Benchmarking StrongDBMS

An Empirical Evaluation of an Immutable, Append-Only, Database

Callum Robert Alistair Fyffe
B00343094

Thesis Project for the partial fulfilment
of the requirements for the Master Degree
in Information Technology

University of the West of Scotland
School of Computing
15/08/19

UNIVERSITY OF THE
WEST of SCOTLAND

DECLARATION OF ORIGINALITY

I declare that this is an original study based on my own work and that I have not submitted it for any other course or degree.

Signature [illegible]

UWS UNIVERSITY OF THE WEST of SCOTLAND

Library Form to Accompany MSc Project

To be completed in full

Surname: **Fyffe**	
First Name: **Callum**	**Initials: CF**
Banner No: B00343094	
Course Code: **COMP11024**	
Course Title: Masters Project	
Project Supervisor: **Santiago Matalonga, Malcolm Crowe**	
Project Title: : Benchmarking StrongDBMS: An empirical evaluation of an immutable append only database	
Session: **2018-19**	**Date of Submission:** 15/08/2019
Signature: [illegible]	

Please ensure that a copy of this form is included with your project before submission.

CONTENTS

1 STRUCTURED ABSTRACT

StrongDBMS (Strong) is currently in development and has unique features that make it stand out against other well-established relational database management system (RDBMS) software. Compared to other database system providers, it has extremely strict Atomicity, Consistency, Isolation and Durability (ACID) characteristics. This work compares the concurrency and throughput performance of StrongDBMS, PostgreSQL and an anonymous commercial DBMS in an implementation of the Transaction Performance Processing Council's (TPC) TPC-C benchmark while enforcing each DBMSs definition of "serializable" isolation. The aim is to generate feedback for StrongDBMS at an early stage to aid development. The research was conducted in a combined quantitative (experimental/case study) and qualitative (a survey with database experts) manner. Results suggest poor throughput but impressive concurrency handling abilities by StrongDBMS compared to the other DBMSs. However, the results cannot be regarded as conclusive because of irregular results and a suspiciously poor concurrency performance by PostgreSQL. Despite this, the results provide the foundations for the next stage in development for StrongDBMS and highlight the potential implementation of shareable data structures in a RDBMS.

Keywords: StrongDBMS, Shareable Data Structures, Benchmarking, TPC-C

2 INTRODUCTION

The RDBMS was conceptualised by Codd in 1970 and implemented into a prototype by the late 1970s by IBM. In a RDBMS, all data is stored in relations (tables) which have attributes (columns) and tuples (rows). A primary key (PK) defines the unique and irreducible attribute of the relation and foreign keys (FK) reference candidate keys (CK) in other, or the same, relation(s). A CK is also unique and irreducible but may not have been selected as a PK due to a better candidate being available. It has a very simple, logical structure able to hold simple data and handle complex queries. The data needs to be highly independent allowing any applications using it to function while the data is being manipulated by other users of the application or other applications. Standard Query Language (SQL) is the standardized language used to interact with RDBMS's (Connolly & Begg, 2015, pp. 149 - 166). StrongDBMS is a new experimental technology and it is based on these fundamental RDBMS principles yet differs in design from conventional modern DBMSs in that it is based on shareable data structures and read-only transaction logs (Crowe, et al., 2019).

In the 1980's, benchmarking DBMSs began when businesses started to automate interactions with on-line databases that an end-user could easily follow and complete transactions. Initial benchmarks developed by companies, such as IBM, had significant flaws in that they could exaggerate a systems performance and give it more attractive results. This meant that DBMS products could not be trusted based on their performance in these benchmarks as there was no standardization or auditing performed to ensure a quality, fair test was performed on the software. To supervise and produce reliable benchmark performance claims a council between eight companies was formed called the Transaction Processing Performance Council. TPC created their first benchmark in 1989 called TPC-A and it wasn't until 1992 that TPC-C , was implemented in its first test. Early benchmarks such as TPC-A are no longer supported or trustworthy as deemed by the TPC however, TPC-C still remains a supported and trusted benchmark that companies can perform on their systems today. TPC-C still remains one of the supported and most trusted benchmarks to test the On-Line Transaction Processing (OLTP) performance of a DBMS to date (Shanley, 1998).

The motivation behind this project is to test the performance of StrongDBMS against industry established RDBMS software, with an implementation based on a widely-trusted benchmark guidelines, and to see how it compares. This was yet to be done before this paper and TPC-C was chosen because it is widely recognised within the database community. StrongDBMS has shown promising performances in preliminary benchmarks so it is appropriate to take it to the next stage.

Results indicate that StrongDBMS has, potentially, poor throughput but impressive concurrency handling abilities. The experts consulted on the results noted that, PostgreSQL had a surprisingly poor concurrency performance and the Commercial DBMS chosen had curiously inconsistent concurrency results after it had reached a maximum New Order-to-clerk ratio. Due to this, it is suspected that there may be a bug in the TPC-C implementation used that is not testing the DBMSs fairly. Regarding this, the results could not be justified as conclusive. However, the results and comments from the experts will provide a useful platform to plan future development of StrongDBMS.

The source code for StrongDBMS and its user guides as well as the TPC-C implementation for each DBMS is at https://github.com/MalcolmCrowe/ShareableDataStructures. Raw data from the experiment is found at https://github.com/cfyffe1/TPC-C-Implementation-Raw-Data.

2.1 Context

2.1.1 History of StrongDBMS

Before StrongDBMS, there was Pyrrho, a RDBMS that is strictly in accordance with SQL:2011. It is a RDBMS that is not resource intensive, so it is ideal for devices that don't have access to a lot of computing power like mobile storage. StrongDBMS is similar to Pyrrho in that it applies the same strict attitude towards ACID (Begg, et al., 2015) and also has a transaction log that is append-only (Crowe, et al., 2019). StrongDBMS builds on Pyrrho by implementing shareable data structures and immutable components. It is in the current aim of development to add support for "Big Live Data" (Crowe, et al., 2017, p. 30), roles, triggers, modules and views. To aid in this development it is important that the performance and capabilities of StrongDBMS are tested throughout its development to guide future work by means of fixing bugs, tuning parameters for better performance, and adding these new features. To do that an implementation of TPC-C has been created (Crowe, 2019d) to test StrongDBMS and other RDBMS to compare performance capabilities based on a tried and trusted benchmark.

As of January 2019, StrongDBMS has completed a TPC-C benchmark with a result of 40 New Order measurements per second . As more features are being added to StrongDBMS (Crowe, 2019a), it is appropriate to test the latest version to ensure that features and updates to the RDBMS are improving the capabilities of the system without being detrimental to the performance. Features can be added, modified or dropped based on the desired performance after benchmark evaluation. These trade-offs will play an important part in how the DBMS develops.

2.1.2 TPC-C Justification

TPC-C is widely recognised as a successful benchmark with proven longevity (Baru, et al., 2012). A desirable feature of big data benchmarks is some form of scaling. TPC-C enforces this as the user cannot generate massive transactions per minute (tpmC) rates while keeping the database small as the benchmark requires the user to add another warehouse for every 12.86 tpmC (Transaction Processing Performance Council, 2010, pp. 61 - 68). This scale factor is to prevent vendors from manipulating their TPC-C scores to look more favourable to the market. Although the scalability rules are practiced in official TPC-C specifications, this limitation is not implemented in the variation used for this project as there is no agenda to put StrongDBMS on the market while in such an early stage of development. The scaling factor can be viewed as a consideration for future work on the TPC-C application that is being utilised for this project.

2.2 Problem Statement

StrongDBMS has yet to be thoroughly evaluated against established software. To achieve this, the technologies underpinning StrongDBMS must be characterised and its performance benchmarked against industrial solutions. The advantages and disadvantages of StrongDBMS and its competitors must be researched to see how these will affect the results of a TPC-C benchmark. The TPC-C benchmark must be implemented in a way that fairly tests each RDBMS and produces comparable results. The results must be compared, and areas of work identified, to improve StrongDBMS.

StrongDBMS, version 0.0, has already had preliminary tests against an implementation of the Transaction Processing Performance Council's (TPC) TPC-C benchmark, but it has had a number of new features added to it since then. These new features may slow down its transaction processing time. This benchmark is a good indicator of whether a database can perform adequately in a situation that requires a large amounts of data generation and gathering, can handle multiple transactions, and can demonstrate strong ACID properties. TPC-C is widely recognised amongst the database community so its results will be easy to interpret. Strong and two highly regarded RDBMS products will be tested in identical tests to compare results and gauge the development of StrongDBMS. StrongDBMS version, 0.1, has yet to be tested and it will be used as feedback in the development of the software. The DBTech community will be asked to comment of the validity and value of the results. This project will demonstrate the advantages of a RDBMS that has been designed differently to the most commonly used database software.

The benchmark tests are being done at this stage of development to evaluate how StrongDBMS's approach to immutable data structures behaves compared to establish DBMSs. This is important as early feedback is desired at this developmental stage.

2.3 Research Objectives

- Literature research must be conducted into the documentation of the TPC-C benchmark, StrongDBMS, shareable data structures, the other benchmarked RDBMS and previous benchmark attempts on them. Other benchmarking methods other than TPC-C will also be investigated as a consideration for future work to be carried out. Future work considerations are important as this is part of the developmental cycle.
- The aim is to understand the background of StrongDBMS and compare ACID characteristics and performance of it against more popular RDBMS. The under-development StrongDBMS will be evaluated to see what steps need to be taken to improve it. As more features are added it will be retested to determine the effect of new features on performance. This comparison can also be used to highlight strengths and weaknesses of the RDBMSs and recommend their use for specific situations. This research will be integrated into the development of StrongDBMS rather than a finished product summary. This is being done with the collaboration of Academics form the University of the West of Scotland (UWS).
- The DBTech community and database experts will be consulted for validation of the research methods and results. This will be presented to them in an email and provide guidance with open ended questions. The questions can be developed once the data from the benchmarking process is available.
- After the results are analysed, and the DBTech comments are mined, the literature will be revised, and suggestions will be made for the future development of StrongDBMS.

2.4 CONTRIBUTIONS

The work done in this project is being used to feed back into the development in StrongDBMS. Preliminary benchmark results were contributed to Prof Malcolm Crowe's keynote speech at the Ninth International Conference on Smart Grids, Green Communications and IT Energy-aware Technologies (IARIA, 2019). The TPC-C benchmark application for PostgreSQL was adapted in this project from other TPC-C applications created by Prof Malcolm Crowe to comply with PostgreSQL's SQL syntax and server connection settings. The raw data obtained from the TPC-C implementation tests are accessible on GitHub (https://github.com/cfyffe1/TPC-C-Implementation-Raw-Data) for critique or replication of the study carried out. The findings of this project will also be made available to the DBTech community after their comments have been considered.

3 Literature Review

3.1 StrongDBMS

StrongDBMS is an experimental SQL RDBMS that is currently in development at the University of the West of Scotland (Crowe, 2019a). The transaction log is append-only allowing for admin and user transparency. In Strong, Shareable data structures are set up so that when a new data structure is created, the old data structure is not destroyed (immutable). This type of database structure design means you can access the older versions of the data structure which is known as persistent. The structure is partially persistent if you can only make changes to the current version of the data structure while maintaining access to older versions. It is fully persistent if you can modify every version and have access to them as well (Driscoll, et al., 1989). StrongDBMS currently supports partial persistence as the previous state of the database can be reconstructed using the Log table and truncating the transaction log. This could be utilised by a client that requires transparency with its data so that any appends can be traced back to a user. This discourages any users from manipulating the data in a malicious way as they won't be able to cover their tracks. Common features in RDBMS, such as users, permissions and roles, are in progress of being added to StrongDBMS. As it is in its developmental stage that should allow it to compete on a performance level with established RDBMSs in a benchmark test. It is open source and free to use as long as original authorship is acknowledged. (Crowe, 2019a; Crowe, 2019b).

3.2 Brief Overview of Commercial DBMSs

Oracle was the first commercially available RDBMS in the world. It is the world's most popular DBMS (solid IT, 2019) and has been building on the foundations of their earliest RDBMSs that have been released since 1979. It has many different commercial editions that can be customised to suit each client for a licence fee, or it has an Express edition that is free (Oracle, 2019b). Oracle's newest version of their DBMS is 19c, which was released in February 2019 (Giles, 2019).

Microsoft (MS) SQL Server was released in 1988 as a joint development between Microsoft and Sybase before Microsoft took over sole ownership in 1994 (Spenik & Sledge, 2003). Companies based in the United States are the main users of SQL Server, such as M.I.T., Stack Overflow and intuit. It is suited to e-commerce and data warehousing environments (Stackshare, 2019b). MS SQL Server's most recent version, 19, was released as of 2018. This new version boasts new machine learning features for the supported R and Python languages. It also builds on previous versions to give more choice when selecting things such as data types, the RDBMS can be implemented on premises or cloud, and it also has multiple operating systems that it can run on (Microsoft, 2019b).

PostgreSQL has the self-proclaimed accolade of being the "world's most advanced open-source database" that was initially developed, at the University of California at Berkeley in 1986, under the name "POSTGRES" (PostgreSQL Global Development Group, 2019a, p. xxxi). This is a well-deserved title as there a number of large companies such as Spotify, Netflix and Uber that use its services (Stackshare, 2019a). The current version is 11.3 with version 12 currently in beta.

3.3 Transaction Processing Characteristics of a RBDMS

For a DBMS to be effective in its job, it must have certain characteristics of Atomicity, Consistency, Isolation, and Durability or ACID for short (Feuerstein, 2014, pp. 461 - 484). The principles of ACID refer specifically to the way a DBMS handles transactions. Transactions are multiple SQL commands written together in a script or queued on the client application to be executed all together. These can come in the form of Data Manipulation Language (DML) such as INSERT, DELETE, UPDATE, etc.; queries using SELECT; and extensions of SQL (like PL/SQL for Oracle and T-SQL for SQL Server) such as functions, procedures, etc. It is important to understand these properties to effectively compare the results of a benchmark that exploits the way a RDBMS handles them. A transaction has Atomicity when it entirely commits at the same time or none of it is committed at all. Data has Consistency when the data is altered from one consistent state to another by abiding by the constraints of the schema. Isolation is when transactions must occur independently of each other or appear to execute in a serial order i.e. transactions occur before and after each other so two transactions won't overlap at the same time. If a transaction has Durability, it will survive any failures that occur to the system such as a loss of power and will not currupt the data as a result. So if a transaction is only partially executed when the system fails it will not be permenantly written to the system.

