Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 232 (2009) 101–124
www.elsevier.com/locate/entcs

Application of Queueing Network Models in the Performance Evaluation of Database Designs

Rasha Osman, Irfan Awan, Michael E. Woodward[1]

Department of Computing
University of Bradford
Bradford, West Yorkshire, UK

Abstract

In this paper, we model database designs using queueing networks, giving visibility to the dynamic behaviour of the database design and allowing the database designer to experiment with different design decisions. Our approach is to abstract away the more detailed levels of the database system design by concentrating on the information that is available to the database designer at design time. It differs from other methods of database system performance evaluation in that the performance assessment is specifically targeted at the database design, not at the database system software architecture. We present a bottleneck evaluation of the Transaction Processing Performance Council TPC-C benchmark under different workload conditions and demonstrate how this affects database design decisions.

Keywords: Database design performance, queueing networks.

1 Introduction

The application of queueing networks to the performance evaluation of database (DB) systems and database management systems (DBMS) has been in the context of DBMS component performance in the database field [32] and software system architectures and design models in the performance engineering field [3]. Performance evaluation of database designs, as described in this paper, to the best of our knowledge, has not been previously addressed.

Conventional software testing and performance evaluation methods do not aid in database design assessment, mainly because software testing focuses on the functionality of the system [23, 30] and how well it conforms to its requirements. Performance requirements are tested through software performance engineering techniques [29], but these techniques, again, focus on the functionality of the whole system.

[1] Email: r.i.m.osman@bradford.ac.uk, i.u.awan@bradford.ac.uk, m.e.woodward@bradford.ac.uk

doi:10.1016/j.entcs.2009.02.053

No efforts are made towards the changes that occur inside the database and their effect on overall system performance, as demonstrated in [28], [29] and [3]. This reality discourages any attempts to evaluate database system performance at early design stages, thus promoting the adoption of post-deployment tuning.

Moreover, commercial database performance tuning tools advocate post-deployment tuning. These tools rely on query optimizers and statistics from working production databases to give recommendations for performance improvements [2, 7, 36]. To the best of our knowledge, there are no performance evaluation tools for database designs.

With the emergence of 24x7 Web-based e-service applications with back-end databases, unacceptable performance and unavailability has a high cost. Furthermore, post-deployment database performance tuning is the major contributor to the total cost of ownership of database systems [35]. Therefore, performance evaluation of database designs can no longer be ignored.

In addition to the general acceptance of the high impact of the performance evaluation of software systems in early development life cycle phases [29], a performance evaluation method for database designs has the following benefits:

- prevents the propagation of design problems to the detailed design and implementation stages of a database system;
- simplifies work for database designers as well as application developers; performance evaluation feedback is relevant to the current state of the development process, thereby preventing costly backtracking to change requirements or application design;
- integrates performance evaluation in the database design process as well as the software development process;
- contributes in minimizing post-deployment database system performance tuning.

The intention of this paper is to establish the validity of our method in effectively modelling database designs as a first step in achieving a complete framework for the performance evaluation of database designs and database systems. In [21], we built the argument for the necessity of a performance evaluation model specifically targeted at database designs. In [20], we have shown that queueing network models are applicable to database designs. This was demonstrated through the validation of a queueing network model for the Transaction Processing Performance Council TPC-C benchmark against implementations of the benchmark, namely [9], [10], [14], [15], and [5]. In this paper, using the TPC-C benchmark, we extend the work presented in [20] to include the effect of more realistic transaction arrival rates. Some of the results in [20] are repeated for the sake of clarity.

The rest of this paper is organized as follows: in Section 2, related work is discussed. The application of queueing networks to database designs is described in Section 3. Performance analysis of the TPC-C benchmark is in Section 4. Finally, conclusions and future work are in Section 5.

2 Related Work

The first approach, to the best of our knowledge, to evaluate *database systems* using queueing networks was taken by [26]. Sevcik describes a general framework for estimating workload characteristics from a database system to use as input parameters to a queueing network model, which represents the physical hardware configuration of the final system. Sevcik's work was followed by other frameworks that are minor modifications of his work, either specified for a certain data model as in [6] and [13], or expanded with extensive details on workload acquisition and characterization as in [1].

In general, performance evaluation in the database community focuses on DBMS components and their effect on database systems [32], with the previously discussed research efforts being the few studies into database system performance evaluation described in the literature. A comprehensive survey of performance evaluation in the database field can be found in [31] and [32].

On the other hand, in the performance evaluation discipline, the majority of methods and approaches are concerned with the performance evaluation of the software architecture of software systems [3], modeled on the physical hardware devices. An attempt in the performance evaluation field to evaluate database system performance at a more detailed level was a method for performance evaluation of client/server architectures using queueing networks for a propriety system [18]. Client/server system performance was calculated by estimating transaction service demands, using a model of a DBMS query optimizer, to derive input parameters for a queueing network model of the underlying software architecture of the system. A similar attempt was conducted by [25].

In all the previous performance evaluation methods, the main objective is to define a technique to extract performance parameters for queueing networks from the characteristics of database transactions and tables. The concern is in providing these parameters for a queueing network model that represents the software architecture of the evaluated system. The performance of the transactions as individual entities or the contribution of the database design to this performance, in addition to their effect on overall system performance was not a concern, even though performance tuning of database systems takes this into consideration. The consequence is that performance problems are identified on hardware devices, thereby giving a general indication to the database designer of where the problem is, e.g. the bottleneck is on Disk2, therefore review all transactions that access Disk2: this information, however, is not helpful to the database designer.

In addition, performance evaluation is conducted as the final stage of the overall system design process *after* all design decisions have been bound at the upper layers of the database design process, even though enough information is available in early design stages to permit performance assessment. Hence, bottlenecks that are identified on hardware devices are resolved through a reverse process, by backtracking to early software and database design artifacts to identify the cause of the performance problem, redesigning, and then re-evaluating the performance model

once more. This leads to delays in feedback, complicates the performance evaluation methods and affects their accurate application. Moreover, it questions the applicability of these methods in an industrial setting.

3 Queueing Networks for Database Designs

Database system performance is usually measured in terms of query and transaction response time; the major indicator of a system capacity problem. After a database system has exhibited a performance problem, the main effort of post-deployment performance tuning is concentrated on the revision of the design of the database and the transactions running against the database [16, 24, 27, 36]. Hence, if the flaws of the database design had been discovered before system implementation and deployment, some of the post-deployment performance problems would have been avoided. The database design artifacts are the main contributors to performance problems; therefore, an early evaluation of their performance coupled with the knowledge of the application design is a major factor in the reduction of post-deployment database tuning.

The concept of using queueing networks to evaluate the performance of database designs is built on the premise that the database designer has the ability to estimate the cost of executing a given transaction on the database, based on its cost of execution, by using database query optimization techniques.

At the design stage of database development, query optimization techniques are used as guidelines in designing efficient queries and transactions. These techniques can be adapted by a database designer to optimize a given SQL statement, at design time, in isolation, but are very cumbersome to use when considering the effect of a query on the performance of other transactions, or the effect of concurrent access to the database of different transactions or different invocations of the same transaction. Being so, the trend is to wait until database system deployment, when the effect of concurrency and the interaction of different transactions will be clearer to optimize the performance of problematic queries and updates [8, 24]. By using queueing networks to model the dynamic behaviour of the database design, the database designer can evaluate the dynamic behaviour of the design, before the physical deployment of the database system.

Now, consider a database design with tables and transactions that access these tables. A valid assumption would be:

Total response time of a transaction =

$$\sum_{i=1}^{n} \Big(\text{total time to } \textit{wait for access} \text{ to table}_i + \text{total time to } \textit{access the data} \text{ of table}_i + \text{total time to } \textit{return data to the client} \text{ from table}_i \Big) + \text{total time to } \textit{process procedural statements} \text{ on the client}$$

where $\text{table}_{1,2,\ldots,n}$ are the tables accessed by the transaction.

