Reconsidering Optimistic Algorithms for Relational DBMS

Malcolm Crowe, University of the West of Scotland (presenter)

Fritz Laux, Reutlingen University, Germany

Malcolm Crowe

- Emeritus Prof, University of the West of Scotland
- Joined UWS in 1972 (as Paisley College)
- Various academic posts in UWS, retired 2018
- Open-source software projects
  - https://github.com/MalcolmCrowe
- Since 2004 developed PyrrhoDBMS
  - https://pyrrhodb.blogspot.com

I'm Malcolm Crowe, a retired professor from the University of the West of Scotland. Over the years I have led numerous funded research projects and contributed a lot of open-source software.
My most recent work is on Github, and it includes the two DBMS I want to talk about today.
Since 2002 I have written in C# for Windows and Linux/Mono, and since 2004 I have been developing Pyrrho, my own relational DBMS.
Since 2018 I have been exploring the use of immutable, shareable data structures to help with implementing optimistic concurrency control.

Background

- At DBKDA 2019 I demonstrated StrongDBMS with TPCC
  - StrongDBMS uses optimistic concurrency control OCC
    - And OCC is not supposed to work well in high concurrency!
  - We changed the TPCC benchmark for high concurrency
    - And the usual TPCC design then had more sources of conflict
  - StrongDBMS demo outperformed commercial DBMS
    - The demo showed concurrent operation of 100 clerks/warehouse
- This contribution explains the results:
  - The demo showed OCC can be robust and effective

So at last year's DBKDA, I presented a demo of StrongDBMS, a little DBMS that followed this design. The demonstration modified the famous TPCC benchmark slightly to create high transaction concurrency, initially as a proof that the DBMS had ACID properties.

To my great surprise, StrongDBMS handled concurrency much better than commercial DBMS products. We are not allowed to talk about the performance of the major commercial DBMS, so I provided a test based on TPCC in versions for StrongDBMS and several popular DBMS, for people to try at home on their favourite DBMS.

The test allowed people to choose the number of clerks per warehouse, and at the Athens conference I showed a live 10-minute demo of StrongDBMS handling 100 concurrent clients in this test. We found no other DBMS capable of this level of concurrency.

The demonstration was convincing, but there was little attempt to explain the difference in performance. We offer such an explanation today, as we believe there are lessons here that implementations based on optimistic concurrency algorithms can be robust and effective, and it is time to reconsider the myths about their performance.

DBMS mostly use locking

- The standard mechanism for concurrency
  - Appears to provide assurance of commit success
    - But only if CAP theorem and 2-army problems are ignored
- Locking leads to complex implementation
  - And does not work well for interactive use, or Web
- Most Web applications use OCC
  - But middleware brings a third step in commit

Locking appears to offer the assurance that transactions can be committed atomically and serialisably. However, it is not an absolute guarantee, since networking or hardware problems can defeat even the most secure systems, resulting in what is called heuristic completion as a last resort. It is also extremely complex in practice. On the Internet, data is often spread over multiple databases and users often want to make complex changes to multiple data, such as when booking holidays. Long transactions result in unacceptable delays and so many compromises must be made.

For these reasons, most Web applications must use a form of optimistic transaction control, as the Web servers can mediate the interaction with the DBMS. But this only serves to complicate the picture, as now there are more participants in what has become a distributed transaction.

The case study in detail

- TPCC benchmark has 1 clerk per warehouse
  - A clerk can process up to 16 new orders in 10 mins
    - Transactions are quite long
  - A conflict rate of 4% is designed in to the test
- For our case study we add more clerks (100)
  - Conflicts on running totals of orders and payments
    - Read/Write conflict at row level
  - Conflicts on allocating new order numbers
    - Write/write conflict
  - With 100 clerks and 1 warehouse, nearly all transactions must fail

The demonstration used a modified version of the famous TPCC benchmark. This simulates an online transaction processing system and has been in use for decades in comparing the performance of different DBMS.

There are warehouses with many items and customers arranged in districts, and one clerk per warehouse accepting orders over the telephone. There is only one database server.

The specification is very detailed, with a standard initial state and models a range of routine tasks that the clerks must carry out. In the design, interaction between warehouses results in about 4% conflicts, mostly on stock levels.

The performance measure for DBMS is the number of new-order transactions per second, and for commercial databases this can run to millions. Despite this, the specification requires each clerical task to take quite a long time: up to 20 seconds for a new order, so that in a 10-minute test run a single clerk can complete 16 new orders.

To create a more challenging environment, our demonstration kept everything from the TPCC specification, but allowed the number of clerks to increase for a single warehouse.