It is important to execute transactions and queries at a serializable level to prevent certain phenomena occurring. When data is read before it has been committed, it is known as a dirty read. This can happen up to an isolation level of read uncommitted. When a non-repeatable read occurs, the data displayed in the read has been altered so it is impossible to read the data again and get the same result. This can occur up to a read committed isolation level. A phantom read is when a query is run again by a transaction that finds additional committed data since the first query read. This is possible up to a repeatable read isolation level. None of these mentioned phenomena can occur at the SQL defined Serializable isolation level (Oracle, 2019a, PostgreSQL Global Development Group, 2019a).

Oracle (2019a), Fekete, et al. (2005) and Feuerstein (2014, pp. 461 - 484) demonstrate that isolation levels can be customised so that transactions can read uncommited data but cannot have resources locked. This allows multiple users to access the data without having to take turns. If the data is at a lower level of Isolation, it is also at a lower level of Consitency. The Isolation setting configurations for the considered RDBMSs are as follows:

- READ UNCOMMITTED which will read rows that are in the process of being altered by another transaction and also allows the data in rows to be altered between reads while a transaction is occuring. This is a low level of Isolation and can provide inconsistent results.
- READ COMMITTED (Oracle, PostgreSQL and SQL Server default) will avoid reading rows that are locked in another transaction until that transaction is finished and unlocks the row and will only read data that is committed before any query has started.
- REPEATABLE READ locks data so that other users cannot alter it during the time a transaction is accessing it but it does allow for new data to be inserted while this transaction is occuring. In PostgreSQL, phantom reads are not possible in this mode due to PostgreSQL implementing stricter behaviour than the SQL standard for Isolation level definitions.
- READ ONLY (Oracle only feature) is comparable to SERIALIZABLE but the 'sys' user (admin) can make changes to the data in Oracle.
- SNAPSHOT ISOLATION avoids all the previously mentioned phenomena and can provide the DBMS with a better concurrency performance than serializable. However, it does not avoid all possible phenomena. Takes a "snapshot" of the data at the beginning of a transaction and allows concurrent users to read it without conflict.

- SERIALIZABLE (Strong's only Isolation setting) prevents any DML statement from changing a table that has already been altered by an uncommitted transaction. This is the highest level of Isolation.

Table 1 shows which phenomena are possible at each isolation level with an 'O' to mark possible and a 'X' to mark not possible. It is reminiscent of tables displayed in RDBMS documentation (PostgreSQL Global Development Group, 2019a, pp. 427, Oracle, 2019a, Microsoft, 2019a) with the addition of the snapshot isolation row and write skew column:

Transaction Isolation Level	Dirty Read	Non-repeatable Read	Phantom Read	Write Skew
Read Uncommitted	O	O	O	O
Read Committed	X	O	O	O
Repeatable Read	X	X	O	O
Snapshot Isolation	X	X	X	O
Serializable	X	X	X	X

Table 1: Possible Isolation Level Phenomena

Academic attack tests like ACIDRain (Warszawski & Bailis, 2017) prove that there is a need for strict serializable behaviour in web applications. This attack exploits web-based applications that do not implement serializable transactions and have setups that can be compromised by accessing their functionality over the internet through HTTP and other protocols. This means that the attacker doesn't need to hack into the application's servers and can take advantage of vulnerabilities from publicly available protocols and data. ACIDRain is a concurrency-based attack that can be used in certain scenarios such as in an online shopping cart. Warszawski & Bailis (2017) demonstrate one of many vulnerabilities that are present in non-serializable web applications. An online shopper can pay for the items in their cart at the same time as adding another item to the cart. This will allow the attacker to essentially add this extra item for free as the application will see the cart has been paid for even though the concurrent item has not.

3.4 Treatment of Serializable Transaction Level

Snapshot Isolation (SI) offers users the ability to achieve a higher throughput rate than serializable isolation. Oracle, SQL Server and PostgreSQL all allow this setting and Oracle even employs it as its own definition of serializable instead of applying true serializability to its transactions (Laiho & Laux, 2011, pp. 23 - 25). This is because the transaction only allows a snapshot of the already committed data, in the state it was in as the transaction began, to be viewed by the user. Fekete, et al. (2005) argue that TPC-C will execute in a serializable manner despite Oracle technically only achieving snapshot isolation levels. None of the previously mentioned phenomena can occur on snapshot isolation. However, it does not technically conform with the traditional textbook restrictions of serializable. It still allows for a phenomena called write skew. This is where data is corrupted when different programs mix signals by swapping between them (Alomari, et al., 2008).

TPC-C only specifies that the dirty write, dirty read, non-repeatable read, and phantom read do not occur in the business level transactions (Transaction Processing Performance Council, 2010, pp. 51 - 56) of a TPC-C benchmark test. This allows for Snapshot Isolation to be used in a TPC-C benchmark and execute in a serializable manner (Alomari, et al., 2008), MS SQL Server describes Serializable as “restrictive” in terms of performance at higher levels of concurrency (Microsoft, 2019a). To ensure serializable behaviour MS SQL Server employs locking on its data. This blocks other users from altering data in a way that would cause issues for the lock owner. SQL Server also provides Row versioning at lower isolation levels such as SI and Read-Committed, to improve concurrency. This however, does not ensure serializable behaviour so cannot be implemented at serializable isolation level.

Abadi & Faleiro (2018) discuss concurrency control with Multi-Version Concurrency Control (MVCC). Modern DBMSs usually implement data storage in a multi-versioned format. This means that when data is updated a new version is written to disk, the old version(s) is/are available for read purposes. This method lacks consistency and it cannot guarantee serializable behaviour. Serializable MVCC guarantees serializable behaviour but restricts concurrency between read and write transactions that conflict. This limits the advantage of having multiple versions as they are used ineffectively when serializable behaviour is enforced. PostgreSQL maintains its data using a MVCC model rather than implement conventional locking which can cause read-write conflicts. PostgreSQL can guarantee serializable behaviour through serializable snapshot isolation (SSI) (PostgreSQL Global Development Group, 2019a, pp. 426 - 441). It is also stated that by using MVCC, PostgreSQL will only demonstrate “reasonable” performance when multiple users are accessing the system (PostgreSQL Global Development Group, 2019a, p. 426).

Oracle and SQL Server can also implement their own version of MVCC as a form of concurrency control although Oracle states that it is not turned on by default, is only supposed to be used with Snapshot Isolation (Oracle, 2019a) and cannot guarantee serializable isolation level without locking. SQL Server refers to MVCC as “Row-Versioning” (Microsoft, 2019a) and also does not rely on it at the highest isolation level as it uses two phase locking to achieve a serializable isolation level. Some documentation and literature (Microsoft, 2019a) still refer to MVCC as optimistic however it doesn’t abide by the true meaning of optimistic concurrency control (OCC) (Laiho, et al., 2013, p. 25). In terms of read dependency, a transaction can be referred to as optimistic because MVCC will allow the data to be read and will not produce any read conflict errors. Technically this is a more “optimistic” approach than locking everything as would be done in pessimistic concurrency control. However, it is not true OCC when it comes to write conflicts as the DBMS will still need to employ locking to achieve serializable isolation.

Oracle also makes use of locking, while operating serializable transactions, that is applied to a fine granularity within the data as well as implementing queries that are non-blocking to ensure that write-write contention is reduced (Oracle, 2019a). Oracle can generate a greater throughput rate against competitors that encounter a lot of read-write contention. However, Oracle also warns developers about the resource expensive process of rolling back and retrying transactions while in serializable isolation mode. This can cause high conflict when there is a lot of contention. Oracle also recommends against long transactions as they are unlikely to be the first transaction to alter a row in any given table. This means that they are more likely to be rolled back and a lot of time is wasted trying to commit the long transaction.

StrongDBMS has a strict definition of Serializable and it is its only transaction level isolation setting. It also employs a type of MVCC as each transaction cannot see any other concurrent transactions until the commit stage. This differs from other DBMSs which will try to lock data at the update stage of a transaction until it commits (Crowe, et al., 2019). It allows a user to attempt to manipulate data that is already in mid-transaction by another user but will only allow one user to commit. The loser in this situation will have their transaction rolled-back (Crowe, 2019a). This is a true optimistic approach to concurrency compared to the pessimistic locking approach implemented by PostgreSQL, Oracle and

SQL Server at a serializable isolation level. This was adopted by StrongDBMS from its parent research project, Pyrrho, which also implements serializable as its only isolation level, and practices true OCC (Crowe, 2015, pp. 7 - 13). OCC, when properly implemented, does not use any locking so the DBMS will not experience deadlocking and does not commit when presented with concurrency conflicts. This means that any exceptions will be triggered at the commit phase whereas with locking they can occur when starting a transaction (Laiho & Laux, 2011, pp. 33 - 37). These established DBMSs use the pessimistic concurrency control of locking to deny access to any data in use by a transaction. StrongDBMS does not require any transactional locking, however locking is implemented on the actual DBMS and the file for each database is also locked (Crowe, 2019c). This optimistic approach means that first committer wins. In locking, the first person to begin a transaction using the data wins by locking the affected data. This can lead to delays in data entry as a transaction can lock data for long periods of time unless a timeout period is set for transactions. StrongDBMS's optimistic approach may be more frustrating for concurrent users as there is no guarantee when the user starts their transaction that they will be the first to commit. However, it does encourage more efficient data entry at higher levels of concurrency.

3.5 TPC-C

The Transaction Processing Performance Council (TPC) is an industry recognised standard for performing benchmark testing and has been doing so since formation in 1988 (Transaction Processing Performance Council, 2019). There are several active TPC benchmarks, however, TPC-C is best suited to testing a new RDBMS as it tests fundamental performance aspects that would be required in a successful business environment. TPC-C is a benchmark that applies OLTP. Chen, et al. (2012) explain that TPC-C is based on requirements found across many use cases to test the tables, population, transactions, and scheduling of a DBMS or application. Tables will be tested on their layout and relationships. The data populating the tables starts off with a foundation layer of data that replicates how data would appear before the working day begins in a business, then the data develops gradually to mimic business conducted during the working day. Transactions replicate functions of the business, where data in the tables would be computed against, input variables would be spread, and the transactions would interact with each other. Scheduling replicates activity within a business where the rate and type of transaction is varied.

TPC-C replicates an industrial environment by flooding the DBMS with new data. The benchmark simulates a business that is a wholesale parts supplier with several warehouses, each having 10 sales districts, and each district has 3,000 customers that it must serve. Each warehouse has 10 terminals that every type of transaction/operation can be performed on. Each warehouse must attempt to keep stock for 100,000 products and complete order requests from the available stock. It also realistically simulates that each warehouse won't have all the items necessary to complete all its orders, so it requires that at least 10% of all orders require products from a different warehouse if there is more than 1 warehouse. Operators from these districts create 1 out of 5 available transactions/operations to be processed. TPC-C tries to deliver these in a way that replicates a real-world scenario in that each transaction/operation is delayed a certain amount of time. This simulates an operator typing out the data that is needed for a new transaction/operation or reading the data from a previous transaction/operation. The results are reliant on the DBMS demonstrating capable ACID properties as well as backup and recovery features (Transaction Processing Performance Council, 2019).

One important output variable from TPC-C is the number of New Orders that a DBMS can generate per minute which is measured in tpmC. A New Order is referred to as a business transaction whereas other transactions are referred to as database transactions. Database transactions are simple units of work processed with ACID characteristics. The business transaction is important because it is a transaction that is executed frequently, requires to read and write, and it places variable workloads on the DBMS to simulate a similar environment to that of real-world businesses. There are other business type transactions in a TPC-C benchmark such as Payment, Order-Status, Delivery and Stock-Level. These transactions must occur while New Orders are being processed. Out of all these business transactions, New Order is the only transaction that does not have a minimum restriction in regard to the required transaction mix. This is because it is used as a throughput measurement whereas the others are not. This is why the number of New Orders produced in 10 minutes is considered a worthy measurement.

Poess (2012) discusses that the TPC are recognized as one of the top three associations for industry standards in terms of database benchmarking. The benchmarking standards are approved by the council under the guidance of strict voting regulations that make development of benchmarks slow but ensure that they have longevity. The advantage of using a benchmark that has been around so long is that it allows users to perform a long-term study of various versions of software. This allows such studies to be easily comparable and repeatable. As the DBTech community are going to be consulted on the results of this project, it is best to use a widely recognised benchmark as they will more than likely already be familiar with TPC-C-like benchmarks within academia and industry. This will translate well and make the results easier to evaluate by the community. Alexandrov, et al., (2013) and Nambiar, et al. (2012) also argue this point of view. They demonstrate that it is better to use well-known benchmarks such as TPC-C to evaluate the performance of DBMSs. This is because the specifications are well known to the database community as it has been around for over 30 years, so, the experiment translates well to the community and it is trusted by them. It is also open source meaning that it can be implemented by third parties with ease. This takes away the need to generate a concept of an original benchmark and prove its validity.

Chen, et al. (2012) explain that there were no standardized big data performance benchmarks as of 2012. However, they propose, as TPC-C is a widely recognised standard for on-line transaction processing (OLTP) that it would be a good starting point for developers to begin designing a standardized big data benchmark. While big data benchmarking is not the goal of this project, it is worth noting that future benchmarks are likely to be derived or similar to TPC-C. This implies that past implementations could be altered, with minimal difficulty compared to implementing an entirely new benchmark, to test DBMSs that have not been benchmarked for future database needs such as big data.

An official TPC-C benchmark would be very expensive to run as indicated by Oracle (2013) and SAP (2014). It is very common in academia and research fields to implement a variation of TPC-C to validate projects to a certain extent. The implementations are usually altered in a way to test specific aspects that an official TPC-C may not test to a thorough extent. TPCC-UVa by Llanos (2006) excludes the pricing performance measurement implemented by TPC-C. CH-benCHmark by (Cole, et al. (2011) is another example of unofficial TPC-C implementation where it is combined with aspects from the TPC-H benchmark, also by the TPC.