But:

- total time to wait for access to the table = total time waiting for other transactions to complete their access to the table; this can be considered as the *queueing time*;
- total time to access the data = total number of disk accesses (I/O DB pages) to fetch the data into main memory × the duration of one disk access + total time to complete the operations on the data; this can be considered as the *service demand*;
- total time to return data to the client depends on the rate the data oscillates between the client and the server;
- total time to process procedural statements depends on the processing speed of the client.

Therefore, the relationship between tables and transactions can be represented as a queueing system and modelled using a queueing network. In the queueing network the tables will represent the shared resources, i.e. the servers, and the transactions that use these resources are the customers. Total time to return data to the client and process procedural statements can be aggregated for each transaction as the client think time or added as a delay resource in the queueing network.

Performance modelling of database designs is possible because transaction service demands on their relevant tables can be estimated from the procedural structure of the transaction design, i.e., from the SQL statements, the procedural statements and the structure and relationships between tables by using database query optimization techniques [8, 16, 24]. Disk I/O cost is the dominant factor [8, 11] in query execution costs, especially for large databases; this is the cost criteria that is used to calculate service demands for transactions on the relevant tables for our queueing network models. Other performance evaluation inputs, e.g., frequencies and counts of transactions, number of transaction invocations, user population, etc, are available or can be calculated from the application design [29]. In our approach, we limit ourselves to the query optimization techniques that are available to database designers in commercial DBMS, see [24], [16], or [8].

3.1 Building the Queueing Network

The steps to build a queueing network model for a database design are described next.

The Input

A database design composed of:

- Tables:
  - Structure, data types, attribute selectivity, etc.
  - Expected number of rows and record length.
  - Index types and structure.
- Transactions:
  - Rate of occurrence or % of total transactions.

- Structure:
  - SQL statements:
    - Tables accessed.
    - Join/retained attributes: sequence, selectivity.
    - Access path: can be calculated in I/O DB pages.
  - Procedural statements.

The Method

1. Specify server parameters:
   - Servers: each table in the database design is a server in the queueing network; partitioned or replicated tables are represented as separate servers.
   - Customer classes: each transaction type is considered as a different customer class: transaction types that access identical tables with equal service demands may be considered as one class.
   - Queueing discipline is FCFS:
     - DBMS use queues to control access to data objects; a new transaction is given access to a data object depending on the state of the current transactions waiting to access or currently accessing the data object. Depending on the concurrency control mechanism implemented by the DBMS, access is either granted immediately to the new transaction or it is forced to wait behind the current transactions [24]. FCFS abstracts this in forcing all transactions to wait.
     - Given that the queueing network model represents the whole database, a transaction still inside the queueing network is analogous to a transaction still accessing the database, i.e. has not committed or aborted. In this scenario, when transaction A finishes service at table X and enters table Y, any transaction entering table X, is in fact accessing table X in parallel with transaction A. Therefore, FCFS gives serial access at the transaction statement level (i.e. at the lowest granularity of access: the row level in this case), but the model gives parallel access at the transaction level.
     - In addition, queueing network models of distributed system architectures represent each database node as a FCFS queue [19].
   - Queue length is infinite: this is based on the assumption that aborts due to deadlocks are rare in DBMS [24] and system overload causes long response times instead of transaction aborts.
2. Specify performance characteristics for the customer classes:
   - Transaction service demands on each server: the total cost of executing the SQL statements in terms of I/O DB pages:
     - Service demands are assumed to be exponentially distributed with the mean being the calculated I/O DB page cost $\times$ the duration of one disk page access.
     - From [19], the number of database objects accessed by a transaction can be represented by a geometric distribution. Given that the number

of I/O DB pages accessed by a transaction is related to the number of database objects accessed by that transaction, then the number of I/O DB pages accessed by a transaction is also geometrically distributed. Since we are using the duration of one I/O DB page access as a cost metric, then the I/O service time is related to the number of I/O DB pages accessed by a transaction on a table. Hence, service times for a transaction can be approximated by the exponential distribution, which is the continuous version of the geometric distribution [19].

- Transaction arrival rates for open queueing networks and for closed queueing networks: number in system or transaction think times.

3. Specify the routing table for the customer classes; i.e. the order in which the transactions access their tables.
4. Solve the queueing network model.

The Output

Depending on the complexity of the queueing network model and the solution method used, some possible outputs are:

- For each table
  - Bottleneck resource.
  - Total access compared to other tables.
  - Mean queue length.
  - Mean waiting time to access the table.
- For each transaction
  - Mean response time.
  - Mean waiting time to access each table.
  - Response time distribution.

The information needed to build the performance model is easy to acquire during early database design phases. If the amount of detail that is available to the database designer at design time is considered, more detailed information would not be appropriate. However, this amount of information is sufficient for our performance modelling purpose.

4 Performance Analysis

The Transaction Processing Performance Council (TPC) TPC-C benchmark is an on-line transaction processing (OLTP) benchmark. It is written to be as representative as possible to actual production applications and environments. However, the TPC-C benchmark has some shortcomings in its ability to represent actual OLTP database applications and workloads: the benchmark's accommodation of known optimal memory buffering techniques cannot be replicated on real workloads [17], its workload is considerably different from actual production workloads [12], as well as its I/O reference behaviour [4, 11].

In spite of the previous shortcomings, the TPC-C benchmark is still the de facto standard benchmark for OLTP systems in industry, as well as being the only database system benchmark with published results for different software and hardware configurations. Moreover, the purpose of this work is to establish the ability to model database designs using queueing networks; thereby, for our purposes, the TPC-C benchmark's published results are just an implementation of the TPC-C benchmark database design. In this context, we believe that the TPC-C benchmark fulfils our needs and its shortcomings can be viewed as particular properties or specifications of the database system under evaluation.

The TPC-C benchmark revision 5.8.0 [33, 34] is used as an example of a database system design. The TPC-C disclosed implementations in [9], [10], [14], [15], and [5] are used as an example of the implemented database system. The TPC-C benchmark is a design specification of an order-entry system. The specification is composed of [33]:

- 9 tables (WAREHOUSE, DISTRICT, CUSTOMER, HISTORY, ORDER, NEW-ORDER, ORDER-LINE, STOCK, ITEM);
- 5 transactions (New-Order, Payment, Order-Status, Delivery, Stock-Level).

A brief description of the transactions is in Table 1. The details of the design of the tables, the relationships between them, and the details of transaction functionality can be found in [33].

Table 1
Summary of the TPC-C benchmark transactions.

Transaction	**Description**	**Min. % of the total number of transactions**	**Min Mean of Think Time Distribution (seconds)**	**Min Keying Time (seconds)**
New-Order	Initiates a new order.	No minimum	12	18
Payment	Updates the customer's balance and reflects the payment on the district and warehouse sales statistics.	43	12	3
Order-Status	Queries the status of a customer's last order.	4	10	2
Delivery	Processes a batch of 10 new orders, one for each district for a given warehouse.	4	5	2
Stock-Level	Counts the number of items in the last 20 orders in a district that fall below the stock threshold.	4	5	2

The TPC-C benchmark also includes performance specifications related to the implementation of the database system such as [33]:

- regulation of the transaction mix during the measurement period (see Table 1);
- database initial loading size: Table 2 shows the initial loading size of the database for the queueing network model used for the evaluation in [20];
- randomness and probabilities of values for the initial database loading;
- the probability of operations on the database and the probability of choosing the values of the parameters for the transactions;

Table 2
Initial loading size for the TPC-C queueing network model.