The TPCC design then has many additional sources of conflict, especially on total sales and payments to date, and on allocating order numbers. There still are issues with stock levels. Transactional behaviour is important for two of the standard tasks: new order and payment. Inevitably therefore, as the number of clerks increases, the number of new orders per clerk will diminish, and the total number of new orders in 10 minutes (the throughput) will be less than 16 times the number of clerks.

If there are 100 clerks, most new-order transactions will fail.

StrongDBMS worked well

- Very simple algorithms, and only two locks
- Fully serialisable transactions despite conflicts
- Correctness implies most transactions fail in this test
- But higher throughput resulted with OCC implementation
- Serialisation of transactions evident in the log
  - Despite large number of overlapping transactions
- StrongDBMS uses immutable shareable data structures
  - PyrrhoDB v7 extends this approach to a full DBMS
    - Under development: a demo alpha version is available

StrongDBMS has only two kinds of lock: one on the list of open databases, and one per database file. The database file is the transaction log, a serialised record of transactions committed. The transaction log will show the correctness of the successful transactions.

In addition, the demo maintained a full list of the requests made to the DBMS, so that its behaviour could be verified.

The special feature of StrongDBMS enabling reliable operation is the use of immutable shareable data structures, and this has led me to use the same idea in the next version of Pyrrho (v7, currently alpha).

Results for StrongDBMS

- The warehouse initially has 30000 orders,
  - Including 9000 "new" orders.
- In 10 minutes a single clerk has done 16 more
  - Too many clerks result in reduced performance

Name	Initial	1 clerk	10 clerks	20 clerks	30 clerks
Commits	0	39	302	512	565
Exceptions	0	0	104	387	1071
ORDER	30000	30016	30138	30199	30241
NEW_ORDER	9000	9016	9138	9199	9241
ORDER_LINE	285007	285158	286207	286638	286965

This table of results was included in the DBKDA 2019 conference paper. The initial state of the database is as specified by TPCC: 3000 orders for each of the 10 districts (30000 in all), some of which are called "new". With 1 clerk, we see 16 new orders being completed in the 10 minutes.

With 10 clerks, we get 138, and with 30 only 241. We see that the number of successful commits also falls. 39 for a single clerk, but only 565 for 30 clerks, when the number of exceptions reported is greater than the number of successful commits.

Other DBMS did less well

- 2 top commercial DBMS and PostgreSQL
- All tests show completed orders in 10 minutes

Name	1 clerk	2	5	10	20	30	40	50	60
StrongDBMS laptop	16			138	199	241	*		
StrongDBMS 16GB RAM	16			129	220	254	409	331	328
Commercial1	16			111	127	132	16	*	
Commercial2	16			107	114	119	124	117	*
PostgreSQL	16	33	69	6	*				

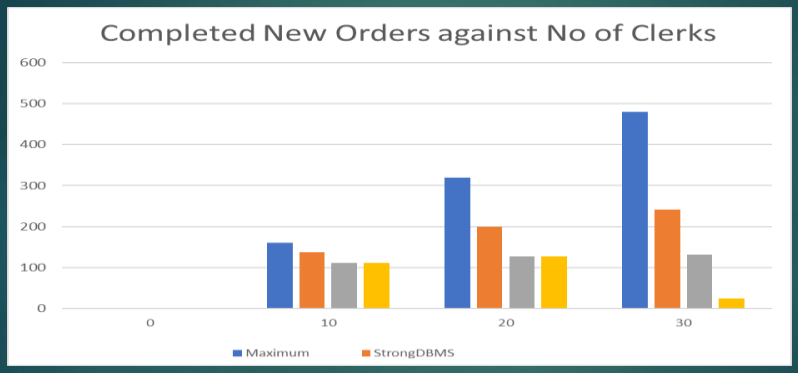
- The * indicates a collapse in performance

This slide shows results reported at the Athens conference (where PostgreSQL was called Comercial3). The top row shows the results mentioned in the last slide. We can see an improvement from using a large machine. But the commercial DBMS (on the same machine) were outperformed. The asterisks indicate a collapse in performance, markedly less throughput and/or non-reproducible behaviour such as is found when the device is overloaded.

PostgreSQL performed particularly badly, and after receiving some feedback on this we agreed to discover the reasons why. Hence this paper.

Comparison chart

- Can't show PostgreSQL on this chart



- Blue: Theoretical max 16 orders/clerk. Orange: StrongDBMS
- Grey: Commercial1. Yellow: Commercial2

Some of the tests from the last slide are charted here (the line for PostgreSQL can't be shown). The blue bar shows the unachievable 16 orders per clerk. StrongDBMS is the orange one,, and the other two bars are the commercial DBMS tested.