4 RELATED WORK

4.1 VARIATIONS FROM OFFICIAL TPC-C

The TPC-C implemented in these benchmark tests varies in a number of ways from the official TPC-C benchmark. In this particular adaption of TPC-C there is only 1 warehouse. An official TPC-C benchmark may run with 1 warehouse, but it can incorporate several to test the ability of a DBMS to handle large amount of data and allows for more complex transactions to take place. This TPC-C implementation is run over the course of 10 minutes and the amount of New Orders that are processed in the 10 minutes are used to evaluate which RDBMS has superior concurrency handling capabilities in a saturated, data warehouse environment. In TPC-C there is a method for calculating the price of running the database for 3 years, this has been discarded for these benchmark tests. Since the TPC-C implementation is used mainly for concurrency testing there is little justification for running it at TPC-C's required minimum duration of 120 minutes. 10 minutes is deemed a sufficient amount of time to analyse conflict within each RDBMS. In TPC-C, each warehouse has 10 terminals that request transactions whereas, in the TPC-C implementation, terminals are referred to as clerks. The number of clerks is a variable rather than a fixed number to test the transaction conflict resolution and concurrency capabilities of the DBMS that is being tested. The TPC-C implementation does not take an average throughput measurement by TPC-C standards. This is because the number of New Orders generated includes New Order rows acquired in the "Ramp-Up" phase which occurs upon initialisation. This "Ramp-Up" phase is where tpmC are still increasing due to the TPC-C application "warming up". A steady tpmC is needed to measure throughput which occurs after "Ramp-Up" and before "Ramp-Down", in a period of time where the test is generating a fairly consistent throughput (Transaction Processing Performance Council, 2010).

It is likely that the benchmark tests will create throughput rates above and below that of the accepted TPC-C margins of 9 - 12.86 tpmC per warehouse because of high concurrency. In an official test this is prevented by scaling-up or down the database and adding/subtracting warehouses so the tpmC does not escape this margin (Transaction Processing Performance Council, 2010, pp. 61-68). In official TPC-C this is implemented to prevent vendors from making their setup appear favourable compared to others on the market. This variation's purpose is not to test scalability so the tpmC margin will be ignored. The tpmC may initially appear exaggerated as the 10-minute New Order Concurrency tests produce results 10 times greater than tpmC as it is over 10 minutes.

This TPC-C implementation sets the transaction isolation level to each DBMS's definition of serializable for every transaction whereas the official TPC-C only specifies that certain phenomena cannot occur and doesn't strictly demand isolation level to be set at serializable (Transaction Processing Performance Council, 2010, pp. 51 - 56).

As seen later on in consistency checks (Table 7), the TPC-C variation also does not update all the tables in the schema. This is because the implementation is simple and only designed for concurrency and throughput performance measurements as well as basic ACID testing.

4.2 BENCHMARKING

Benchmarking can be described as a set of executions to be performed on a database under a certain data model. The results of benchmarking can include: contrasting the performance of different systems whilst implementing a set of initial conditions, or, comparing different ways of designing the data structure (i.e. hierarchal, star, network) and studying the impact they have on the performance of the system (Darmont, et al., 2017). Benchmarking can be done in many ways, it just depends what aspect of the system's performance you want to measure. In terms of RDBMS, common benchmarking practices include measuring throughput, query response time, and testing the DBMS in a scenario that would be similar to how the DBMS would be applied in an industrial, commercial or business setting (Fiannaca & Huang, 2015).

Cole, et al. (2011) demonstrate CH-benCHmark, which combines the structure of TPC-C and TPC-H. TPC-H is a benchmark designed for decision support, particularly on systems that examine vast quantities of data, execute highly complicated queries, and answer questions important to business, Business Intelligence (BI). Online Analytical Processing (OLAP), instead of OLTP, is the application domain for TPC-H. Combining this with TPC-C allows the benchmark to assess the effectiveness of a DBMS to handle OLTP and OLAP on the same tables at the same time.

Alomari, et al. (2008) argue that TPC-C does not compare the ways that serializable isolation level effects certain DBMSs. This is because TPC-C will run transactions that are technically serializable on systems that can only support snapshot isolation (SI). They propose a benchmark called SmallBank that allows the user to modify the schema to allow serializable transactions to occur on SI.

IBM Software Group Information Management (2009) developed Telecommunication Application Transaction Processing (TATP) which is a benchmark that tests throughput capabilities of a DBMS in a simulated telecommunications environment. It floods the DBMS server with data until the maximum throughput, that the server can sustain, is found.

Difallah, et al. (2013) show that OLTP-Bench is an open-source benchmarking testbed. It can be used in support of "all major DBMSs". It has an implementation of TPC-C available to use however, this implementation does not include the time delay to simulate thinking and reading of a clerk that is required for a strict TPC-C benchmark. There are other benchmark implementations in the OLTP-Bench library that are used to evaluate very specialised use of RDBMSs. Since StrongDBMS focuses on strict ACID properties, transactional benchmarks are most relevant:

- AuctionMark has been specifically designed to simulate an on-line auction type environment. This tests the performance of a DBMS when handling large numbers of tables that cannot be denormalized easily. The transactions are all non-deterministic due to the unpredictable nature of an auction.
- SEATS simulates an airline ticket database system where users search for flights and reserve seats on-line. The test takes requests from different applications that each provide a different form of input to the other, such as, logging in using a user name, customer ID number, or a frequent flier number. This means the DBMS has to rely on heavy use of foreign keys or secondary indexes to find the primary key of the customer.
- The testbed also includes the previously mentioned SmallBank, TATP and CH-benCHmark.

As this is the first series of benchmark tests on Strong it would make sense to use TPC-C as it will test the core attributes of Strong as well as being a reliable and industry recognised benchmark. This will make interpretation by the wider community relatable and easy to grasp as they are more than likely already familiar with the format of a TPC-C benchmark test. Specialised benchmarks may be more applicable later on in its development to fine tune or add features that are desirable.

4.3 Previous Benchmark Tests on RDBMSs

Most current research papers on TPC-C benchmarks focus on improving the performance of a RDBMS by means of altering logical server architecture, altering database structure, rather than performance comparison between SQL DBMSs.

Shao et al. (2015) carried out a TPC-C benchmark test on SQL Server (Version 7.0) to determine whether a certain hierarchal server structure can improve the performance of a TPC-C benchmark. In TPC-C, scaling is determined by the quantity of warehouses in the database (Transaction Processing Performance Council, 2010, pp. 61 - 68). They refer to the scaling of the database used which is either 30 or 60 compared to the one that is proposed for Strong. In this experiment, the number of clients/clerks numbers up to 2000 compared to roughly 100 that is expected for the proposed Strong experiment. This particular paper focuses on the number of transactions that can be processed per second as a measure of performance rather than the number of successful and unsuccessful transactions. It also isn't clear what number of clerks are run for particular tests in this paper. From this, it is acknowledged that any tables and graphs must be clearly labelled to make interpretation by others easy.

It is worth noting transactions per second and tpmC (throughput), a standard unit of measurement for TPC-C, are the most common units of measurement that many research papers use (Shao et al. (2015), Tongkaw & Tongkaw (2016), Llanos (2006), Poess & Nambiar (2008), Cole, et al. (2011)). Displaying the output variables such as number of transactions in a 10-minute period is easily comparable to tpmC as the results need only be divided by 10 to be in tpmC units . This should easily translate to the database community.

Oracle (2013) and SAP (2014) have both disclosed official reports on the performance of Oracle and MS SQL Server. These reports both have 3 distinct outputs: Total System Cost, TPC-C Throughput, and the Price over Performance (Total System Cost/tpmC) ratio. Total System Cost includes all the equipment needed (server hardware, server storage, server software, client hardware, client software and other hardware) and the estimated cost for maintenance/support for 3 years to run the DBMS. Throughput is measured by taking a sample somewhere in the middle of a benchmark and taking the average Throughput across that time. The benchmark lasts tens of thousands of seconds and the sample is taken somewhere in the middle to avoid the "Ramp-Up" and "Ramp-Down" periods at the beginning and end of each test.

Due to the legal restrictions on benchmarking surrounding Oracle (2018) and Microsoft (2019c, p. 6) DBMSs, it is problematic finding research papers on the benchmarking of the two database systems. Most benchmarks of SQL Server and Oracle are formally published by TPC-C in full disclosure reports that have to be verified by TPC-C (Transaction Processing Performance Council, 2019).

5 METHODOLOGY

This report is structured as case study that is related to experimental research methodology. It will be quantitative in that it tests Strong and other DBMSs to retrieve data and display statistics that can be used to come to constructive conclusions. The case study design will be flexible as the parameters of the experiment change regularly due to the experimental nature of Strong as well as regular updates and features to the RDBMS. It will also be qualitative in nature since the DBTech community will be consulted. This strategy is effective as a combined quantitative and qualitative approach usually yields a better understanding of the observed phenomena (Runeson & Host, 2009).

The quantitative part of the research in conducted "in-silico" (Travassos & Barros, 2003, p. 118) , with data generated through the stimulation of StrongDBMS and other RDBMSs by our implementation of the TPC-C Benchmark. For the qualitative stage, the DBTech community will be involved. The DBTech is a small community of practice of experts in database technology. The DBTech community has been supporting the development of StrongDBMS, and they are the ideal population to validate and comment on the results of the research described in this dissertation.

5.1 GOALS AND HYPOTHESIS

The goal will be to compare how StrongDBMS performs against commercially available RDBMSs and gather expert opinion on the matter to evaluate its developmental progress.

It is hypothesized, that StrongDBMS will not to perform as well as other major DBMSs in terms of throughput as it is an experimental project and is likely in need of optimization to benefit the single user environment however, as it is likely to have less overhead it should not be much slower that the established DBMSs. In contrast, StrongDBMS is expected to perform well in the concurrency tests as StrongDBMS has an optimistic approach to the committing of transactions and no row-locking compared to the other DBMSs. The documentation for the other DBMSs all state that using serializable isolation level in high contention environments may result in a poor performance so StrongDBMS is expected to perform slightly better in this respect.

5.2 RESEARCH DESIGN

Two different tests will be defined to evaluate StrongDBMS. The first is a test with 1 clerk to evaluate how quickly each RDBMS can reach 2000 new orders with the output being the time taken to reach 2000 new orders. This is a simplified implementation of the TPC-C to gauge throughput for a single user. The second test involves a certain number of clerks over 10 minutes performing transactions and producing a measured output of New Orders. The number of clerks is incrementally increased each time the test is run until the DBMS is judged to have reached a maximum clerk-to-New Order ratio. This output of New Orders is easily comparable to the TPC-C recognised tpmC as it would translate to tpmC being multiplied by 10. Consistency checks are carried out according to TPC-C guidelines (Transaction Processing Performance Council, 2010, pp. 48 - 50) to evaluate how the DBMSs are being treated by the TPC-C implementation and to compare how similar it is to official TPC-C. The professional opinion of the DBTech community will be gathered and related to the interpretation of the work carried out and some discussion behind the results.

5.3 INSTRUMENTATION

MS Visual Studio 2019, Windows Command prompt, Windows Task Manager, StrongDBMS v0.1, PostgreSQL 11.3, PgAdmin4, and anonymous DBMS software.

5.4 Data Collection Procedure

The database was set to its initial state before the TPC-C implementation is initialised. Then the throughput or concurrency test was run while CPU and RAM utilization was monitored in case of either being exhausted. After each test was completed, the tested DBMS was queried, using its own SQL syntax, for the number of rows in each table of the TPC-C implementation database. The results of this query were then copied and stored. The spreadsheet was used to then display the data in graphs that are easy to decipher. When the throughput test was conducted, the time at the start of the test was recorded as well as the time on the final form. The difference was calculated in Excel to find the time taken for the test. This was then displayed in an Excel graph. Before iterations of each test was carried out, the database was reset to its initial state. The consistency checks were conducted at the end of a benchmark test by querying each DBMS using TPC-C consistency check specifications and each DBMSs SQL syntax. The output was gathered in a Word document. The results were compared to TPC-C regulations to see if each test passed or failed for consistency. This was then displayed in an Excel table comparing each DBMS, each consistency check, whether they passed or failed the consistency check, and the regulations for the consistency check.

5.5 Collecting Comments From Expert Community

The DBTech community and associates are a sample by convenience as summarised below in Table 2 influenced in methods described by Linaker & de Mello (2015, p. 16). The DBTech community has been apprised of the progress of StrongDBMS and has been active in suggesting potential features. As a result, they are a suitable target population for commenting on the results of this research. However, the DBTech is also a small community (about 8 members). Which means the results will be generalizable to potential users of StrongDBMS but will serve to feedback the results of this research into the development and evolution of StrongDBMS. A duration of one week was given to the experts as a response window to the questionnaire.

Target Audience	**Units of Analysis and Observation**	**Search Unit**
Experts acquainted with StrongDBMS and database development/technologies	Individual email responses	Writing emails directly to DBTech contact list

Table 2: Identifying the Target Frame and Sample Audience

The information presented to the DBTech community was a background into the investigation with the major findings as seen in 12.2 Information and Questionnaire Sheet on page 47. The background on the study that was given to them included the results Figure 1: 2000 New Order Test and Figure 2: 10-Minute New Order Concurrency Tests. Further information was made available to them upon request. Relevant links such as the source code for the TPC-C application and StrongDBMS files at https://github.com/MalcolmCrowe/ShareableDataStructures were made available should they want to conduct deeper analysis into the findings. The anonymity of the Commercial DBMS was revealed privately to them as well to help their analysis. The DBTech community responses that may conflict with the anonymity of the commercial DBMS will be masked so to not breach the legal terms for Oracle (Oracle, 2018) and MS SQL Server (Microsoft, 2019c, p. 6). These responses can be found in 12.3 Email Reponses From Expert Community beginning on page 54.

5.6 EXPERTS

DBTechNET is a community of European IT companies and tertiary schools that are collaborating to, among other objectives, design and develop tools that allow greater world access to data and database technology (DBTech NET, 2019). As UWS is a collaborative partner of this group, it is likely that there will be more positive responses to any questions posed to the group. There are a wide variety of partners from institutions that are likely to have expert experience on the issues that this experiment will try to tackle. The DBTech community have published a number of papers on the transaction handling qualities, concurrency technologies and other features of major DBMSs. The level and variety of experience that DBTech possess, make them the ideal candidates to approach regarding this project. After the information and questionnaire sheet were sent out, three replies were received within the acceptable time frame. The experts have skipped comment on certain questions as they might not be able to offer conclusive comments on the matter as it may be outside their field of expertise. The three out of eight experts that replied are both directly and indirectly involved with the DBTech community. They have experience as professors and as industry professionals in the fields of systems analysis, innovative technologies for RDBMSs, and database theory and practice.

6 Execution

6.1 Preparation

The server of each database must be initialised before tests can begin. The TPC-C database must be in its original state for tests to be fair and repeatable. On creation of the database using the implementation of TPC-C, a back-up is created for the TPC-C database and then saved as a different file. This back-up of the initial database state can then be restored after each test has completed and the necessary data has been collected. Task Manager is monitored to ensure CPU or RAM usage does not exceed maximum values as this could result in poor performance by the DBMSs and may detract the validity of the benchmark results. To minimise the risk of CPU/RAM saturation, all unnecessary programs on the test PC are closed.