Table Name	**Cardinality** (in rows)	**Typical Row Length** (in bytes)	**Rows Per Page**[1] **(in rows)**
WAREHOUSE	150	89	24
DISTRICT	1500	95	22
CUSTOMER	4,500,000	655	4
HISTORY	4,500,000	46	45
ORDER	4,500,000	24	86
NEW-ORDER	1,350,000	8	256
ORDER-LINE	45,000,000	54	38
STOCK	15,000,000	306	7
ITEM	100,000	82	25

[1] DB page size is assumed to be 2048 bytes. DB pages are assumed to be fully loaded.

- the required performance results.

The mean keying and think times for the different transactions are specified in Table 1. In view of the fact that the benchmark is emulating a real user environment, it states that after transaction i finishes executing and returns the result to the user, the user processes that data (think time of transaction i) before choosing a new transaction and keying in its parameters (keying time for transaction *i+1*).

Table 3
I/O cost model for SQL operations.

Table Type	**Equality Search**	**Range Search**	**Insert**	**Update/ Delete**
Heap	0.5BD	BD	2D	*Search* + D
Sorted	$D\log_2 B$	$D(\log_2 B + \#\ of\ matching\ pages)$	*Search* + BD	*Search* + BD
Clustered tree index	$D\log_F 1.5B$	$D(\log_F 1.5B + \#\ of\ matching\ pages)$	*Search* + D	*Search* + D
Clustered Hash index	1.2D	1.2D ×(*#of hash keys in range*)	*Search* + D	*Search* + D
Unclustered tree index	$D(1+\log_F 0.15B)$	$D(\log_F 0.15B + \#\ of\ matching\ records)$	$D(3+\log_F 0.15B)$	*Search* + 2D
Unclustered Hash index	2D	BD	4D	*Search* + 2D

B: denotes the number of data pages in a table neglecting header information, i.e. pages are fully loaded, D: the average time to read or write a DB page, F: the tree index fan-out, here we have used the assumption in [24]: F=100

Table 4
Indexes used for the Stock-Level transaction evaluation.

Table Name	Index Type
DISTRICT	Clustered Hash Index
ORDER-LINE	Clustered Tree Index
STOCK	Clustered Tree Index

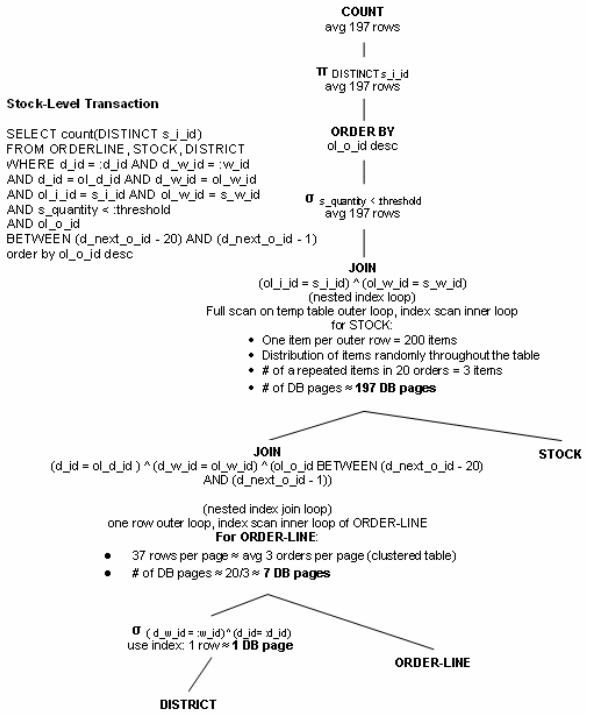

Fig. 1. TPC-C Stock-Level transaction and its optimized query tree.

4.1 Building the TPC-C Queueing Network Model

Using the transaction descriptions of the TPC-C benchmark in [33] and the order of execution, index structures and SQL statements described in [9], [10], [14], [15], and [5], in addition to the assumptions detailed in the appendix, the service demands (number of I/O DB pages) are calculated for the transactions on each table by using query optimization techniques. The Stock-Level transaction of the TPC-C benchmark implemented in [9], [10], [14], [15], and [5] will be used as an example.

Query optimization techniques use query trees as an alternative representation of relational algebra expressions, which are a translation of a SQL query. Thus, a DBMS query cost optimizer builds a query tree to represent the SQL query based on the most efficient method to evaluate the query and in turn implement the relational operators. Efficiency is measured in DB I/O pages [8, 16, 24]. Therefore, the optimized query tree provides the optimal access plan for the SQL query, in

terms of the most efficient order to access the tables as well as the number of I/O DB pages needed to retrieve the data. To complete the calculation of the service demands on the tables, table physical structure, i.e. clustering, partitioning, ...etc, and index types are used to calculate the final service demands using the formulas of the cost model described in [24] which is summarized in Table 3. Details of applying query optimization techniques to estimate query DB page cost can be found in [8, 16, 24], details of the cost model are in [24].

Figure 1 shows the Stock-Level transaction and its optimized query tree with calculated costs (DB pages) for each relational algebra operation. Assuming DBMS query optimizers use left-deep query trees [24] to decide on an execution plan for a transaction, the order of table accesses for the Stock- Level transaction will be DISTRICT, then ORDER-LINE, and finally STOCK (see assumption 1(d) in the appendix).

The number of DB I/O pages to retrieve the data on each table for the Stock-Level transaction is shown in Fig. 1. Using the optimal index structures for the tables accessed by the Stock-Level transaction described in the implementations [9], [10], [14], [15], and [5] and summarized in Table 4, in conjunction with the cost model described in Table 3, the total DB I/O page cost on each table for the Stock-Level transaction is calculated (see Table 5).

Applying the steps described above, the service demands are calculated for the other transactions, giving Table 5. Subsequently, applying the steps described in Section 3 we get the multi-class queueing network of Fig. 2. The TPC-C benchmark specifies that the transaction think times follow an exponential distribution and we have assumed that the service demands for the transactions on the tables are exponentially distributed with means as calculated in Table 5.

Table 5
Service demands for the TPC-C benchmarks.

Transactions	Service Demands (in DB pages) [2]								
	I	**II**	**III**	**IV**	**V**	**VI**	**VII**	**VIII**	**IX**
New-Order	1.2	2.2	1.2	0	2.2	2.2	7.3	7.3	4
Payment	2.2	2.2	12.76	2	0	0	0	0	0
Order-Status	0	0	15.51	0	3.45	0	4.63	0	0
Delivery	0	0	7.3	0	13.2	19.7	15.4	0	0
Stock-Level	0	1.2	0	0	0	0	10.63	200.76	0

I = WAREHOUSE, II= DISTRICT, III= CUSTOMER, IV= HISTORY, V= ORDER,
VI= NEW-ORDER, VII= ORDER-LINE, VIII= STOCK, IX= ITEM

Figure 2 is a multi-class queueing network, which was solved using simulation. The queueing network was simulated with 9 servers (tables), 5 classes (transac-

[2] These values are multiplied by the duration of one disk page access to give the final service demands used in the simulation.

