Performance Modelling of Database Contention using Queueing Petri Nets

David Coulden Rasha Osman William J. Knottenbelt
Department of Computing
Imperial College London
London SW7 2AZ, UK
{drc09, rosman, wjk}@imperial.ac.uk

ABSTRACT
Most performance evaluation studies of database systems are high level studies limited by the expressiveness of their modelling formalisms. In this paper, we illustrate the potential of Queueing Petri Nets as a successor of traditionally-adopted modelling formalisms in evaluating the complexities of database systems. This is demonstrated through the construction and analysis of a Queueing Petri Net model of table-level database locking. We show that this model predicts mean response times better than a corresponding Petri net model.

Categories and Subject Descriptors
C.4 [Performance of Systems]: *Modelling techniques.*

General Terms
Performance

Keywords
Queueing Petri nets, performance modelling, database locking.

1. INTRODUCTION
The data landscape has changed dramatically in size and complexity in the past decade. The Internet, ubiquitous communication, cloud services and e-Science applications have led to an explosion in data generation and storage. A large proportion of this data is stored and managed in databases. Market growth for relational database management systems is expected to double by 2016 [11], making performance of these large DBMSs a critical issue for users and vendors alike.

Database system performance is influenced by complex and interdependent functionalities (e.g. transaction usage scenarios, database cache management, disk contention, concurrency and lock contention and the implementation of logical and physical structures in DBMS). In the performance evaluation literature there have been many performance studies of different components of database systems and many methodologies developed for their performance evaluation [14]. However, the impact of these studies on industry has been limited. One of the reasons is the lack of detailed modelling needed to represent production-grade database systems. A main cause is the interaction of physical and logical resources within database systems, which is difficult to represent using traditional modelling formalisms. In this work, we return to the issue of modelling database systems by modelling table level Two Phase Locking using queueing Petri nets and illustrate the potential that this new approach has in performance modelling of database systems.

Queueing Petri Nets (QPNs) [1] extend coloured stochastic Petri nets by incorporating queues and scheduling strategies into places forming *queueing places*, thus producing a very powerful modelling formalism that has the synchronization capabilities of Petri nets (PNs) while also being capable of modelling queueing behaviours. These queueing places consist of two components: the *queue*, and the *depository* where serviced tokens (customers) are placed. Tokens enter the queueing place through the firing of input transitions, as in other Petri nets; however, as the entry place is a queue they are placed in the queue according to the scheduling strategy of the queue's server. Once a token has been serviced it is deposited in the depository where it can be used in further transitions. Queueing places can have variable scheduling strategies and service distributions; these are known as *timed queueing places*. *Immediate queueing places* impose a scheduling discipline on arriving tokens without a delay.

Queueing Petri nets have been recently applied in the performance evaluation of component-based distributed systems [4, 5] and grid environments [10]. In this paper, we apply QPN in modelling locking contention in database systems. We have chosen QPN as the modelling formalism over other variations of Petri nets, as the queueing places allow for the representation of lock scheduling in database systems, while the places and transitions naturally represent the flow of execution of a transaction in the system. Even though queueing network models (QNMs) are currently the prevailing formalism for performance modelling of database systems, QPNs are more expressive when representing simultaneous resource possession and blocking.

The rest of this paper is organized as follows. Section 2 overviews related work. Section 3 details the QPN model for a particular measured system. Section 4 analyzes the results and Section 5 concludes the paper.

2. RELATED WORK
2.1 Queueing Network Models
Osman and Knottenbelt [14] surveyed queueing network performance models of database systems. They found that the majority of studies that model concurrency control in database systems assume a uniform distribution of the locks over the total number of data items, in addition to representing update only transactions in the models. While these assumptions produce tractable models, they neglect hot-spots and do not represent the

ICPE'13, April 21–24, 2013, Prague, Czech Republic.

effect of read transactions holding shared locks on the execution of update transactions that hold conflicting locks. Moreover, these models assess the performance of the transaction at the physical hardware level; therefore they are incapable of representing the effect of lock conflicts at the table or row level, which is more beneficial to database performance tuning.