6.2 Data Collection Performed

Each RBMS will be queried in each respective SQL syntax to find the initial state and end state of each TPC-C benchmark. For the 10-minute New Order Concurrency tests, the number of rows added to the New Order table will be collected for comparison. For the 2000 New Order test, the time for the test to run will be measured and collected for comparison. For the consistency checks, the TPC-C specifications (Transaction Processing Performance Council, 2010, pp. 48 - 50) will be consulted to implement SQL queries for each RDBMS to check the data consistency. All the results will be recorded to produce graphs and tables for effective comparison.

6.3 Validity Procedure

Multiple executions of the TPC-C implementation are run for each number of clerks for each DBMS to ensure that the tests are running consistently. Any tests that demonstrate inconsistent grouping will be considered untrustworthy and insufficiently valid.

The consistency checks are used to evaluate how validated the results would be in comparison to an official TPC-C benchmark.

As the DBTech community are involved to provide their insights on the results, they were likely point out any discrepancies that will help gauge the validity of the study.

6.4 Environment

The tests are run on a PC with a 64-bit Windows 10 operating system on a PNY CS900 SATA SSD, i5-4460 3.2 GHz processor with 4 cores, 8 GB of RAM, with a NVIDIA GeForce GTX 750 Ti graphics card. The RDBMS's servers, client applications and TPC-C data will be accessed from the E: drive on a Seagate ST1000DM003 Barracuda HDD.

7 Analysis

7.1 Descriptive Statistics

As seen in Figure 1, StrongDBMS has a significantly lower throughput than the Commercial DBMS and PostgreSQL averaging 172.3, 27.3 and 37.7 seconds respectively. The grouping of the test times for the Commercial DBMS and PostgreSQL are very tight compared to the grouping of StrongDBMS which is less consistent but not untrustworthy.

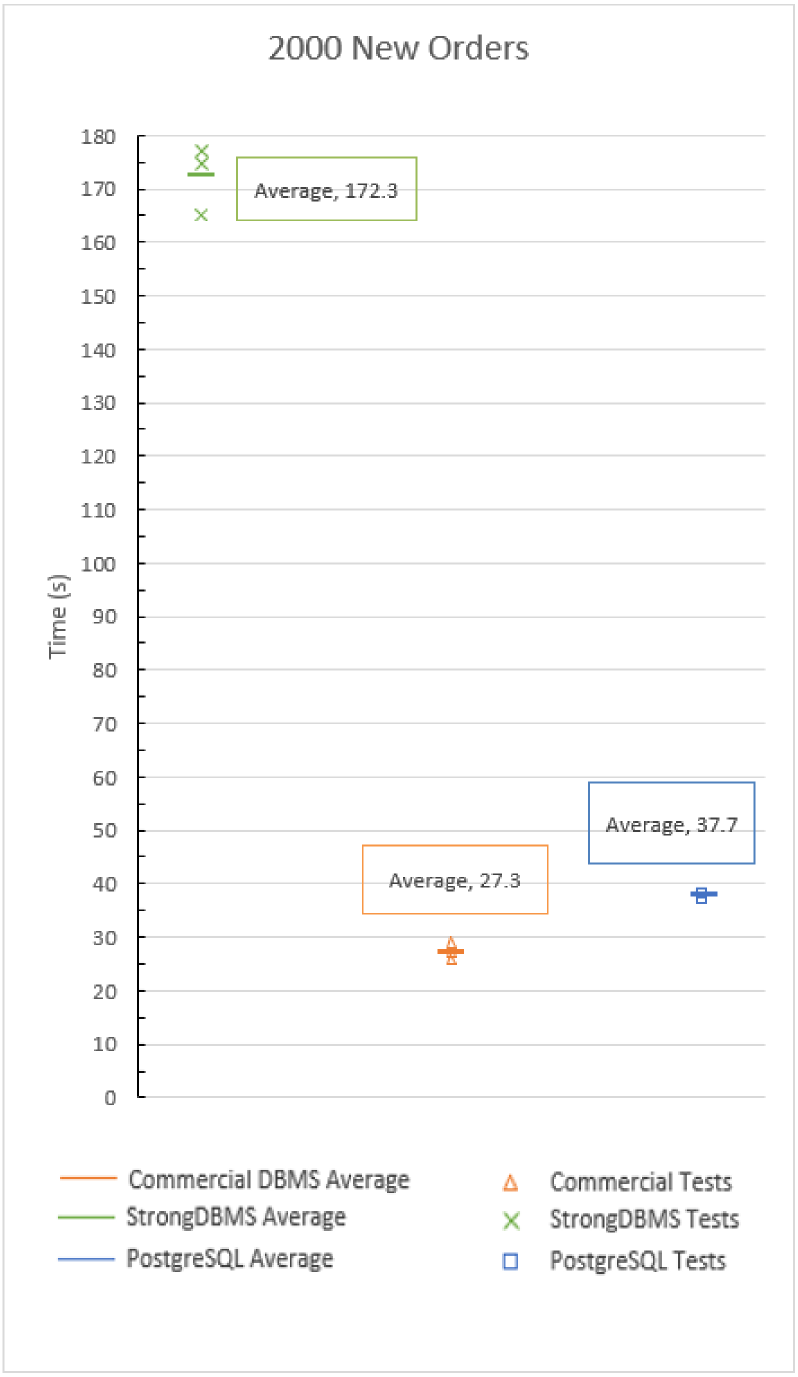


Figure 1: 2000 New Order Test

As seen in Figure 2, StrongDBMS can process substantially more New Orders on average compared to PostgreSQL and the Commercial DBMS. The results for each DBMs become less consistent as more clerks are added. StrongDBMS, Commercial DBMS and PostgreSQL have maximum average saturation points at 80, 10, and 2 clerks respectively; and maximum values at 110, 15, and 2 clerks respectively. The maximum values are not as trustworthy as the maximum average values as they are not guaranteed as shown by the grouping of results during each test. PostgreSQL and Strong demonstrate tight grouping and can be trusted up to 10 and 120 clerks respectively. Commercial DBMS is only trustworthy up until 10 clerks. With more clerks, Commercial DBMS becomes unpredictable and the results are untrustworthy.

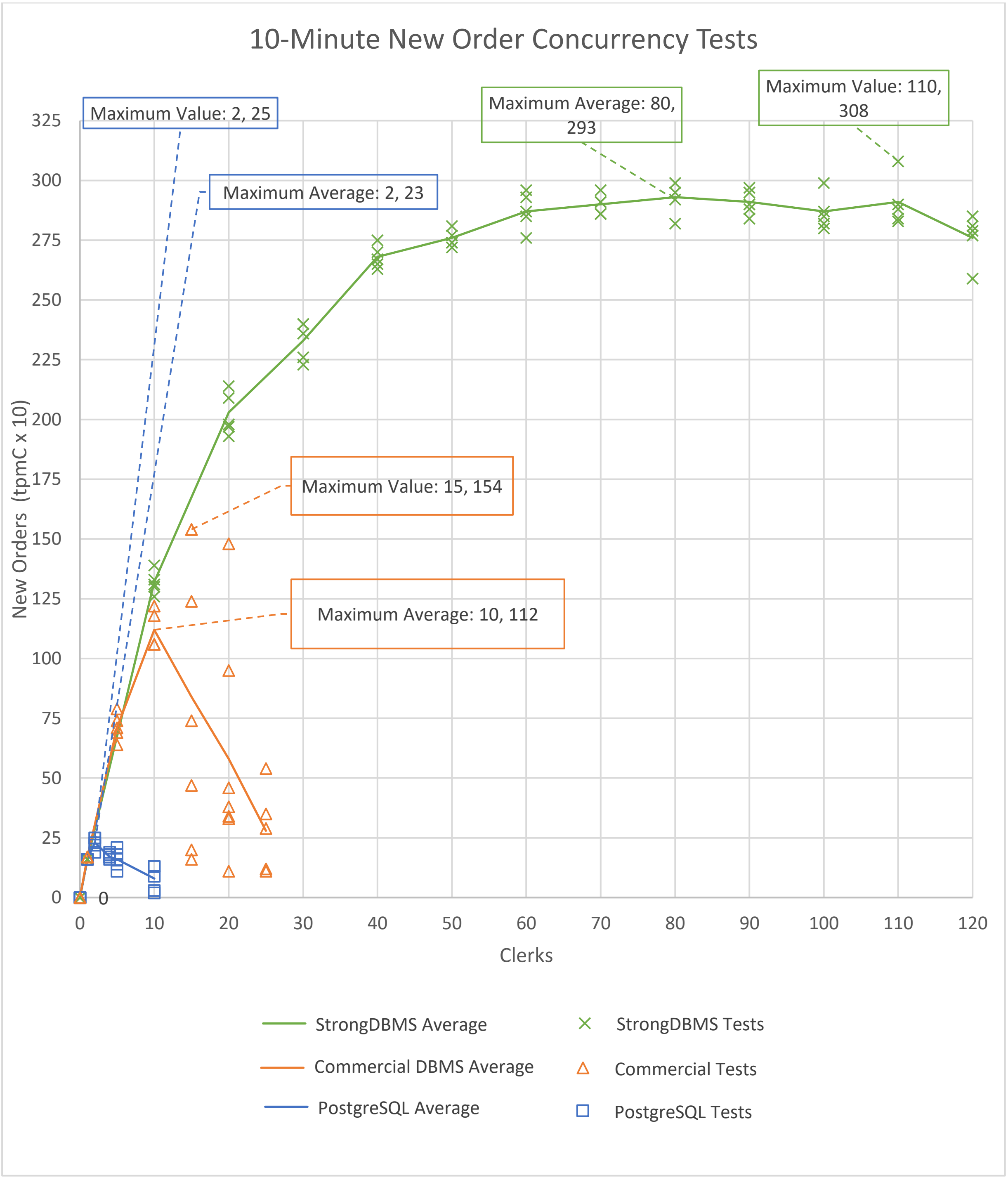


Figure 2: 10-Minute New Order Concurrency Tests

As seen in Table 3, Table 4, and Table 5; the efficiency of each clerk drops as the number of clerks increase. StrongDBMS, Commercial DBMS and PostgreSQL maintain clerk efficiency levels of above 50% up to 30, 15 and 4 clerks respectively. Table 6 shows a clerk efficiency comparison between the DBMSs taking the best Perfect New Orders estimation from Commercial DBMS. This shows that StrongDBMS's efficiency levels drop gradually when clerks are increased compared to the steep drop in efficiency shown by PostgreSQL and Commercial DBMS.

StrongDBMS													
Clerks	**1**	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**	**100**	**110**	**120**
Perfect New_orders	16	160	320	480	640	800	960	1120	1280	1440	1600	1760	1920
STRONG_AVE_NEW_ORDER	16	132	203	233	268	276	287	290	293	291	287	291	276
Estimated Exceptions	0	28	117	247	372	524	673	830	987	1149	1313	1469	1644
Efficiency	100%	83%	63%	49%	42%	35%	30%	26%	23%	20%	18%	17%	14%

Table 3: StrongDBMS Perfect Concurrency

Commercial						
Clerks	**1**	**5**	**10**	**15**	**20**	**25**
Perfect New_orders	17	85	170	255	340	425
COMMERCIAL_AVE_NEW_ORDER	17	71	112	84	58	28
Estimated Exceptions	0	14	58	171	282	397
Efficiency	100%	84%	66%	33%	17%	7%

Table 4: Commercial DBMS Perfect Concurrency

PostgreSQL					
Clerks	**1**	**2**	**4**	**5**	**10**
Perfect New_orders	16	32	64	80	160
POSTGRESQL_AVE_NEW_ORDER	16	23	17	16	8
Estimated Exceptions	0	9	47	64	152
Efficiency	100%	72%	27%	20%	5%

Table 5: PostgreSQL Perfect Concurrency

Comparison										
Clerks	**1**	**1**	**1**		**10**	**10**	**10**		**20**	**20**
Perfect New_orders	17	17	17		170	170	170		340	340
AVE_NEW_ORDER	16	17	16	…	132	112	8	…	203	58
Exceptions	1	0	1		38	58	162		137	282
Efficiency Comparable to **Commercial** Perfect	94%	100%	94%		78%	66%	5%		60%	17%

Table 6: Efficiency Comparison Based on Highest Perfect DBMS Concurrency

As seen in Figures 3 - 5, the estimated perfect New Orders is far greater than any DBMSs actual New Order concurrency performance. The data on these graphs is taken from the efficiency tables: Table 3, Table 4, and Table 5. They can be compared effectively in Figure 6 where you can see the PostgreSQL clerk efficiency drops most dramatically as concurrency increases, followed by Commercial DBMS. StrongDBMS maintains a significantly slower rate of deterioration in clerk efficiency despite initially being less efficient than Commercial DBMS. This data is drawn from the efficiency comparison table: Table 6.