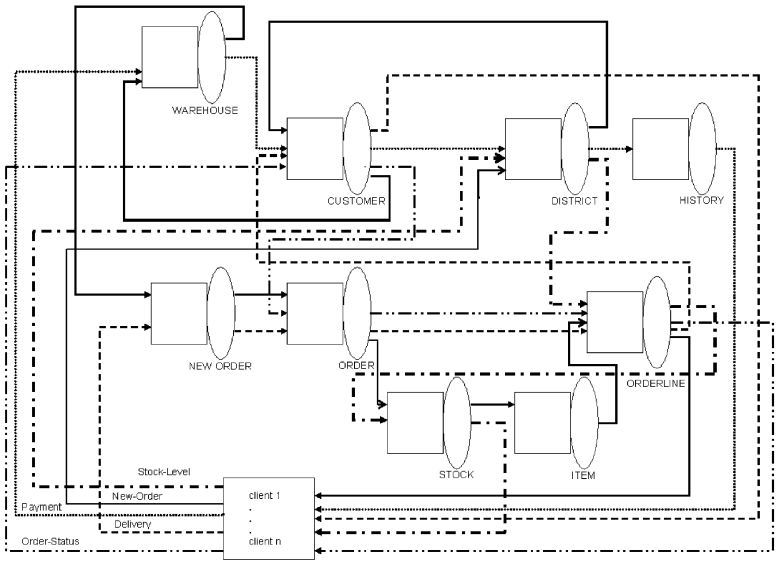

Fig. 2. Multi-class queueing network model for the TPC-C benchmark.

tions) using QNAP2, a discrete-event simulator for queueing networks [22], at 95% confidence interval. The simulation parameters are summarized in Table 6.

We have shown in [20], that the queueing network model of the TPC-C benchmark database design exhibits comparable performance behaviour and a similar transaction average response time pattern as the disclosed results of the implementations in [9], [10], [14], [15], and [5]. The queueing network model was able to capture the expected behaviour of the database transactions, using the details of the database design, the transaction DB I/O page costs, and the assumptions detailed in the appendix without considering the level of detail of the TPC-C benchmark disclosed implementations. This demonstrates the ability of the model to represent the database system.

In the next sections, we will show how a queueing network model can be applied by a database designer for the performance analysis of a database design. The work in Section 4.2 is taken from [20].

Table 6
Simulation parameters.

Simulation Parameter	**Value**
Simulator	QNAP2
Simulation time	39000 seconds
Think time distribution	Exponential
Keying time distribution	Constant
DB page I/O access time	0.00002 seconds
Confidence interval	95%

4.2 Performance Measures

The simulation of the TPC-C benchmark queueing network was run for 10, 100, 500, 1000 and 1500 clients to illustrate the model's behaviour under different load conditions. The TPC-C transaction mix is that of Table 1. Figures 3–5 show the throughput, mean response time and mean queueing time for the TPC-C queueing network transactions. Figures 6 and 7 show the mean queue length and mean queueing time for the tables (servers) of the TPC-C queueing network model. Figure 8 shows the relationships of the mean queueing times on each table for each TPC-C transaction.

As indicated by Fig. 3–5, throughput, mean response time and queueing time for the transactions increase as the load on the model is increased. This holds for the mean queue length and mean queueing time for the tables from Fig. 6–8. This result is typical of any database system: the transaction load affects performance.

Using these figures, we will demonstrate how a database designer can deduce performance indications from the database design.

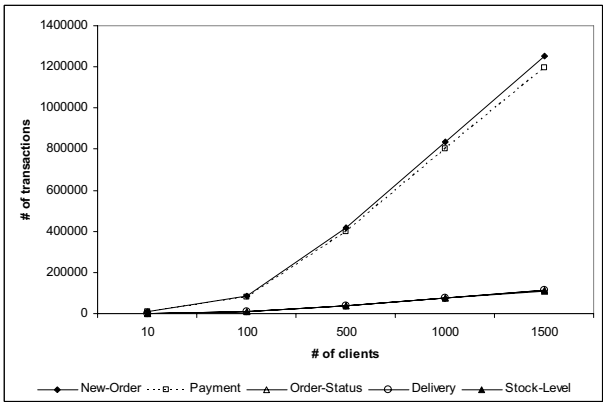

Fig. 3. Simulation results for the throughput of the TPC-C transactions for different number of clients.

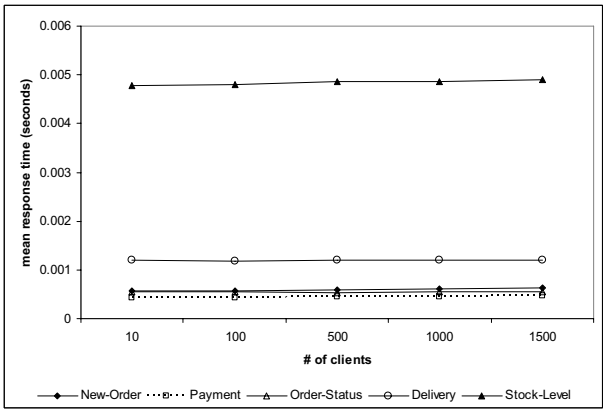

Fig. 4. Simulation results for the mean response time for the TPC-C transactions for different number of clients.

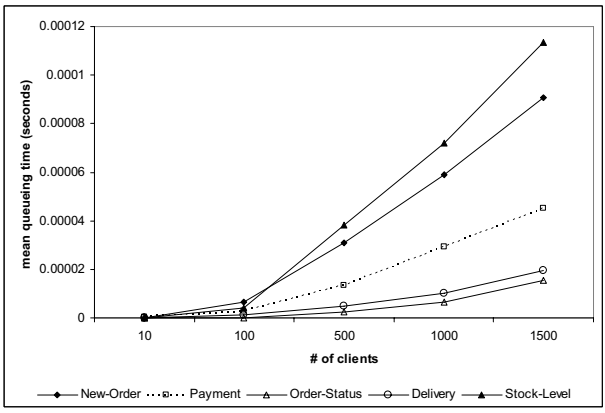

Fig. 5. Simulation results for mean queueing time of the TPC-C transactions for different number of clients.

From Fig. 4 and 5, the Stock-Level transaction has the longest mean response time and mean queueing time, even though the Stock-Level transaction has low throughput (Fig. 3), 4% of the total throughput (this is incorporated in the TPC-C benchmark specifications). This indicates that this transaction is a candidate for performance tuning, i.e. redesign.

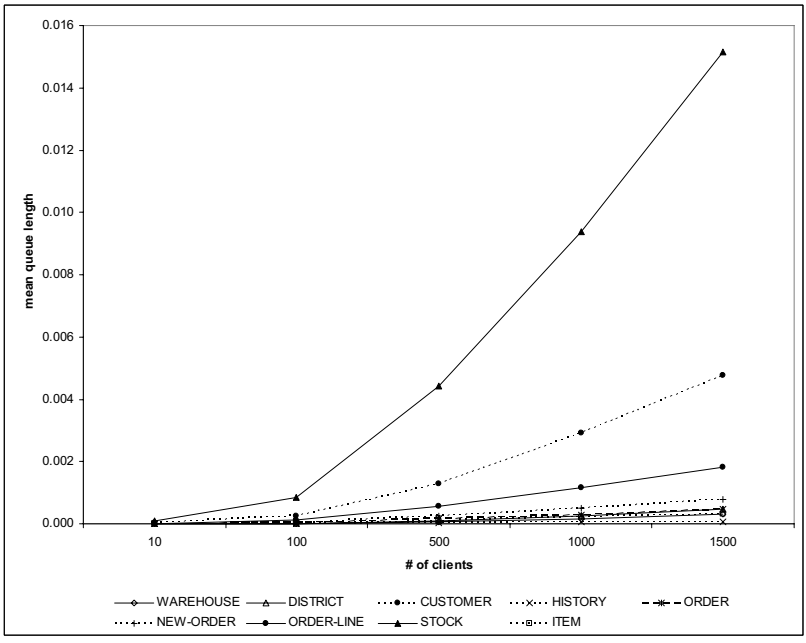

Fig. 6. Simulation results for the mean queue length on the tables for the TPC-C queueing network for different number of clients.