2.2 Petri Nets

There is a paucity of studies that apply Petri nets to database systems in comparison to the available research that applies queueing network models. Of the studies that do exist, Chen [2] analyzes deadlock detection scheduling in centralized databases using stochastic Petri nets. The places in the model represent transaction execution states, i.e., waiting for a lock, locking an item, CPU processing, etc. Firing of the timed transitions represents the delay in moving from one state to the next.

For parallel databases, Mikkilineni et al. [9] use a Petri net to model concurrent parallel query execution plans in a distributed database. In the PN model, the transitions represent query operations and the places represent data blocks. The firing of a transition represents data communications. Jenq et al. [3] analyze two-phase locking in a parallel database machine using a two-layered model. The higher-layer model is a Petri net representing the parallel and synchronized execution of the relational operations of a transaction. The lower-level model is a QNM that represents the hardware resources and lock waiting queues.

The modelling approach presented in this paper differs from that of previous work in that we do not model the synchronization of query execution plans. Instead, we borrow the concept of modelling the execution of the transaction at the database table level from previous work in modelling database systems using queuing networks [13]. This work is an improvement over previous models of database systems in that we are able to represent locking contention between read and write transactions in a way similar to that of actual systems. Moreover, our QPN models have a more intuitive structure that maps the transaction and database design to the QPN model. This makes the model easier to comprehend by database developers and administrators.

3. Queueing Petri Net Model

3.1 Measured System

DBMSs implement concurrency control through locking protocols. The most widely used protocol is Strict Two Phase Locking (Strict 2PL) [15]. Strict 2PL forces transactions to hold *exclusive locks* to modify data and *shared locks* to read data. For a transaction to acquire an exclusive lock on a data object, no other transaction should hold a shared or exclusive lock on that object. Transactions can acquire shared locks on data objects only if no transaction has an exclusive lock on the objects. In our case study, we model Strict 2PL at the table level. A version of table level Strict 2PL is the default locking method implemented in the MySQL default storage engine [12]. In our experiments we use the PostgreSQL 9.1 [16] DBMS. In order to mimic table level Strict 2PL we explicitly lock the table within the transactions.

The measured system has two types of transactions: shared and exclusive that compete to access *Table A* (100,000 rows). Both transactions explicitly lock the table in the appropriate lock mode (shared or exclusive) and read or modify 1% of the table rows. Access to the table is achieved through a full table scan, i.e. no index is utilized by either transaction. The structure of the transactions is shown in Figure 1. In order to simulate a TPC-W-like workload in which update transactions are longer than read transactions [17], the execution time of the exclusive transaction is artificially lengthened by 40ms. Clients submit transactions to the DB server with exponentially distributed think times with mean 500ms. The mean response time of each transaction type is measured and compared to that emerging from the QPN model for different transaction mixes. The measured system was run on an Intel(R) Core(TM) i7-2600 CPU@3.40GHz box running Ubuntu 12.10 64-bit and PostgreSQL 9.1.

```
shared transaction
BEGIN;
  LOCK TABLE Table A in
                    ACCESS SHARE MODE;
  SELECT count(*) FROM Table A
                    WHERE id > value;
END;

exclusive transaction
BEGIN;
  LOCK TABLE Table A in
                    ACCESS EXCLUSIVE MODE;
  UPDATE Table A SET other-id = other-value
                    WHERE id > value;
   SELECT pg_sleep(0.04);
END;
```

Figure 1. Structure of shared and exclusive transactions.

3.2 QPN Model of Table Level Locking

The QPN model for the measured system was developed using QPME2.0 [6, 7]; it is detailed in Figure 2. The clients are represented by a timed queueing place with an infinite-server queue. The tokens in the *client* place have two colors; each color represents a client of one transaction type. Clients submit transaction jobs to the database server after an exponentially distributed think time. Then, transactions enter the *lock waiting* place where they wait for the lock on the table to be free. The lock waiting place is an immediate queueing place with FIFO departure discipline which ensures that the transactions are serviced in order of arrival.