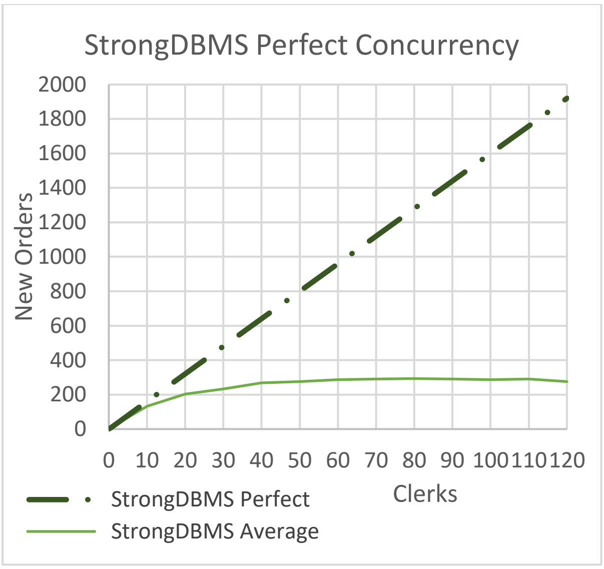


Figure 3: StrongDBMS Perfect vs. Actual Concurrency

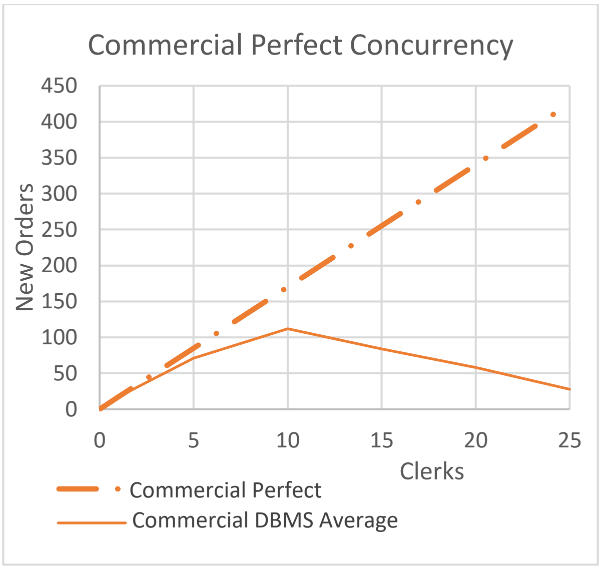


Figure 4: Commercial DBMS Perfect vs. Actual Concurrency

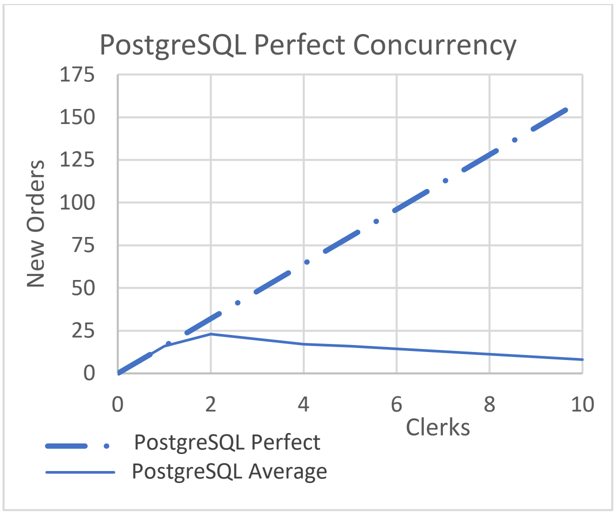


Figure 5: PostgreSQL Perfect vs. Actual Concurrency

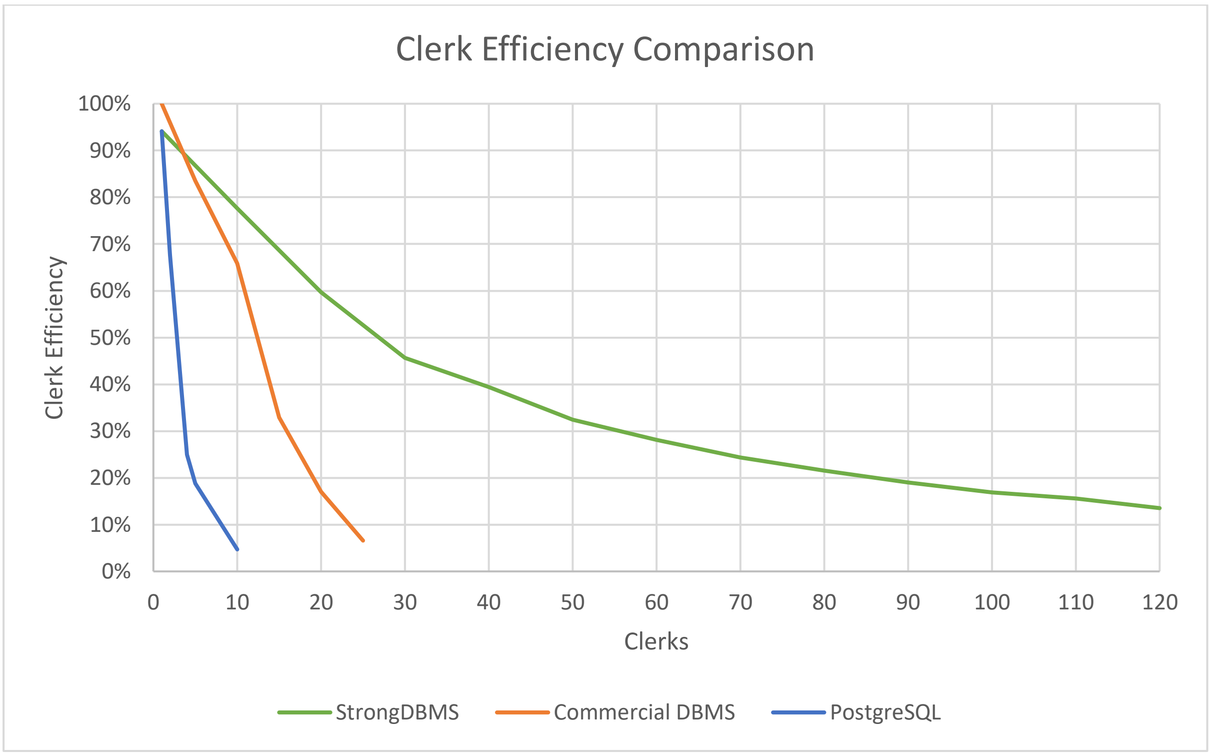


Figure 6: Clerk Efficiency Comparison Based on Highest Perfect DBMS Concurrency

As seen in Table 7 below, the Consistency checks show what queries fail to uphold TPC-C standards. All DBMSs pass and fail on the same checks with the exception being StrongDBMS for checks 7, 10 and 12. These checks returned non-verifiable results for StrongDBMS as the client Window's Command prompt displayed errors despite the correct SQL being used to query the database. All DBMSs pass and fail on the same tests with the exception being StrongDBMS which had some issues with some consistency checks related to using 'NOTNULL' as a constraint. These checks show how the TPC-C variant differs from the official TPC-C.

P=Passed, F=Failed, NV=Not Verified	StrongDBMS	Commercial	PostgreSQL	Consistency Check
1	F	F	F	**Consistency Condition 1**: Entries in the WAREHOUSE and DISTRICT tables must satisfy the relationship: W_YTD = sum(D_YTD)
2	P	P	P	**Consistency Condition 2:** Entries in the DISTRICT, ORDER, and NEW-ORDER tables must satisfy the relationship: D_NEXT_O_ID - 1 = max(O_ID) = max(NO_O_ID) for each district defined by (D_W_ID = O_W_ID = NO_W_ID) and (D_ID = O_D_ID = NO_D_ID). This condition does not apply to the NEW-ORDER table for any districts which have no outstanding new orders (i.e., the number of rows is zero).
3	P	P	P	**Consistency Condition 3**: Entries in the NEW-ORDER table must satisfy the relationship: max(NO_O_ID) - min(NO_O_ID) + 1 = [number of rows in the NEW-ORDER table for this district] for each district defined by NO_W_ID and NO_D_ID. This condition does not apply to any districts which have no outstanding new orders (i.e., the number of rows is zero).
4	F	F	F	**Consistency Condition 4**: Entries in the ORDER and ORDER-LINE tables must satisfy the relationship: sum(O_OL_CNT) = [number of rows in the ORDER-LINE table for this district] for each district defined by (O_W_ID = OL_W_ID) and (O_D_ID = OL_D_ID).
5	P	P	P	**Consistency Condition 5**: For any row in the ORDER table, O_CARRIER_ID is set to a null value if and only if there is a corresponding row in the NEW-ORDER table defined by (O_W_ID, O_D_ID, O_ID) = (NO_W_ID, NO_D_ID, NO_O_ID).
6	F	F	F	**Consistency Condition 6**: For any row in the ORDER table, O_OL_CNT must equal the number of rows in the ORDER-LINE table for the corresponding order defined by (O_W_ID, O_D_ID, O_ID) = (OL_W_ID, OL_D_ID, OL_O_ID).

P=Passed, F=Failed, NV=Not Verified	StrongDBMS	Commercial	PostgreSQL	Consistency Check
7	NV	P	P	**Consistency Condition 7**: For any row in the ORDER-LINE table, OL_DELIVERY_D is set to a null date/time if and only if the corresponding row in the ORDER table defined by (O_W_ID, O_D_ID, O_ID) = (OL_W_ID, OL_D_ID, OL_O_ID) has O_CARRIER_ID set to a null value.
8	F	F	F	**Consistency Condition 8**: Entries in the WAREHOUSE and HISTORY tables must satisfy the relationship: W_YTD = sum(H_AMOUNT) for each warehouse defined by (W_ID = H_W_ID).
9	F	F	F	**Consistency Condition 9**: Entries in the DISTRICT and HISTORY tables must satisfy the relationship: D_YTD = sum(H_AMOUNT) for each district defined by (D_W_ID, D_ID) = (H_W_ID, H_D_ID).
10	NV	F	F	**Consistency Condition 10**: Entries in the CUSTOMER, HISTORY, ORDER, and ORDER-LINE tables must satisfy the relationship: C_BALANCE = sum(OL_AMOUNT) - sum(H_AMOUNT) where: H_AMOUNT is selected by (C_W_ID, C_D_ID, C_ID) = (H_C_W_ID, H_C_D_ID, H_C_ID) and OL_AMOUNT is selected by: (OL_W_ID, OL_D_ID, OL_O_ID) = (O_W_ID, O_D_ID, O_ID) and (O_W_ID, O_D_ID, O_C_ID) = (C_W_ID, C_D_ID, C_ID) and (OL_DELIVERY_D is not a null value)
11	P	P	P	**Consistency Condition 11:** Entries in the CUSTOMER, ORDER and NEW-ORDER tables must satisfy the relationship: (count(*) from ORDER) - (count(*) from NEW-ORDER) = 2100 for each district defined by (O_W_ID, O_D_ID) = (NO_W_ID, NO_D_ID) = (C_W_ID, C_D_ID)
12	NV	F	F	**Consistency Condition 12**: Entries in the CUSTOMER and ORDER-LINE tables must satisfy the relationship: C_BALANCE + C_YTD_PAYMENT = sum(OL_AMOUNT) for any randomly selected customers and where OL_DELIVERY_D is not set to a null date/time.

Table 7: Consistency Check Results

7.2 Data Set Reduction

The 2000 New Order tests were run 3 times for each DBMS. The test times were collected to find an average and this was used for comparison in a single graph as seen in Figure 1

The 10-minute New Order Concurrency test was run a total of 5 times for every iteration of clerks for every DBMS. The 15 and 20-clerk tests for Commercial DBMS were run a total of 7 times in a search for better consistency because the initial spread of results was inconsistent. However, this was still not enough to make the results trustworthy. Each run was queried for the number of rows in each table in the TPC-C database. This query was copied and pasted into Word files and the number of rows in the New Order table was put into an Excel spreadsheet. This total number of New Orders was then subtracted from the initial database state, in Excel, to ascertain the number of New Orders that had been added in the 10-minute test. Each New Order count for each DBMS was averaged and all data points were added to Figure 2 to provide a clear comparison.

Data gathered from the consistency tests contained the queries as per TPC-C specifications modified to fit each DBMS SQL syntax. The results of the queries and the queries themselves were copied into a Word file for analysis. The results were then put together in Excel as a simple Pass or Fail of each check along with the conditions required.

7.3 Hypothesis Testing

StrongDBMS did not have as strong throughput performance in as expected, however, it was expected to perform better than it did. These DBMSs have larger development teams and have been in development for years so this is expected.

StrongDBMS outperformed both established DBMSs in terms of the amount of New Orders it could process in 10 minutes and the number of clerks that could concurrently work on the database. This was expected; however, it was not expected for the Commercial DBMS and PostgreSQL to perform so poorly as they are well-established DBMS.

The consistency checks demonstrate that this implementation of TPC-C was not used to fully test consistency as recommended by TPC-C standards (Transaction Processing Performance Council, 2010). It is likely that all DBMSs are tested equally in terms of data consistency as they pass the same consistency checks excluding the ones that returned “Not Verifiable” results.

8 Contrasting Results with Comments from the Expert Community

8.1 Evaluation of Results

This section contrasts the quantitative results gathered in the benchmark tests with the qualitative comments from the experts.

8.1.1 2000 New Order Tests

As seen in Figure 1, Strong takes nearly five to six times as long to process 2000 New Orders as PostgreSQL and the Commercial DBMS. This shows that StrongDBMS may not be as optimised as the established DBMSs considering they have built upon years of development. It could also be due to the structure of StrongDBMS differs to the structure of the other DBMSs. StrongDBMS is different from the Commercial DBMS and PostgreSQL in that it uses immutable data structures and an append-only transaction log.

8.1.1.1 Comments from Expert Community on 2000 New Order Tests

Expert A discusses that users regard the speed of a DBMS as a fundamental characteristic. StrongDBMS's poor performance in this test is disappointing and could be improved by compensating between concurrency and speed by tuning to find a desirable balance.

Expert B provides valuable insight into the transaction isolation level of PostgreSQL as a highly experienced user of the DBMS. They explain that the extremely poor performance of PostgreSQL in the 10-minute New Order concurrency test is enough to suspect a bug in the TPC-C implementation. This expert is very experienced with PostgreSQL and finds the poor concurrency performance very surprising so, due to this, the 2000 New Order throughput test cannot be accepted as trustworthy. The testing application will need further tuning and verification before conclusive results can be given.

Expert C theorises that StrongDBMS may be slower if it needs to issue more disc space for the growing database file due to the append-only nature of StrongDBMS.

8.1.2 10-Minute New Order Concurrency Tests

During the concurrency tests, Strong outperforms PostgreSQL and the Commercial DBMS by a significant margin saturating at about 80, 10, and 2 clerks respectively. This clearly shows that StrongDBMS potentially has far superior concurrency handling abilities. PostgreSQL and the Commercial DBMS both did not produce any read-conflicts due to the presence of MVCC/Row-Versioning however, this does not help them achieve anywhere near the same concurrency as StrongDBMS. The results that StrongDBMS produced always contained several read conflicts as well as write conflicts but this did not result in a poor concurrency performance. This would suggest that true OCC can produce favourable performance in high concurrency environments without the need to remove read-conflicts. Despite the advantage of having guaranteed no read-conflicts, OCC and shareable data structures deliver much better concurrency performance in this particular test as shown by StrongDBMS.

PostgreSQL was found to aggressively deal with conflicts, in an earlier implementation of TPC-C, by aborting the transaction rather than rollbacks then trying to commit later. This initially led to results that would not reach above 19 New Orders. However, the TPC-C application was modified to force PostgreSQL to rollback results rather than abort altogether.

8.1.2.1 Comments from Expert Community Comments on 10-Minute New Order Concurrency Tests

Expert A argues that although the results in this test favour StrongDBMS, the OLTP nature of TPC-C is not the typical environment for a modern DBMS as most are used with websites. He demonstrates that, web applications don't usually experience high levels of concurrency with many users as they usually deal with a limited number of threads from web servers. More so, the results from this TPC-C implementation will be used to tune StrongDBMS.

Expert B strongly suspects that the results for PostgreSQL are in error so the findings cannot be conclusive until the performance of PostgreSQL in the TPC-C implementation is validated.