In addition, from Fig. 6 and 7, the STOCK table has the longest mean queue length and the longest mean queueing time, even though the STOCK table is accessed by only two transactions (New-Order and Stock-Level), in contrast to the CUSTOMER table that is accessed by four transactions (New- Order, Payment, Order-Status, Delivery). Furthermore, from Fig. 8, the majority of the waiting time

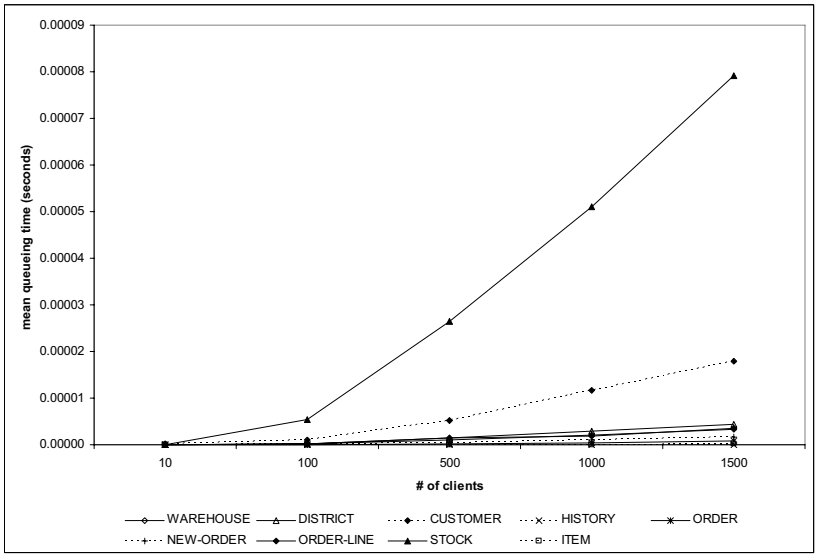

Fig. 7. Simulation results for the mean queueing time for the TPC-C tables for different number of clients.

spent by the New-Order and Stock-Level transactions is queueing for the STOCK table. Given that the TPC-C benchmark specifies the New-Order transaction as the measure of system performance [33], the STOCK table is a major bottleneck for the New-Order transaction as well as the Stock-Level transaction.

The data retrieved by the New-Order and Stock-Level transactions from the STOCK table cannot be changed due to the TPC-C specifications, i.e. these transactions cannot be redesigned and hence their service demands cannot be changed. Therefore, a redesign of the STOCK table or its access methods (indexes) would benefit the response time of both the Stock-Level and New-Order transactions. This conclusion is consistent with the TPC-C implementations that have kept the STOCK table resident in the DBMS buffer [9], [10], [14], [15], and [5], therefore eliminating I/O disk access, thus decreasing transaction response times.

Using this simple queueing network model, we have been able to conduct a bottleneck analysis of the TPC-C benchmark database design and provide performance indications of the expected database system. Due to the limited number of clients in our simulation and owing to the fact that actual data is not available for the TPC-C disclosed implementations, this result gives an encouraging indication of the applicability of queueing networks to database designs and their ability to represent database system performance behaviour.

In the next section, we use a more realistic workload to study the effects this will have on the performance evaluation of the TPC-C database design.

4.3 Bursty Arrivals

Production workloads have been shown to be bursty in comparison to the TPC-C workload model [12]. We will show how bursty arrivals influence the database design and the performance considerations that the database designer has to make

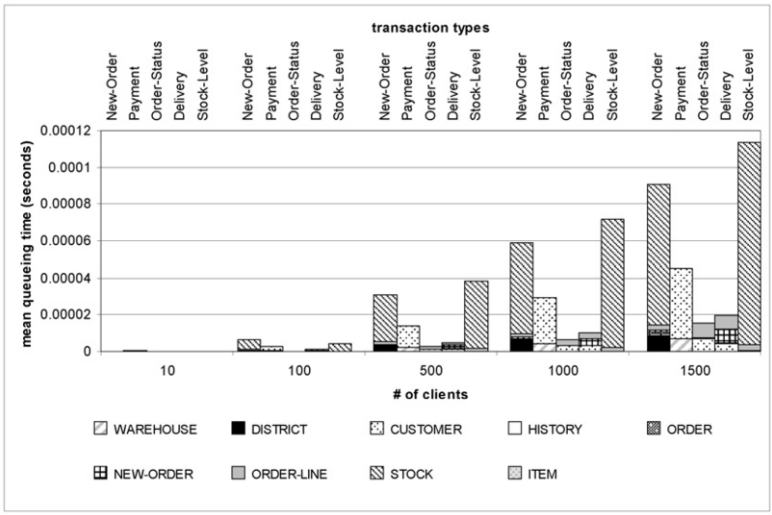

Fig. 8. Simulation results showing the relationships between mean queueing times on the tables for the TPC-C transactions for different number of clients.

to ensure that the database system performs according to its requirements.

To model bursty arrivals, a Weibull distribution was used to model the transaction think times. The constraints of having a constant keying time for each transaction and forcing the client to wait for a result before issuing a new transaction were relaxed, thus giving an open queueing network model. We also simulated the model with exponential think times only, under the same relaxed constraints. Table 7 shows the parameters for the three different simulations and the naming conventions used in the rest of this section. The think times follow the values and principles stated in the TPC-C benchmark. We will refer to the simulation results of the previous section as EXP+CST.

The performance measures for the EXP, Weibull0.7 and Weibull0.3 models were similar to the results obtained for the EXP+CST model in the previous section, except for the pattern of the mean response times of the transactions for the Weibull0.3 model. As seen from Fig. 9, the mean response time of the New-Order transaction is longer than that of the Delivery transaction, while in EXP+CST the mean response time of the Delivery transaction was longer than that of the New-Order transaction (Fig. 4). We concern ourselves with the New-Order transaction because it is the performance measure of the TPC-C benchmark.

In addition, from Fig. 10–12, which show the mean queueing time for the transactions, the mean queueing time for the New-Order transaction exceeds that of the Stock-Level transaction for all models, EXP, Weibull0.7 and Weibull0.3, as well as running the slowest in Weibull0.3. This differs from the EXP+CST model, in which the Stock-Level transaction had the longest mean queueing time (Fig. 5).

This change in the performance measures shows that the TPC-C design is sensitive to changes in the workload. This is an indication to the database designer of the necessity of having a more flexible design that performs well under different workload conditions.

Table 7
Simulation parameters for the EXP, Weibull0.7 and Weibull0.3 models.

Simulation Parameter		Value
Simulator		QNAP2
Simulation time		39000 seconds
DB page I/O access time		0.00002 seconds
Confidence interval		95%
Model 1 (EXP)	Think time distribution	Exponential
Model 2 (Weibull0.7)	Think time distribution	Weibull, shape parameter = 0.7, scale parameter = transaction think time
Model 3 (Weibull0.3)	Think time distribution	Weibull, shape parameter = 0.3, scale parameter = transaction think time

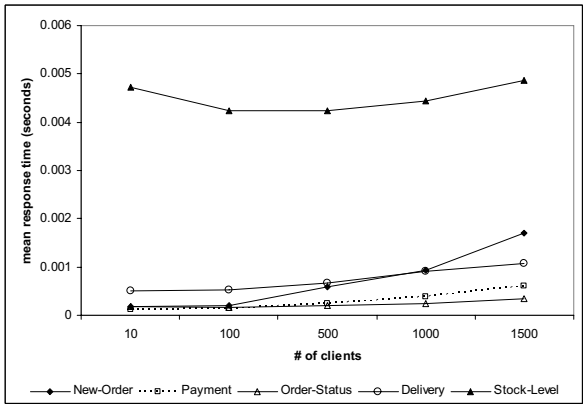

Fig. 9. The mean response time for the Weibull0.3 model for different number of clients.