The table-level locking mechanism is represented using a *lock repository* place, which is an ordinary place containing *lock* tokens. A share transaction will require one lock token and an exclusive transaction will require the maximum number of tokens defined for the lock repository place. By setting the number of lock tokens within the lock repository place to be equal to the maximum number of share transactions, all share transactions will be able to run simultaneously and an exclusive transaction will be forced to wait if there is a least one share transaction accessing the table. Any transaction entering the database queues behind any waiting transaction. Once a transaction has acquired a lock it will access *Table A*. Table A is represented by a timed queuing place with an infinite server queue which models transaction execution. Each type of transaction is treated as having an exponentially distributed service time that models the entire execution of the transaction. When a transaction has been serviced it will be passed back into the client place to repeat the process.

In our QPN model, we are assuming logical resources are the bottleneck, not physical resources. Therefore, the model does not directly capture disk and CPU contention and performance. However, the effects of processing are partially reflected in the

G/M/∞-IS queueing place representing *Table A*. Infinite server scheduling models the forking of PostgreSQL processes for each database connection. To minimize the effect of DBMS automated disk access, the default PostgreSQL configuration has been modified[1]. This modified configuration will not eliminate disk access but configures the DBMS for performance instead of durability [16].

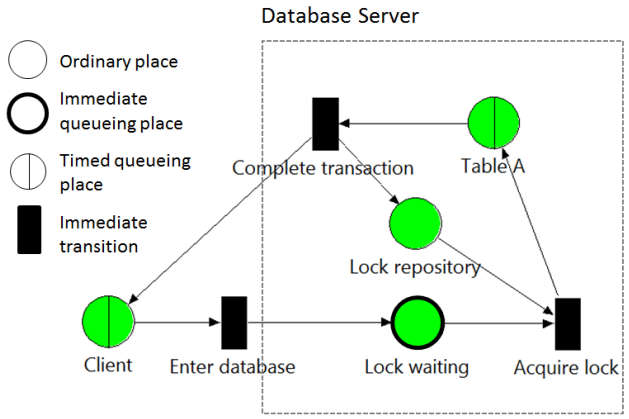


Figure 2. Queueing Petri net model of table level locking.

4. RESULTS

The experiment was based on the workloads of the TPC-W benchmark. The TPC-W benchmark is an e-commerce benchmark implementing an on-line bookstore. It has three workload mixes [8]: the *browsing mix* has 95% reads and 5% updates, the *shopping mix* has 80% reads and 20% updates, and the *ordering mix* has 50% reads and 50% updates.

Each transaction type was executed in isolation, i.e. without any locking contention, and the mean response time calculated. The measured mean response time for the shared transaction was 18.8ms and for the exclusive transaction 60.4ms.

We compare the measured system with our QPN model and an equivalent PN model. The PN model is the same as the QPN model in Figure 2 except that the *lock waiting* place is an ordinary place[2]. We have not compared to a QNM, as this type of simultaneous resource possession and blocking is difficult to express using QNMs.

The results for the browsing, shopping and ordering workloads are presented in Figure 3. First, we will discuss the performance of the actual system. For the browsing workload (Figure 3(a)) the share transactions dominate the traffic and their performance is minimally affected by the increase in exclusive transactions. The *step-like* trend for both transactions is caused by the constant number of exclusive transactions for the corresponding number of clients. For the shopping mix (Figure 3(b)) the increased number of exclusive transactions has affected the performance of both types of transactions, i.e. lock waiting time has increased in comparison to transaction execution time. This is especially evident for the shared transactions where we notice a sharp increase in the mean response time for large number of clients. For the ordering workload (Figure 3(c)) in which the number of shared and exclusive transactions are equal, lock waiting time dominants transaction execution which is evident in the performance degradation of the shared transactions, leading to approximately the same response times for both transaction types at high client numbers.

