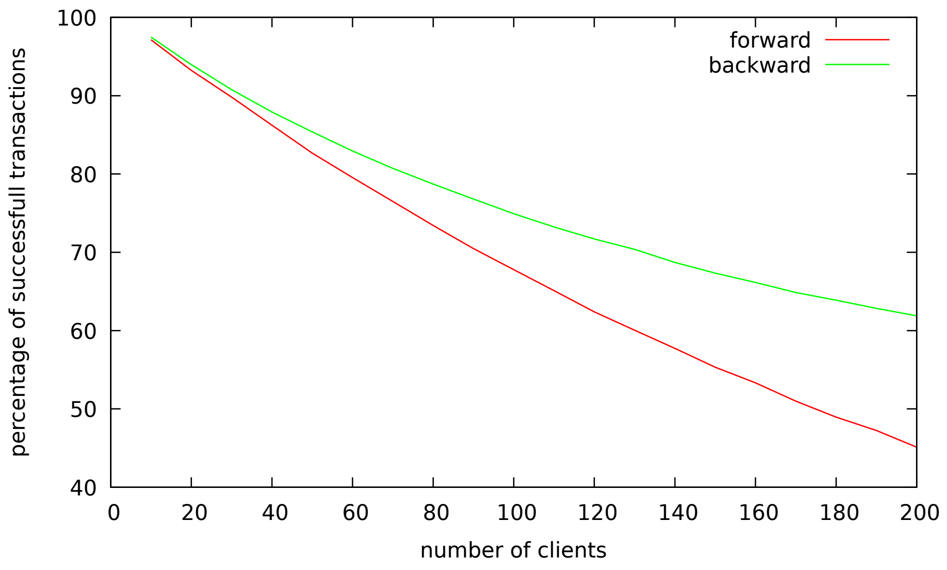


Figure 8: comparison forward and backward validation

- We also need to make sure, that no transaction reads an entry between the time, when the active reads are checked and the actual value is written. For this purpose each entry maintains the number of pending writes, which is incremented each time it is checked and decremented on each write or when the validator sends a message, that the write was aborted. A read message is only considered by an entry, if no writes are still pending.

The implementation with forward validation was compared against the version with backward validation based on the number of concurrent clients with 1000 entries, 2 reads and 2 writes per transaction and with a timeout of 5 seconds. The results can be seen in figure 8. It shows, that the forward validation performs worse than the backward validation, if the number of clients grows.

4 Open questions

All questions are answered in the experiments section.

5 Personal opinion

As with the previous seminar, we thought this was a worthwhile assignment. It's very helpful to apply what we learn in theory class, allowing us to more fully understand and visualize the concepts. We recommend that you include this seminar in next year's class.