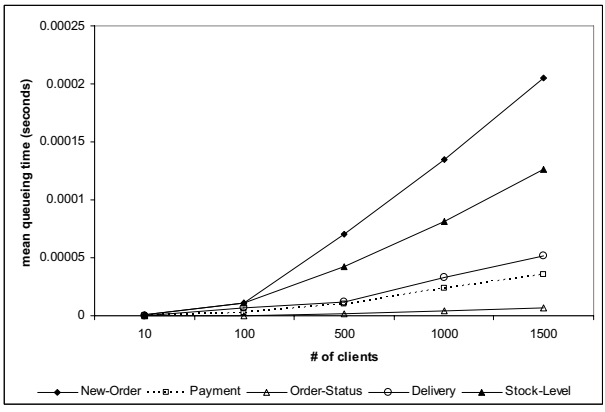

Fig. 10. Transaction mean queueing time for the EXP model for different number of clients.

A straightforward explanation of this, is that the bursty arrivals and the relaxation of the constraint of each client waiting for a reply, have caused congestion in the queueing network and therefore the performance has degraded, see Tables 8–9 for performance measures for all the models for 1500 clients. Hence, the question for a database designer would be where this congestion manifests itself and how to redesign the database with this aspect in consideration.

Now, let us look at the percentage that each transaction takes waiting at each

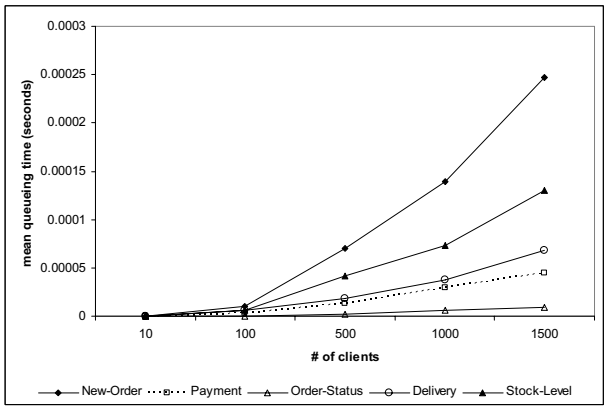

Fig. 11. Transaction mean queueing time for the Weibull0.7 model for different number of clients.

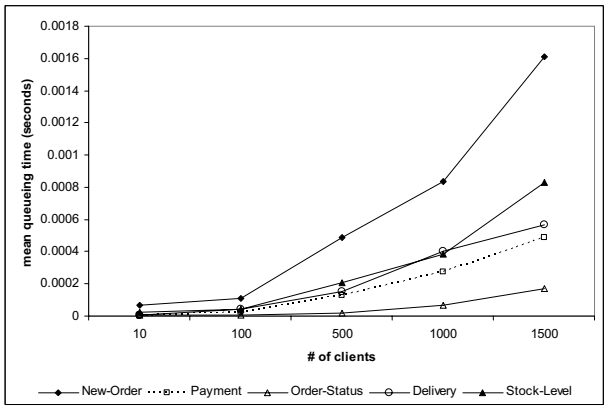

Fig. 12. Transaction mean queueing time for the Weibull0.3 model for different number of clients.

table. We will consider the results for the New-Order and Stock-Level transactions for the EXP+CST, EXP, Weibull0.7 and Weibull0.3 models with 1500 clients. From Fig. 13, the New-Order transaction is spending most of its time queueing for the STOCK table. However, as the arrival rates become more bursty, the major percentage of queueing time is steadily shifting from the STOCK table to the CUSTOMER table. This is also true for the rest of the transactions (not shown). From Fig. 14, the queueing time for the Stock- Level transaction is slowly shifting from the STOCK table as well. As the level of burstiness increases, the effect of the other transactions on the New-Order transaction is more apparent and the new bottleneck for the system will be the CUSTOMER table. Therefore, the database designer has to redesign the CUSTOMER table for more efficient access to accommodate changes in workload conditions.

5 Conclusions & Future Work

In this paper, a novel modelling technique to evaluate the performance of database designs using queueing networks was demonstrated. This application of queueing

Table 8
Performance measures for the TPC-C transactions for the EXP+CST, EXP, Weibull0.7 and Weibull0.3 models.

	Transactions	EXP+CST	EXP	Weibull0.7	Weibull0.3
Throughput (# of transactions)	**New-Order**	1254645	2320241	1834506	262852
	Payment	1196911	2215361	1750643	249127
	Order-Status	111868	206694	163350	23253
	Delivery	111568	206439	163111	23541
	Stock	111300	206068	162860	23178
Mean Response Time (seconds)	**New-Order**	0.000176689	0.000290619	0.000332629	0.001709
	Payment	0.00015162	0.00014241	0.000151834	0.000603
	Order-Status	0.000178462	0.0001686	0.000171894	0.000341
	Delivery	0.00051428	0.0005468	0.00056521	0.001081
	Stock	0.004156386	0.00417658	0.00416389	0.00487
Mean Queueing Time (seconds)	**New-Order**	0.00009085	0.000204869	0.000246885	0.001611
	Payment	0.00004507	0.00003588	0.000045178	0.000489
	Order-Status	0.000015476	0.00000712	0.00000928	0.000171
	Delivery	0.00001969	0.0000515	0.00006836	0.000565
	Stock	0.000113607	0.000126012	0.000130601	0.000831

Table 9
Performance measures for the TPC-C tables for the EXP+CST, EXP, Weibull0.7 and Weibull0.3 models.

	Tables	EXP+CST	EXP	Weibull0.7	Weibull0.3
Mean Queue Length	**WAREHOUSE**	0.00032660	0.00044400	0.00038760	0.00033110
	DISTRICT	0.00047110	0.00066900	0.00062840	0.00048960
	CUSTOMER	0.00477800	0.01080000	0.00962000	0.00736600
	HISTORY	0.00007502	0.00014000	0.00011050	0.00002160
	ORDER	0.00045520	0.00082000	0.00064830	0.00011820
	NEW-ORDER	0.00076870	0.00144000	0.00113800	0.00019560
	ORDER-LINE	0.00182400	0.00331000	0.00264800	0.00046690
	STOCK	0.01516000	0.03300000	0.02729000	0.01005000
	ITEM	0.00031870	0.00059300	0.00047040	0.00008179
Mean Queueing Time (seconds)	**WAREHOUSE**	0.00000328	0.00000193	0.00000232	0.00002260
	DISTRICT	0.00000429	0.00000261	0.00000366	0.00003188
	CUSTOMER	0.00001799	0.00003340	0.00004421	0.00046002
	HISTORY	0.00000001	0.00000002	0.00000002	0.00000005
	ORDER	0.00000091	0.00000070	0.00000066	0.00000245
	NEW-ORDER	0.00000163	0.00000170	0.00000171	0.00000438
	ORDER-LINE	0.00000348	0.00000250	0.00000305	0.00000970
	STOCK	0.00007920	0.00015500	0.00017970	0.00101590
	ITEM	0.00000013	0.00000015	0.00000014	0.00000047

networks to database designs adds the element of dynamic modelling of the database design, giving the database designer more visibility on the expected performance of the design before implementation, thereby improving database designs and reducing costly post-deployment database tuning.