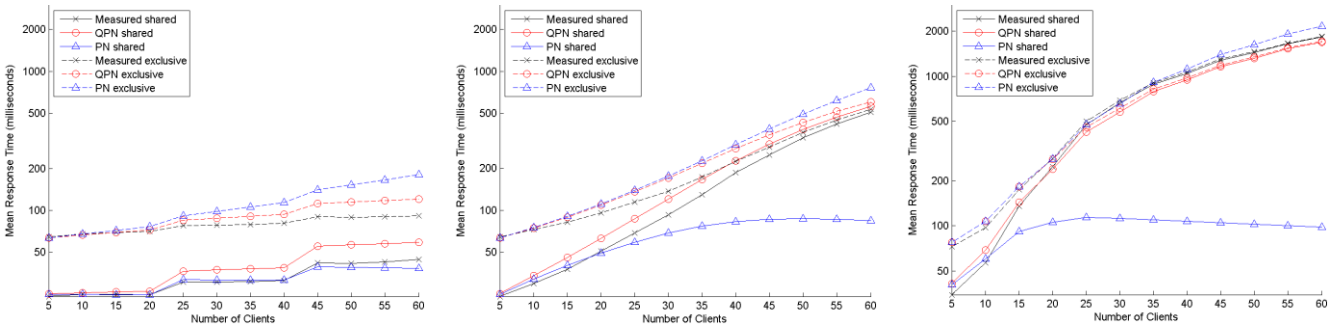
The PN model severely underestimates the performance of the exclusive transactions for the browsing workload with an error of 97% at 60 clients. However, it overestimates the performance of the shared transactions, with an error of 13% at 60 clients. This is due to the fact that in the PN model the transactions do not queue for locking, as in the real system; and therefore the exclusive transactions are starved. This trend continues as the percentage of exclusive transactions increases in the shopping and ordering workloads. From Figures 3(b) and 3(c), the PN model overestimates the performance of the shared transaction by 83% and 94%, and underestimates the performance of the exclusive transactions by 43% and 17% at 60 clients for the shopping and ordering workloads respectively. The increase in exclusive transactions in the workloads increases their probability of holding the lock token, thus the accuracy of the PN increases as the number of exclusive transactions increase. However, the opposite effect is seen on the shared transactions as without a scheduling discipline they are able to skip ahead of the exclusive transactions whenever a lock token is held by at least one share transaction.

The QPN model underestimates the performance of both transactions for the browsing workload with an error of 32% at 60 clients. The accuracy of the QPN model increases as the number of exclusive transactions increase in the system, i.e. when the lock waiting times dominate the response times. For the shopping workload the QPN underestimates the performance of both transactions with an error of 10% for the shared transaction and 13% for the exclusive transaction at 60 clients. The QPN model overestimation is possibly due to the unaccounted multi-core processing. For the ordering workload, the QPN correctly predicts that both share and exclusive transactions have approximately the same response times. Unlike the previous workloads, the QPN model underestimates the performance of both transaction types when the number of clients is less than 20, with an average error of 6%. When the number of clients is 20 or more, the QPN model overestimates the performance of both transaction types with an average error of 8% at 60 clients. The overestimate is due to the high updates that cause more disk access which, in turn, affects the response times of both transaction types. This is not accounted for in the QPN model. Nonetheless, the QPN model is able to follow the performance trend for both transactions for all workloads, especially the share transaction in comparison to the PN model.

---

[1] The modified server configuration parameters are: fsync=off, synchronous_commit=off and checkpoint_segments =600. The reader is referred to [16] for a definition of these parameters.

[2] Access to *Table A* should have been modelled as a timed transition with infinite service. However, QMPE2.0 does not support timed transitions [6], so this was approximated by a serial network consisting of an immediate transition, a timed queueing place and a second immediate transition similar to the QPN model.



(a) browsing (95% shares and 5% exclusives) **(b)** shopping (80% shares and 20% exclusive) **(c)** ordering (50% shares and 50% exclusive)

Figure 3. Mean response time for TPC-W workload.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we have demonstrated the potential in modelling relational database systems using Queueing Petri Nets, thus overcoming some of the limitations of queueing network models and Petri nets which are currently the main modelling formalisms used to represent database systems.

We have presented a QPN model of two-phase table-level locking. The QPN model was able to approximate the queueing of the lock requests to the table for varying workloads, in contrast to an equivalent Petri net model.