Expert C speculates that StrongDBMS has an advantage over the locks employed by the Commercial DBMS and PostgreSQL if the TPC-C application tries to retry transactions that have been rolled back. This could lead to a queue of transactions forming in the other DBMSs. They also observe that the Commercial DBMS and PostgreSQL may be tuned to perform favourably in terms of throughput when applied in an environment without conflicts.

8.1.3 Consistency Checks

The consistency checks uncovered a bug within StrongDBMS when using 'NOT' and NOTNULL'. This does not help determine the consistency across the three tested databases, but it does provide feedback for the development of StrongDBMS. If the three test results that trigger the bug in StrongDBMS are ignored, it can be said that the TPC-C implementation differs from official TPC-C but appears to at least test each DBMS consistently as they fail and pass on the same consistency checks. The failed tests seem to compromise of any _YTD payments, the O_OL_CNT and count of ORDERLINE rows, and anything in the HISTORY table. These could be simple additions to the TPC-C implementation code but may result in reduced performance in the tested DBMSs so it may be worth retesting with an updated TPC-C implementation. There is also no guarantee that the new TPC-C implementation will produce similar results scaled down due to increased work done by the application as increased tasks may increase concurrency conflicts that effect the DBMSs performance to a polynomial scale rather than a linear relationship. The TPC-C implementation will need further modification to fully test scalability, consistency and durability in order to comply to official TPC-C specifications.

9 INTERPRETATIONS

Strong performs very well despite being in the early stages of development and results imply that it has impressive concurrency handling capabilities. However, the throughput of StrongDBMS is disappointing and will likely need improvement in later versions. The errors that occurred in the consistency checks prove that it is still very much in infancy and it will take some more development and testing before it becomes a truly competitive product. Failed consistency checks also imply that the TPC-C implementation needs to update the HISTORY table and any row in multiple tables that have to do with payments as these seem to be the common inconsistencies in the TPC-C application. Although these rules were not needed to necessarily test the concurrency it could be regarded as more trustworthy if a stricter TPC-C implementation is applied.

Comments from the consulted experts imply, and concur with the evaluation, that refining of the current testing methods and further tests to evaluate StrongDBMS in different scenarios are needed to provide conclusive results. They expect that, if an actual TPC-C were to be carried out, only 4% of the transactions would conflict. This would result in more favourable results for the Commercial DBMS and would be scalable to a limit of about "250" warehouses. TPC-C does not actually specify that serializable isolation must be used, and most DBMS guidelines advise against it unless it is necessary.

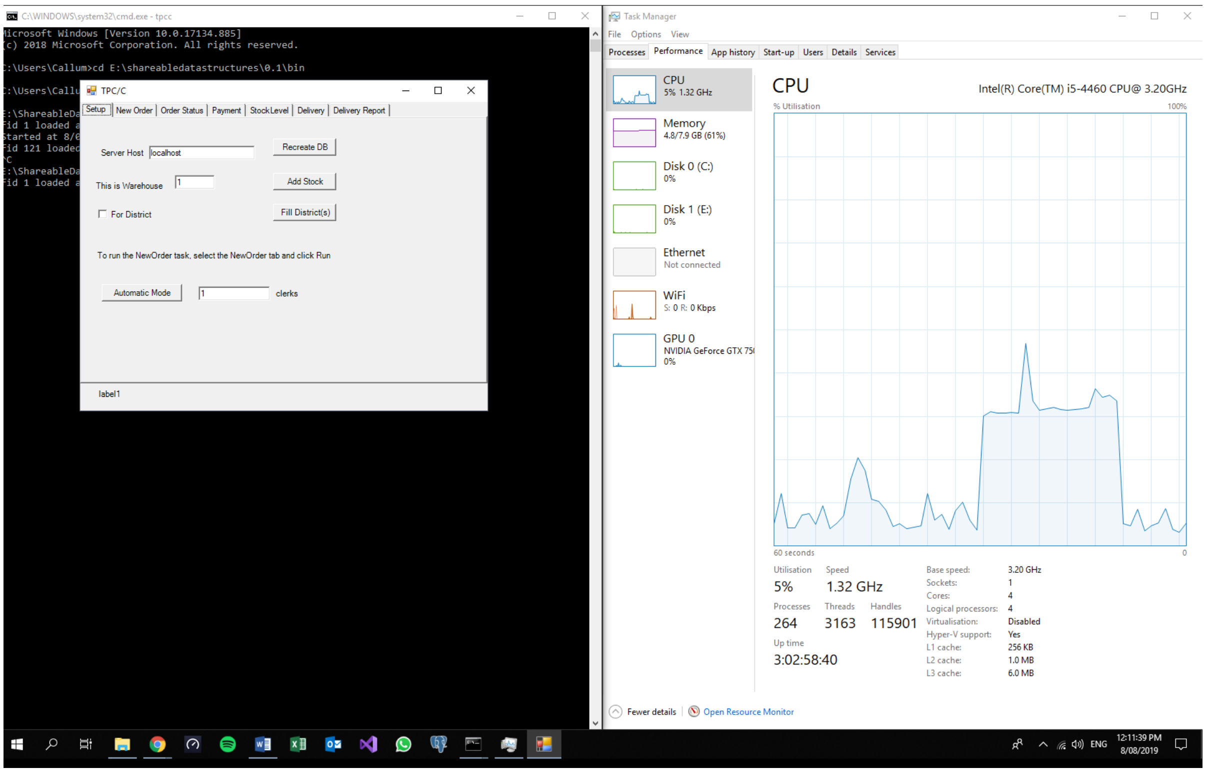
Due to the experimental nature of both the TPC-C implementation and StrongDBMS, the suspiciously poor concurrency performance of PostgreSQL, the inconsistency of Commercial DBMS at higher concurrency, and the bug uncovered in StrongDBMS during the consistency checks; there cannot be any finite conclusions given in this report.

Expert A discusses that the TPC-C implementation was limited to its current format to test each RDBMS in a high conflict environment, so it lacks the features and protocols that test other aspects of the DBMSs. This implementation does not test features that are found in most RDBMSs and are planned to be included in future versions of StrongDBMS such as "views, triggers, stored procedures and structured types".

Expert B states that the Commercial DBMS does not have a lot in common with the isolation levels of StrongDBMS and PostgreSQL. Given that it has to remain anonymous for legal reasons, it is hard to justify the use of it as a comparison in this particular TPC-C implementation for a public report.

Expert B's comments on the Commercial DBMS and its isolation level functionalities are insightful but must remain confidential as they would breach the user agreements of the Commercial DBMS and can still be used privately in consideration for future tests.

It may be worth noting that, while performing the consistency checks, Strong took much longer to respond to queries with multiples joins compared to the commercial DBMS and PostgreSQL. The response time to these queries was not measured however, this could be an avenue for future evaluation of the database and its capabilities. It may outperform the established DBMSs in concurrency however, it performs slowly to complex queries and may not be suited to applications that only have a small number of users that often compute queries with multiple joins. This could be due to StrongDBMS heavy reliance on CPU utilisation compared to the other DBMS servers as seen below in Figures 7 – 12. This could be negated with the use of the NVMe SSD technology.



C:\WINDOWS\system32\cmd.exe - tpcc

```
icrosoft Windows [Version 10.0.17134.885]
(c) 2018 Microsoft Corporation. All rights reserved.

:\Users\Callum>cd E:\shareabledatastructures\0.1\bin

:\Users\Callu
:\ShareableDa
fid 1 loaded a
Started at 8/0
fid 121 loaded
^C
:\ShareableDa
fid 1 loaded a
```

TPC/C

Setup | New Order | Order Status | Payment | StockLevel | Delivery | Delivery Report

Server Host: localhost

This is Warehouse: 1

For District

To run the NewOrder task, select the NewOrder tab and click Run

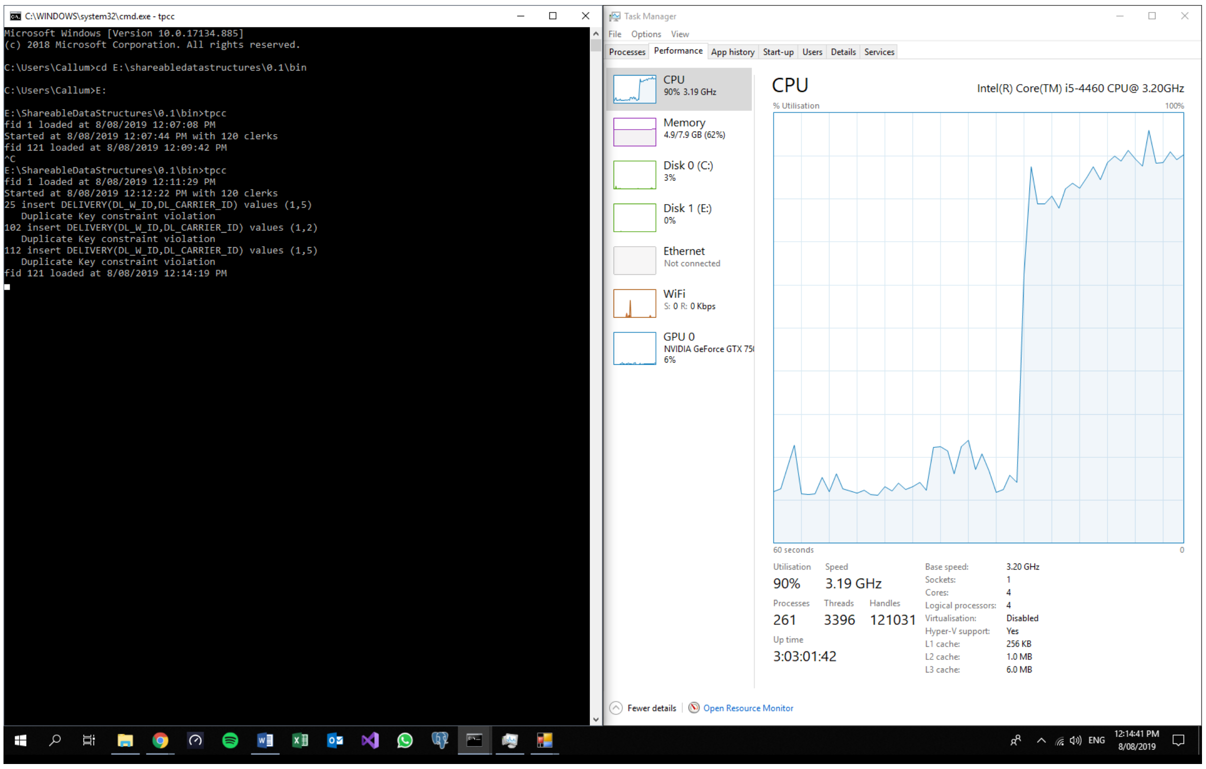
Automatic Mode: 1 clerks

label1

Task Manager — Performance — CPU: Intel(R) Core(TM) i5-4460 CPU@ 3.20GHz

Utilisation	5%
Speed	1.32 GHz
Processes	264
Threads	3163
Handles	115901
Up time	3:02:58:40
Base speed:	3.20 GHz
Sockets:	1
Cores:	4
Logical processors:	4
Virtualisation:	Disabled
Hyper-V support:	Yes
L1 cache:	256 KB
L2 cache:	1.0 MB
L3 cache:	6.0 MB

Figure 7: CPU before StrongDBMS 120 clerk TPCC Test



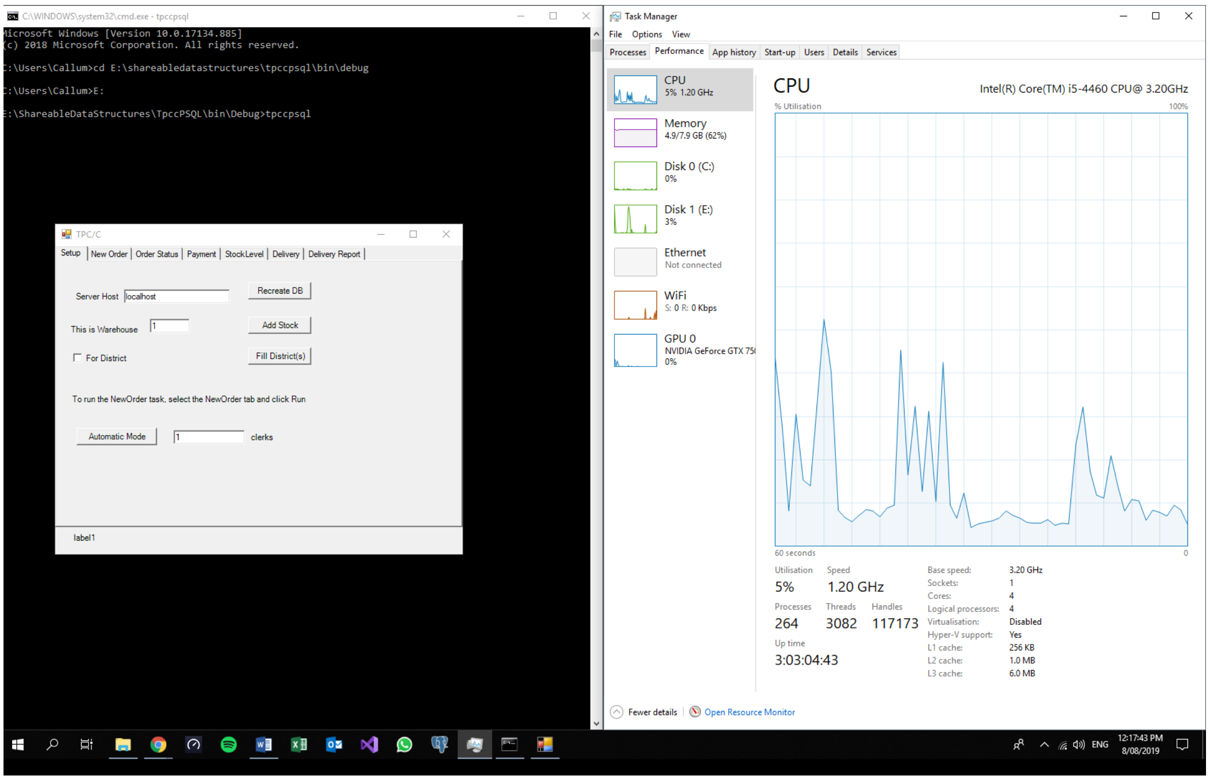
C:\WINDOWS\system32\cmd.exe - tpcc

```
Microsoft Windows [Version 10.0.17134.885]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Callum>cd E:\shareabledatastructures\0.1\bin

C:\Users\Callum>E:

E:\ShareableDataStructures\0.1\bin>tpcc
fid 1 loaded at 8/08/2019 12:07:08 PM
Started at 8/08/2019 12:07:44 PM with 120 clerks
fid 121 loaded at 8/08/2019 12:09:42 PM
^C
E:\ShareableDataStructures\0.1\bin>tpcc
fid 1 loaded at 8/08/2019 12:11:29 PM
Started at 8/08/2019 12:12:22 PM with 120 clerks
25 insert DELIVERY(DL_W_ID,DL_CARRIER_ID) values (1,5)
   Duplicate Key constraint violation
102 insert DELIVERY(DL_W_ID,DL_CARRIER_ID) values (1,2)
   Duplicate Key constraint violation
112 insert DELIVERY(DL_W_ID,DL_CARRIER_ID) values (1,5)
   Duplicate Key constraint violation
fid 121 loaded at 8/08/2019 12:14:19 PM
```