We have shown that our database design performance model has the ability to evaluate expected database system performance from database designs and to pinpoint database design artifacts for performance redesign. This has been established

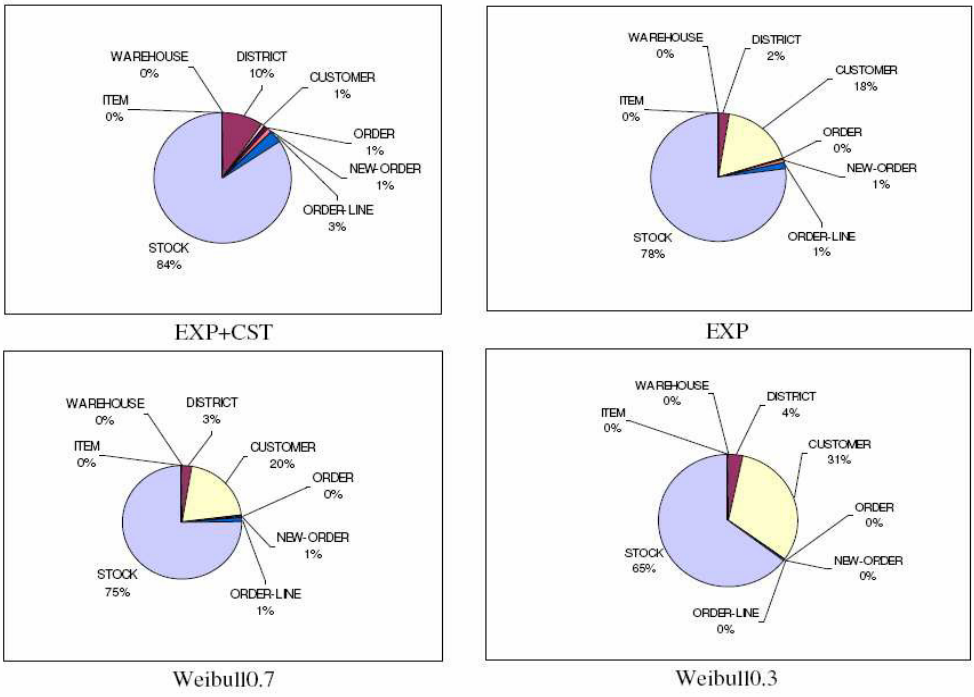

Fig. 13. Percentage of mean queueing times for the New-Order transaction for 1500 clients.

through the modelling of the TPC-C benchmark design by demonstrating how a database designer can deduce performance indications from the queueing network model of the TPC-C benchmark. In addition, the effect of bursty workloads on the expected performance of the TPC-C database design has been demonstrated.

To validate the model more accurately an implementation of the TPC-C benchmark will be used to parameterize our queueing network model, specifically the estimation of buffer hit rates and the response time of the DBMS storage subsystem. This will enable us to obtain absolute results from the queueing network model and validate its assumptions.

Locking and locking delay, even though a major contributor to database system performance bottlenecks [16, 24, 27], will currently not be considered in the refined model due to the fact that the TPC-C benchmark implementations can be implemented to minimize lock contention between different transactions and between instances of the same transaction [9], [10], [14], [15], and [5]. In addition, we have modelled the TPC-C benchmark database design queueing network model with locking but a simulation of the model did not produce any locked transactions or lock conflicts.

In addition, the applicability of queueing networks and our modelling technique to other types of database designs needs to be addressed, typically this will involve modelling other TPC benchmarks: TPC-App: an application server and web services benchmark and TPC-H: a decision support benchmark. As a final step, locking and lock conflict will be incorporated into the model.

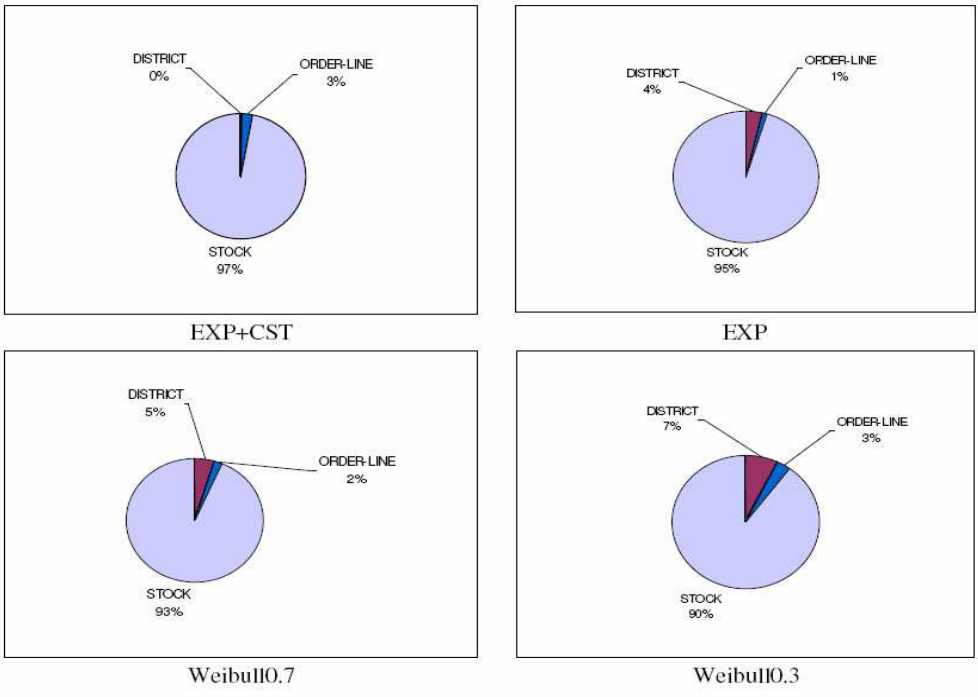

Fig. 14. Percentage of mean queueing times for the Stock-Level transaction for 1500 clients.

References

[1] Adams, E.J., *Workload models for DBMS performance evaluation*, in: *Proc. of the Thirteenth ACM Annual Conference on Computer Science*, ACM Press (1985), pp. 185 -195.

[2] Agrawal, Z., S. Chaudhuri, L. Kollar, A.P. Marathe, V.R. Narasayya and M. Syamala, *Database Tuning Advisor for Microsoft SQL Server 2005*, in: *Proc. VLDB'04*, (2004), pp. 1110-1121.

[3] Balsamo, S., A. Di Marco, P. Inverardi and M. Simeoni, *Model-based performance prediction in software development: a survey*, IEEE Transactions on Software Engineering **30** (2004), pp. 295-310.

[4] Bonilla-Lucas, R., P. Plachta, A. Sachedina, D. Jimenez-Gonzalez, C. Zuzarte and J.L. Larriba-Pey, *Characterization of the data access behavior for TPC-C traces*, in: *Proc. of the 2004 IEEE International Symposium on Performance Analysis of Systems and Software*, IEEE Computer Society (2004), pp. 115-122.

[5] Bull, "TPC benchmark: full disclosure report for Bull Escala PL1660R using AIX 5L version 5.3 and Oracle Database 10g Enterprise Edition, first edition,"(2007), URL: http://www.tpc.org/results/FDR/TPCC/Bull-Escala-PL1660oracle-FDR.pdf.

[6] Casas, I.R. and K.C. Sevcik, *Structure and validation of an analytic performance predictor for System 2000 databases*, Information Systems and Operational Research (INFOR) **27** (1989), pp. 129-144.

[7] Dageville, B., D. Das, K. Dias, K. Yagoub, M. Zait and M. Ziauddin, *Automatic SQL Tuning in Oracle 10g*, in: *Proc. VLDB'04*, (2004), pp. 1098-1109.

[8] Elmasri, R. and S.B. Navathe, "Fundamentals of Database Systems," Addison-Wesley, 2007.

[9] Hewlett-Packard, "TPC benchmark C: full disclosure report for HP ProLiant ML350 G5 using Oracle Database 10g Standard Edition one and Oracle Enterprise Linux 4, second edition," (2007), URL: http://www.tpc.org/results/FDR/TPCC/HP_ML350G5_080831_FDR.pdf.

[10] Hewlett-Packard, "TPC benchmark: full disclosure report for HP ProLiant ML350 G5 using Oracle Database 11g Standard Edition One and Windows 2003 SP1 R2," (2007), URL: http://www.tpc.org/results/FDR/TPCC/HP%20ML350G5_Oracle_070912_FDR.pdf.