This paper is a starting point for further investigations, which will include the extension of the QPN model to incorporate the effect of hardware contention on the system. This will lead to modelling of more representative database systems with more realistic workloads. Furthermore, for this approach to be feasible and applicable, an automated mapping tool will be developed. We will also investigate emerging paradigms, e.g. NoSQL databases, although the diversity of their data models and implementations will likely mean a generic modelling framework will be infeasible.

6. REFERENCES

[1] Bause, F. 1993. Queueing Petri Nets—A Formalism for the Combined Qualitative and Quantitative Analysis of Systems. In *Proc. 5th Int'l Workshop Petri Nets and Performance Models* (Oct 19-22, 1993), 14-23.

[2] Chen, I.-R. 1995. Stochastic Petri Net Analysis of Deadlock Detection Algorithms in Transaction Database Systems with Dynamic Locking. *The Computer Journal*, 38, 9 (1995), 717-733. DOI: 10.1093/comjnl/38.9.717

[3] Jenq, B.-C., Twichell, B.C.; Keller, T.W. 1989. Locking performance in a shared nothing parallel database machine. *IEEE Transactions on Knowledge and Data Engineering*, 1, 4 (Dec 1989), 530-543.DOI: 10.1109/69.43427

[4] Kounev, S. 2006. Performance modeling and evaluation of distributed component-based systems using queueing Petri nets. I*EEE Trans. Software Engineering*, 32, 7 (July 2006), 486-502.

[5] Kounev, S. and Buchmann, A. 2003. Performance modelling of distributed e-business applications using queuing petri nets. In *Proc. IEEE International Symposium on Performance Analysis of Systems and Software* (Mar 2003), 143-155.

[6] Kounev, S. and Spinner, S. 2011. *QPME 2.0 User's Guide*. (May 2011) Karlsruhe Institute of Technology, Germany. http://descartes.ipd.kit.edu/fileadmin/user_upload/descartes/QPME/QPME-UsersGuide.pdf

[7] Kounev, S., Spinner, S. and Meier, P. 2010. QPME 2.0-A Tool for Stochastic Modeling and Analysis Using Queueing Petri Nets. In *From Active Data Management to Event-Based Systems and More*, LNCS vol 6462, 293-311.

[8] Menasce, D. A. 2002. TPC-W: a benchmark for e-commerce. *IEEE Internet Computing*, 6, 3 (May/June 2002), 83-87. DOI: 10.1109/MIC.2002.1003136

[9] Mikkilineni, K. P., Chow, Y.-C. and Su, S. Y. W. 1988. Petri-net-based modeling and evaluation of pipelined processing of concurrent database queries. *IEEE Trans. Software Engineering*, 14, 11 (Nov 1988), 1656-1667. DOI: 10.1109/32.9053

[10] Nou, R., Kounev, S., Julia, F. and Torres, J. 2009. Autonomic QoS control in enterprise Grid environments using online simulation. *Journal of Systems and Software*, 82, 3 (March 2009), 486-502.

[11] Olofson, C. W. 2012. *Worldwide Relational Database Management Systems 2012–2016 Forecast*. International Data Corporation (August 2012**)**, Doc # 236273. www.idc.com

[12] Oracle Corporation 2013. *MySQL 5.6 Reference Manual. Internal Locking Methods*, http://dev.mysql.com/doc/refman/5.6/en/internal-locking.html

[13] Osman, R., Awan, I. and Woodward, M. E. 2011. QuePED: Revisiting Queueing Networks for the Performance Evaluation of Database Designs. *Simulation Modelling Practice and Theory*, 19, 1 (Jan 2011), 251-270.

[14] Osman, R. and Knottenbelt, W. J. 2012. Database System Performance Evaluation Models: A Survey. *Performance Evaluation*, 69, 10 (Oct 2012), 471–493. DOI=10.1016/j.peva.2012.05.006

[15] Ramakrishnan, R. and Gehrke, J. 2003. *Database management systems*. McGraw-Hill, Boston, Mass.

[16] The PostgreSQL Global Development Group 2012. *PostgreSQL 9.1.7 Documentation*. http://www.postgresql.org/docs/9.1/static/index.html.

[17] The Transaction Processing Performance Council 2003. *TPC-W Benchmark version 2*. http://www.tpc.org/tpcw/