Better performance explained

- OCC only checks for conflicts on COMMIT
  - And then First Committer Wins (FCW)
  - DBMS literature assumes this brings high costs
    - And unnecessary rollbacks
- But the demo proves that OCC+FCW works
  - Even when the load is high
- Each transaction works in isolation until commit point
  - Other DBMS keep checking and abort on potential conflict
    - Optimistic keeps open the possibility someone will change their mind
- But also the granularity of conflict detection is better..

A full analysis is in the conference paper. With optimistic concurrency control, all checking for conflicts is delayed until the client wishes to COMMIT their transaction. And then, if there are no conflicts, the commit happens immediately (the file is locked just for this). This is often called First Committer Wins. In DBMS literature, the conventional wisdom is that this brings higher costs and unnecessary rollbacks.

But the demo proves that the combination of optimistic concurrency control and first committer wins works well, even when the concurrency is high.

Each transaction works in isolation until the commit point, whereas other DBMS including PostgreSQL keep detecting for possible conflict and abort when completion cannot be guaranteed.

For the particular sources of conflict here, StrongDBMS also has a better granularity for detection of conflicts, described in the next slide.

Conflict handling in StrongDB

- Finer than row locking! 3 levels of constraint:
- If rows not selected by specific keys, report conflict if any columns have been updated by another transaction
- Else for rows selected by specific key values, limit check to updates of selected records, or to updates of keys
- Else raise conflict on any other update to the table that happened since start of transaction
- This approach has been used by PyrrhoDBMS since 2005
  - Other DBMS including PostgreSQL have a different test

The conflict detection used by StrongDBMS is somewhat finer than row locking. It is not new, as Pyrrho has used the same algorithm since 2005. The actual words in Pyrrho's source code are reproduced in the conference paper.

There are three levels of constraint (applied at the end of the transaction), depending on what has been read from a table by the transaction.

A: If all rows have been accessed, or the rows have been selected otherwise than by key values, report conflict if any columns have been updated by another transaction.

B: If one or more rows have been accessed using specific key values, we can limit the check to updates of these records or updates of key columns.

C: If things are more complicated, any update to the table that has happened since the start of the transaction will cause the transaction to be aborted.

This is the second reason for the PostgreSQL result: their test for conflict is different.

Workarounds

- StrongDBMS and PyrrhoDB allow only serialisable tx
- Reducing the isolation level for other DBMS led to better throughput
  - Though we saw some incorrect operation of a commercial product for these levels
- Escrow methods can also help to avoid hot spots such as running totals and next order
  - With commutative and increment semantics (resp.)
  - These need changes to application protocols

These two DBMS only allow serialisable transactions. Theoretical arguments show that in most cases REPEATABLE-READ isolation is good enough to ensure consistency, and the other DBMS had better throughput with this isolation level.

There are other ways of improving the performance of commercial DBMS in situations of high concurrency for the sorts of conflict that occur here.
In particular, escrow methods allow running totals to be maintained using commutative semantics, and the next-order-number problem to be solved using increment semantics. But such mechanisms need changes to application protocols, and would be a departure from the TPCC specification.

Conclusions

- The study reported here makes a case for extending optimistic algorithms to other database products
- This would help to remove the "impedance mismatch" between application and DBMS protocols
- Myths about OCC are deeply entrenched in the database community, but it is time for better and more considered analysis
- The demonstration code and PyrrhoV7 alpha is on Github, along with further documentation and resources

These optimistic algorithms could be used in other database products. There is occasionally discussion in the literature about the mismatch between optimistic application protocols and lock-based database protocols.

We feel this demonstration has exposed some of the comments about optimistic concurrency control to be myths. It is time for better and more considered analysis of the design choices involved.

The references include locations of publicly-available demonstration code and other resources.

References

- **M. Crowe and C. Fyffe, "Benchmarking StrongDBMS", Keynote speech, DBKDA 2019,** https://www.iaria.org/conferences2019/filesDBKDA19/MalcolmCrowe_CallumFyffee_Keynote_-BenchmarkingStrongDBMS.pdf
- **M. Crowe, *An introduction to the source code of the Pyrrho DBMS*. University of Paisley, 2007**
- **F. Laux and T. Lessner, "Escrow serializability and reconciliation in mobile computing using semantic properties." *International Journal on Advances in Telecommunications* 2.2, pp. 72-87, 2009**
- **F. Laux and T. Lessner, "Transaction processing in mobile computing using semantic properties." *2009 First International Conference on Advances in Databases, Knowledge, and Data Applications*. IEEE, pp. 87-94, 2009**
- F. Raab, W. Kohler, and A. Shah, "Overview of the TPC-C benchmark: The Order-Entry Benchmark", **Transaction Processing Performance Council, Tech. Rep., 2013, http://www.tpc.org/tpcc/detail5.asp**
- **TPCC specification** http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-c_v5.11.0.pdf
- GitHub, https://github.com/MalcolmCrowe/ShareableDataStructures