Task Manager — Performance — CPU: Intel(R) Core(TM) i5-4460 CPU@ 3.20GHz

Utilisation	90%
Speed	3.19 GHz
Processes	261
Threads	3396
Handles	121031
Up time	3:03:01:42
Base speed:	3.20 GHz
Sockets:	1
Cores:	4
Logical processors:	4
Virtualisation:	Disabled
Hyper-V support:	Yes
L1 cache:	256 KB
L2 cache:	1.0 MB
L3 cache:	6.0 MB

Figure 8: CPU during StrongDBMS 120 clerk TPC-C



```
C:\WINDOWS\system32\cmd.exe - tpccpsql
Microsoft Windows [Version 10.0.17134.885]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Callum>cd E:\shareabledatastructures\tpccpsql\bin\debug

C:\Users\Callum>E:

E:\ShareableDataStructures\TpccPSQL\bin\Debug>tpccpsql
```

TPC/C — Setup | New Order | Order Status | Payment | Stock Level | Delivery | Delivery Report

Server Host: localhost — Recreate DB
This is Warehouse: 1 — Add Stock
☐ For District — Fill District(s)
To run the NewOrder task, select the NewOrder tab and click Run
Automatic Mode — 1 clerks
label1

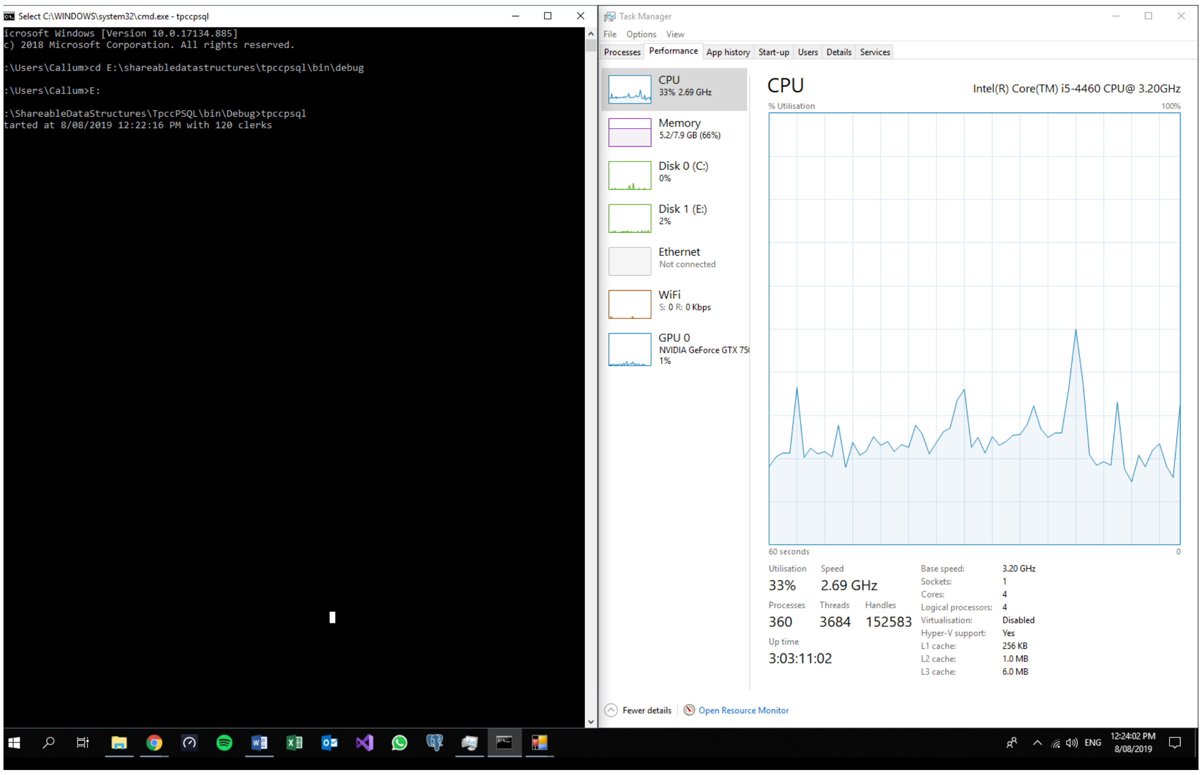
Task Manager — CPU: Intel(R) Core(TM) i5-4460 CPU@ 3.20GHz

Utilisation	5%
Speed	1.20 GHz
Processes	264
Threads	3082
Handles	117173
Up time	3:03:04:43
Base speed:	3.20 GHz
Sockets:	1
Cores:	4
Logical processors:	4
Virtualisation:	Disabled
Hyper-V support:	Yes
L1 cache:	256 KB
L2 cache:	1.0 MB
L3 cache:	6.0 MB

Memory 4.9/7.9 GB (62%); Disk 0 (C:) 0%; Disk 1 (E:) 3%; Ethernet Not connected; WiFi S: 0 R: 0 Kbps; GPU 0 NVIDIA GeForce GTX 75 0%

12:17:43 PM 8/08/2019

Figure 9: CPU before PostgreSQL 120 clerk TPC-C test



```
Select C:\WINDOWS\system32\cmd.exe - tpccpsql
icrosoft Windows [Version 10.0.17134.885]
c) 2018 Microsoft Corporation. All rights reserved.

:\Users\Callum>cd E:\shareabledatastructures\tpccpsql\bin\debug

:\Users\Callum>E:

:\ShareableDataStructures\TpccPSQL\bin\Debug>tpccpsql
tarted at 8/08/2019 12:22:16 PM with 120 clerks
```

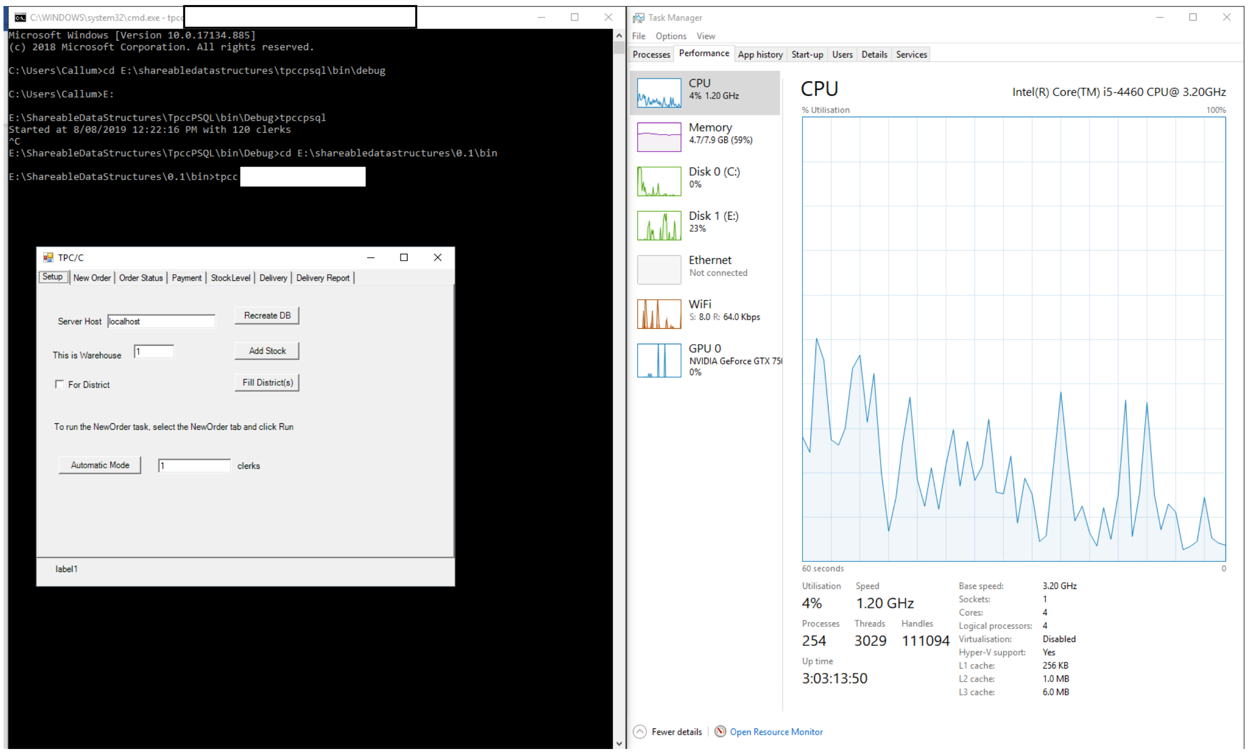
Task Manager — CPU: Intel(R) Core(TM) i5-4460 CPU@ 3.20GHz

Utilisation	33%
Speed	2.69 GHz
Processes	360
Threads	3684
Handles	152583
Up time	3:03:11:02
Base speed:	3.20 GHz
Sockets:	1
Cores:	4
Logical processors:	4
Virtualisation:	Disabled
Hyper-V support:	Yes
L1 cache:	256 KB
L2 cache:	1.0 MB
L3 cache:	6.0 MB

Memory 5.2/7.9 GB (66%); Disk 0 (C:) 0%; Disk 1 (E:) 2%; Ethernet Not connected; WiFi S: 0 R: 0 Kbps; GPU 0 NVIDIA GeForce GTX 75 1%

12:24:02 PM 8/08/2019

Figure 10: CPU during PostgreSQL 120 clerk TPC-C test



C:\WINDOWS\system32\cmd.exe - tpcc[redacted]

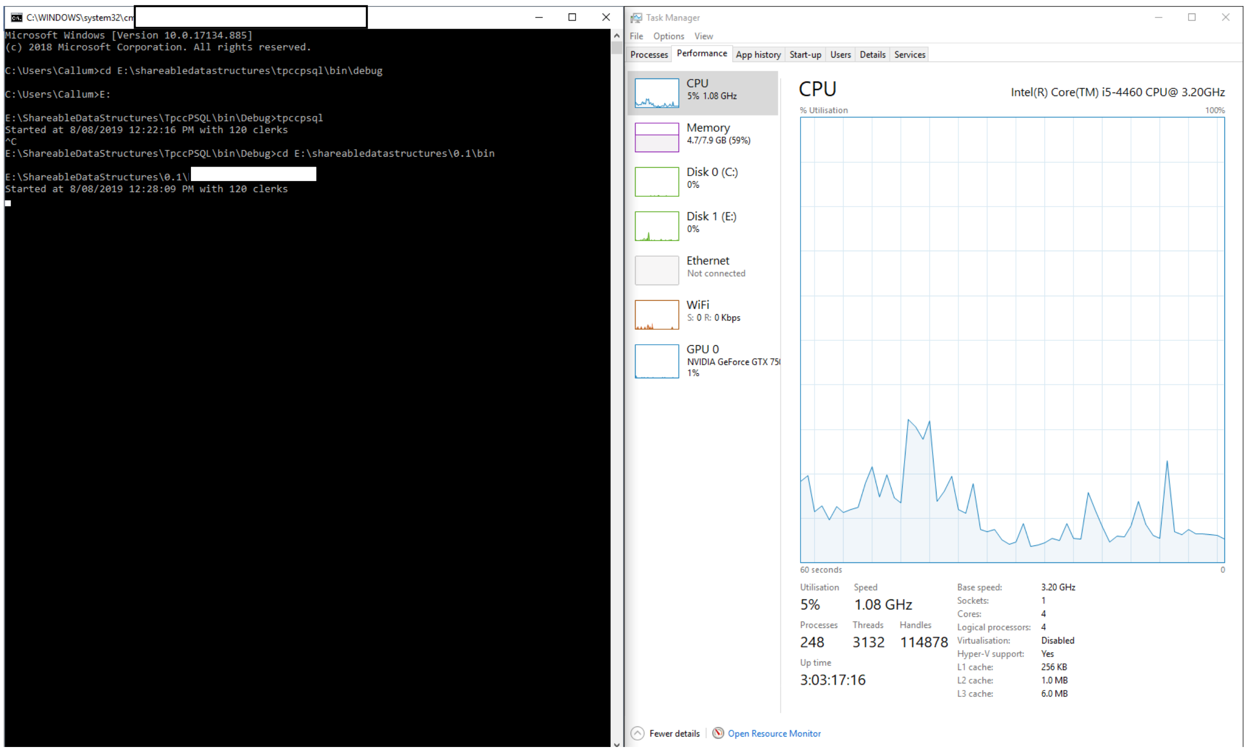
```
Microsoft Windows [Version 10.0.17134.885]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Callum>cd E:\shareabledatastructures\tpccpsql\bin\debug

C:\Users\Callum>E:

E:\ShareableDataStructures\TpccPSQL\bin\Debug>tpccpsql
Started at 8/08/2019 12:22:16 PM with 120 clerks
^C
E:\ShareableDataStructures\TpccPSQL\bin\Debug>cd E:\shareabledatastructures\0.1\bin

E:\ShareableDataStructures\0.1\bin>tpcc [redacted]
```

TPC/C
Setup | New Order | Order Status | Payment | StockLevel | Delivery | Delivery Report
Server Host localhost
Recreate DB
This is Warehouse 1
Add Stock
For District
Fill District(s)
To run the NewOrder task, select the NewOrder tab and click Run
Automatic Mode
1 clerks
label1

Task Manager

Utilisation 4% Speed 1.20 GHz
Processes 254 Threads 3029 Handles 111094
Up time 3:03:13:50
Base speed: 3.20 GHz
Sockets: 1
Cores: 4
Logical processors: 4
Virtualisation: Disabled
Hyper-V support: Yes
L1 cache: 256 KB
L2 cache: 1.0 MB
L3 cache: 6.0 MB

Figure 11: CPU before Commercial DBMS 120 clerk TPC-C test



C:\WINDOWS\system32\cm[redacted]

```
Microsoft Windows [Version 10.0.17134.885]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Callum>cd E:\shareabledatastructures\tpccpsql\bin\debug

C:\Users\Callum>E:

E:\ShareableDataStructures\TpccPSQL\bin\Debug>tpccpsql
Started at 8/08/2019 12:22:16 PM with 120 clerks
^C
E:\ShareableDataStructures\TpccPSQL\bin\Debug>cd E:\shareabledatastructures\0.1\bin

E:\ShareableDataStructures\0.1\[redacted]
Started at 8/08/2019 12:28:09 PM with 120 clerks
```

Task Manager

Utilisation 5% Speed 1.08 GHz
Processes 248 Threads 3132 Handles 114878
Up time 3:03:17:16
Base speed: 3.20 GHz
Sockets: 1
Cores: 4
Logical processors: 4
Virtualisation: Disabled
Hyper-V support: Yes
L1 cache: 256 KB
L2 cache: 1.0 MB
L3 cache: 6.0 MB

Figure 12: CPU during Commercial DBMS 120 clerk TPC-C test

9.1 Future Development of StrongDBMS

As StrongDBMS is experimental software, it needs reviewing and further development before an Alpha version can be sent to end users for a more traditional survey style research. The users would have to be competent with RDBMS technology. However, getting a sufficient sample size could prove problematic as these users are likely to be students or professionals that may not want to contribute a significant amount of their time as this would likely be a time intensive exercise. The users could be given exercises and data sets to perform queries on. Or in a more open-ended approach, users could implement a database design of their own choosing and encouraged to be creative. User comments could be mined to improve features, refine the user manual, make the software more user-friendly, and remove bugs that are found during their experience with the software.