[11] Hsu, W.W., A.J. Smith and H.C. Young, *I/O reference behavior of production database workloads and the TPC benchmarks-an analysis at the logical level*, ACM Trans. Database Syst. **26** (2001), pp. 96-143.

[12] Hsu, W.W., A.J. Smith and H.C. Young, *Characteristics of production database workloads and the TPC benchmarks*, IBM Systems Journal **40** (2001), pp. 781-802.

[13] Hyslop, W.F. and K.C. Sevcik, *Performance prediction of relational database systems*, in: *Proc. of the Canadian Computer Measurement Group (CMG) Conference*, (1991), pp. 298-312.

[14] IBM, "TPC benchmark: full disclosure report for IBM System p 570 Model 9117-MMA using AIX 5L version 5.3 and Oracle Database 10g Enterprise Edition, first edition," (2007), URL: http://www.tpc.org/results/FDR/TPCC/IBM_570_4_20070806_fdr.pdf.

[15] IBM, "TPC benchmark: full disclosure report for IBM System p5 570 using Oracle Database 10g Rel 2 Enterprise Edition and Red Hat Enterprise Linux Advanced Platform 5 for POWER, first edition," (2007), URL: http://www.tpc.org/results/FDR/TPCC/IBM_P570_Linux_Oracle_071005_FDR.pdf.

[16] Kifer, M., A.J. Bernstein and P.M. Lewis, "Database Systems: An Application-Oriented Approach," Pearson/Addison Wesley, Boston, 2005.

[17] Leutenegger, S.T. and D. Dias, *A modeling study of the TPC-C benchmark*, in: *Proc. SIGMOD'93*, ACM Press (1993), pp. 22-31.

[18] Menasc, D.A. and H. Gomaa, *A method for design and performance modeling of client/server systems*, IEEE Transactions on Software Engineering **26** (2000), pp. 1066-1085.

[19] Nicola, M. and M. Jarke, *Performance modeling of distributed and replicated databases*, IEEE Transactions on Knowledge and Data Engineering **12** (2000), pp. 645-672.

[20] Osman, R., I. Awan and M.E. Woodward, *Queuing networks for the performance evaluation of database designs*, 24th UK Performance Engineering Workshop (UKPEW 2008), Dept of Computing, Imperial College London (2008).

[21] Osman, R., I. Awan and M.E. Woodward, *A framework for the performance evaluation of database designs*, 23rd UK Performance Engineering Workshop (UKPEW 2007), Edge Hill University, Ormskirk, UK (2007).

[22] Potier, D., "New users' introduction to QNAP2," INRIA, 1984.

[23] Pressman, R.S., "Software Engineering: A Practitioner's Approach," McGraw-Hill Higher Education, New York; London, 2005.

[24] Ramakrishnan, R. and J. Gehrke, "Database Management Systems," McGraw-Hill, Boston, Mass., 2003.

[25] Salza, S. and R. Tomasso, *A modelling tool for the performance analysis of relational database applications*, in: *Proc of 6th Int'l Conf. on Modelling Techniques and Tools for Computer Performance Evaluation*, (1992), pp. 323-338.

[26] Sevcik, K.C., *Data base system performance prediction using an analytical model*, in: *Proc. VLDB'81*, IEEE Computer Society (1981), pp. 182 -198.

[27] Shasha, D. and P. Bonnet, "Database Tuning: Principles, Experiments, and Troubleshooting Techniques," Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[28] Smith, C.U., "Performance Engineering of Software Systems," Addison-Wesley, Reading, Mass.; London, 1990.

[29] Smith, C.U. and L.G. Williams, "Performance Solutions : A Practical Guide to Creating Responsive, Scalable Software," Addison-Wesley, Boston, MA; London, 2001.

[30] Sommerville, I., "Software Engineering," Pearson Education, Harlow, 2004.

[31] Thomasain, A., *Performance analysis of concurrency control methods*, in: G. Haring, C. Lindemann, M. Reiser, editors, *Performance Evaluation: Origins and Directions*, LNCS **1769** (2000), pp. 329-354.

[32] Thomasain, A., *Performance analysis of database systems*, in: G. Haring, C. Lindemann, M. Reiser, editors, *Performance Evaluation: Origins and Directions*, LNCS **1769** (2000), pp. 305-327.

[33] Transaction Processing Performance Council, "TPC benchmark C: standard specification, revision 5.8.0," (2006), URL: http://www.tpc.org/tpcc/spec/tpcc_current.pdf.

[34] Transaction Processing Performance Council, "Overview of the TPC benchmark C: the order-entry benchmark," (2007), URL: http://www.tpc.org/tpcc/detail.asp.

[35] Weikum, G., A. Moenkeberg, C. Hasse and P. Zabback, *Self-tuning database technology and information services: from wishful thinking to viable engineering*, in: *Proc. VLDB'02*, Morgan Kaufmann (2002), pp. 20-31.

[36] Zilio, D.C., J. Rao, S. Lightstone, G.M. Lohman, A. Storm, C. Garcia-Arellano and S. Fadden, *DB2 design advisor: integrated automatic physical database design*, in: *Proc. VLDB'04*, (2004), pp. 1087-1097.

Appendix: Modelling Assumptions

The following assumptions were made when calculating the input parameters for the queueing network model of the TPC-C benchmark design.

1. Cost Estimation Assumptions
   a. Temporary tables are completely held in main memory; therefore any manipulation of intermediate results incurs no costs.
   b. Any operations on the fetched data blocks are completely performed in main memory, e.g. an UPDATE operation after the execution of its implicit SELECT.
   c. It is assumed that all data pages are flushed from memory after a SQL statement completes its operations on the data; no caching is involved. This is a consequence of the static service demands. Therefore, the simulation does not take advantage of the skewed access property of the transactions [17, 33].
   d. Relational algebra JOIN operations are modeled on the queueing network as sequential access to the tables in the order that they are accessed by the query optimizer (this information is available on the query tree).
2. TPC-C Benchmark Assumptions

   The following assumptions simplify the TPC-C benchmark design and the input parameter calculation for the queueing network, thus simplifying the simulation program.
   a. The TPC-C benchmark states that:
      - 1% of all New-Order transactions rollback after completing all the processing steps for all items in the order, except the last item (which causes the rollback), we assume that no transaction rolls back;
      - the Payment and Order-Status transactions are invoked 60% of the time using the customer's last name and 40% of the time using customer_id. We calculated the I/O costs based on access by customer last name only (this gives on average a higher service demand on the CUSTOMER table compared to access by customer_id);
   b. The effect of the growth of the tables due to the execution of the New-Order transaction is not taken into account when calculating service demands for transactions, i.e. service demands are static. The only transaction affected by table growth is the Order-Status transaction.
   c. We use the average value for all parameters, e.g. the benchmark states that the number of items in an order is randomly selected between 5 and 15, we assume 10 items to an order.
   d. The average number of customers with the same last name was calculated through a simulation of the nonuniform random function stated in the benchmark with parameter C=1, as implemented in [9] and [14]. For a complete specification see [33].
   e. The average number of repeated items in 20 orders for the Stock-Level transaction was calculated through a simulation of the nonuniform random function stated in the benchmark with parameter C=1, as implemented in [9] and [14]. For a complete specification see [33].
   f. Time to return data to the client during the execution of the transaction is not considered.

3. Miscellaneous Assumptions

   These values are used for calculating transaction I/O DB page cost as well as input parameters for the queueing network:

   a. Database block size=2048 bytes.
   b. DB pages are assumed to be fully loaded.
   c. DB page I/O access time is 0.00002 seconds per DB page.
   d. Locking and locking delay is not considered.

The previous assumptions simplify the parameter calculation for the queueing network model, but also represent the information that a database designer might disregard at design time.