Scalability, consistency and durability are not fully tested in this TPC-C benchmark according to official TPC-C documentation (Transaction Processing Performance Council, 2010). This does not necessarily mean that the DBMSs are not capable of performing well and passing an official TPC-C benchmark test. As the implementation of TPC-C was created in a way to mainly test concurrency, the code can be adapted later to test these parameters. Lower isolation settings of the established DBMSs could also be tested to assess how much isolation and consistency need to be sacrificed to achieve a competitive concurrency performance against StrongDBMS. The TPC-C application could also be optimised for each DBMS however this could make the test slightly unfair as optimisation might be more effective for some software more than others. The Commercial DBMS could certainly be optimised to deliver more consistent results as it was the only DBMS to deliver inconsistent results after its saturation point whereas the StrongDBMS and PostgreSQL results were consistently grouped.

A TPC-C application that abides more strictly to the specifications could be used to test StrongDBMS and other DBMSs for consistency, isolation and durability with greater validity. It would be hard to implement a full TPC-C test as these require auditors and a lot of hardware to cope with the scaling of the database while it is running for a long period of time. As StrongDBMS is in development at UWS it would also be hard to fund a full TPC-C test unless there was corporate sponsorship involved which is extremely unlikely as these test can cost in hundreds of thousands in terms of the hardware needed as seen in Oracle (2013) and SAP (2014).

This project did not abide by the scalable parameters set in TPC-C (Transaction Processing Performance Council, 2010) and had limited scalability in that only 1 warehouse was used, more than 10 clerks were used per warehouse, and the tpmC exceeded the maximum stated value for StrongDBMS and Commercial DBMS. So, it would be interesting to see if each DBMS can demonstrate scalability with another benchmark, based on TPC-C specifications, that includes these parameters. Scalability can also be tested by partitioning the DBMS over several servers rather than testing on a single server as was done in this case. This does risk atomicity being compromised if one server fails. However, this can be overcome by implementing a commit protocol which checks that all servers involved in the transaction have not failed. This can be a very resource heavy process which may result in a drop in overall performance (Abadi & Faleiro, 2018). It is also difficult to enforce serializable behaviour over several partitioned servers as it is very hard to guarantee that any particular query will only require one access to the database on one partition. At some point, a query is likely going to need resources from several servers, and this can incur a severe sacrifice in performance while all other contending transactions wait until the query using multiple servers has finished (Harding, et al., 2017). Harding, et al. (2017) also found that two-phase locking and timestamped transaction protocols perform very poorly in this setup. OCC was found to maintain a certain level of performance across a variety of different loads but still suffered from the scheduling of transactions. It is unknown how

StrongDBMS would cope in such an environment at this early stage of development, but it could be an interesting avenue of research to pursue.

Previously mentioned academic attacks such as ACIDRain (Warszawski & Bailis, 2017) could be tried on StrongDBMS. This kind of attack is easily stopped by a DBMS that employs serializable behaviour such as StrongDBMS. Based on the results of this benchmark, StrongDBMS has the potential to perform very well against this kind of attack.

Further features could be added to Strong like a server management application similar to pgAdmin4 for PostgreSQL. Currently a StrongDBMS client user would open a command-line window but it would be interesting to see what a software development student may be able to produce in terms of a user-friendly client application for StrongDBMS. This is not a pressing matter as PostgreSQL also uses the Window's Command prompt as an effective client application to the DBMS server. This may be a useful project to pursue for a more display that is more friendly to the casual user after the development is finished as features are still being added to StrongDBMS. The Window's Command prompt is more than sufficient for the time being.

Since StrongDBMS's strength lay in concurrency, it would make sense to try and improve on the throughput ability of the DBMS as it performed poorly compared to Commercial DBMS and PostgreSQL. StrongDBMS is based on Pyrrho, which has the ability to write to non-volatile storage up to 70 times faster than other DBMSs as all the data in a Pyrrho transaction is written in one process (Begg, et al., 2015). This is promising as NVMe SSD technology is advancing and has the potential to revolutionise how quickly data can be retrieved and stored (Stratikopoulos, et al., 2018). However, it is worth noting that there are other technologies also in development such as Dynamic Random-Access Memory (DRAM, this is a volatile technology) and Non-Volatile Random-Access Memory (NVRAM) that can produce better latency and throughput results than NVMe SSDs (Lüttgau, et al., 2018). These developments may improve throughput significantly for single users connecting to StrongDBMS. This also may be enough to close the gap on performance in the 2000 New Order test with PostgreSQL and the Commercial DBMS. Due to this, it may be worth repeating the benchmark tests using both StrongDBMS and Pyrrho on a similar PC environment that utilizes these new hardware storage technologies.

9.2 Comments from Expert Community on Future Development of StrongDBMS

Expert A explains that StrongDBMS is evidence that shareable data structures provide a feasible foundation for a RDBMS. They also indicate that software such as StrongDBMS, which employ shareable data structures, are a viable alternative to traditional data structures such as lists and arrays. There is also evidence that these type of DBMS structures scale well. It is not obvious that current RDBMS technology should be redesigned to implement shareable data structures but may gain an advantage by using new strategies such as RESTView (Crowe & Laux, 2018). Expert A would like to see StrongDBMS tested in a web-development type environment in the future as it could be shown to perform well.

Other work that is taking place parallel to the development of StrongDBMS, is the redevelopment of Pyrrho, to also embrace shareable data structures and improve performance, which have since declined, from earlier benchmarks. Expert A maintains that keeping serializable as the only isolation level, for StrongDBMS and Pyrrho, is sensible as lower isolation levels are used for usually implemented in application development.

Expert B finds the thought of immutable data structures used in a DBMS to be intriguing and it requires further and varied research to conclusively state its benefits. However, he iterates the importance of validating that the performance, particularly of PostgreSQL, and the other DBMSs are being equally and fairly tested before coming to any conclusion about the performance of StrongDBMS due to its immutable data structures.

Expert C recommends tuning StrongDBMS to improve throughput performance after investigating what makes it so much slower than the other DBMSs and investigating why the Commercial DBMS is so inconsistent after its on currency saturation point, They would also like to see TPC-E implemented to test StrongDBMS as it is also an OLTP benchmark, but it has a more complicated schema than TPC-C. Expert C suggests a future test scenario, similar to the suggested auction environment, that involves "inventory management, seat reservation in an aircraft, trading, etc." The last two of which are implemented in OLTP-Bench (Difallah, et al., 2013) with SEATS and AuctionMark.

10 Conclusions and Future Work

10.1 Conclusion

This project sought out to compare StrongDBMS to commercial databases. A case study research was defined and implemented to provide feedback for the development of StrongDBMS. Experts associated with DBTech were contacted and responded with very helpful and insightful feedback. Despite not being conclusive, it can be said that the results gathered from this report show the encouraging potential that StrongDBMS possesses and these results will help aid in the development of the innovative software and the tuning of the TPC-C implementation for future tests.

10.2 Relation to Existing Evidence

PostgreSQL had a poor concurrency performance at a serializable isolation level which could have been due to its use of MVCC as mentioned before by Abadi & Faleiro (2018) and (PostgreSQL Global Development Group (2019a). PostgreSQL was found to have very poor concurrency in initial tests where the TPC-C application treated it similarly to StrongDBMS and Commercial DBMS.

However, its poor performance could also be because the TPC-C implementation is more suited to StrongDBMS and the Commercial DBMS which is a potential source for bias. Fiannaca & Huang (2015) explain that PostgreSQL needs more fine tuning when using it in relation to custom software applications. They describe improving throughput performance by altering the application that was used to test PostgreSQL. PostgreSQL was initially receiving transactions in auto commit mode. Performance was improved by preventing any auto commits and grouping messages together for PostgreSQL to reduce overhead. However, this only seemed to slightly improve performance so it is unlikely to expect significant concurrency improvement from PostgreSQL should future benchmark tests be performed on it with a modified TPC-C application. (Oracle, 2019a) also warn application developers that applications that operate in serialization mode will require extra work by the developer to reverse the transaction and try it again. Longer running transactions are more likely to encounter concurrency issues and need rolled back. This will waste the initial work and need to be input again.

StrongDBMS truly enforces OCC as its predecessor Pyrrho did (Laiho & Laux, 2011). It does not use locking or conventional Multi-Versioning, and this means it does not have to commit resources to deal with these issues that the other DBMSs have to deal with. Issues such as lock-timeouts and deadlocks cannot occur in StrongDBMS because it does not have a need to apply them to achieve serializable transaction isolation.

Only so much can be said for the Commercial DBMS without revealing its identity which would infringe on user agreements in respect to benchmarking. Commercial DBMS performed well concurrently until more than 10 clerks were connected to the server.

The database community has also discussed the difficulty of enforcing serializable behaviour in web applications as using pessimistic concurrency approaches such as locking, with HTTP is problematic. Timestamping transactions, which is optimistic in some DBMSs eyes, is more practical (Cave, 2018).

10.3 Impact

Based on StrongDBMS's concurrency it shows a lot of potential to be implemented in industrial settings that require a lot of concurrent users and serializable isolated data. An auction bidding system could make use of this as the committing of a transaction would be seen as a complete bid. It would probably not suit the use of a ticketing DBMS for seat purchasing at events. This would be tricky to implement with StrongDBMS as customers that want to book seats together may have their transaction rolled back several times before being able to commit.

Due to its transparency because of the append-only structure it could prove a useful tool for a business seeking greater auditing capabilities than current DBMSs. However due to its limited throughput it would not likely be chosen for low concurrency environments.

This project has the potential to demonstrate the advantages of a stream-lined RDBMS that’s strengths lie in a limited field, compared to major RDBMS products that offer a wide range of features but have too large a scope to be effective in particular scenarios. This is all speculation, as StrongDBMS is still in the early developmental stages. This project will help provide a benchmark from which to help progress with the development of StrongDBMS. It will also help with the future modification of the TPC-C implementation should it be deemed necessary to make it closer related to official TPC-C or indeed make alterations to suit future testing parameters based on TPC-C but with significant differences.

This project is essentially a test on experimental software and established software using experimental software. Testing experimental software with other experimental software can have significant uncertainty in the results. However, as it is early in the development stage it is a very useful exercise for feedback into the development of both StrongDBMS and the TPC-C implementation. More work is needed on both the TPC-C application and StrongDBMS. The TPC-C needs improved in regard to the way it handles all the DBMSs and it needs to be universally validated as being a fair test before results can be conclusive. Once the TPC-C implementation runs fairly, StrongDBMS’s speed may need improvement for it to become a desirable alternative to the mainstream DBMSs.

10.4 LIMITATIONS

This TPC-C application has been implemented with variations to the original TPC-C specifications so cannot be held to the same level of authority as official benchmarks. There is a source for bias in that StrongDBMS and the TPC-C implementation were both designed by the same authors so the TPC-C application might run slightly better for StrongDBMS.

Strong and the TPC-C application are both experimental and will need wider scrutiny from the database community to validate their capabilities. Commercial DBMS concurrency results are too varied after 10 clerks to trust conclusively. Consistency results suggest that a stricter TPC-C test is needed to gain acceptance amongst the wider database community. However, the pattern across the consistency checks suggests that the DBMSs are being tested equally in terms of consistency as they all passed and failed the same tests. The exception was StrongDBMS, but the queries encountered a bug that does not raise an exception message and it cannot be said that StrongDBMS actually passed or failed the consistency checks.

Official TPC-C benchmarks are usually run on state-of-the-art equipment to showcase the capabilities of the system and setup. As it is not practical or realistic to implement high-end hardware to replicate such a test, the performance results are likely to be less favourable than might be expected of an official TPC-C test.

10.5 Future Work

Possible avenues of future development can be summarized in the following bullet points:

- Check the TPC-C implementation's testing of PostgreSQL to make sure it is being tested fairly.
- Improve throughput of StrongDBMS as that was its weakest area of performance.
- Run benchmark on hardware that employs NVMe SSD technology.
- Send a finished version of StrongDBMS to casual users, with RDBMS experience, for feedback on things such how user friendly it is and how easy the manual is to interpret.
- Optimise TPC-C implementation per DBMS to get maximum performance out of each and a more consistent performance out of Commercial DBMS.
- Test lower isolation settings of other DBMSs to see how much isolation needs to be sacrificed to improve concurrency.
- Stricter TPC-C implementation to test ACID properties of each DBMS more thoroughly.
- Run StrongDBMS in a benchmark which would test scalability, possibly partitioned over several servers.
- Run StrongDBMS in academic benchmarks such as ACIDRain (Warszawski & Bailis, 2017) or SEATS (Difallah, et al., 2013) to further showcase its concurrency handling abilities.
- Add more features to StrongDBMS such as a server management application similar to pgAdmin4 for PostgreSQL.
- Implement shareable data structures in current RDBMS projects such as Pyrrho.

10.6 Critical Appraisal

Knowing what I know now about the different isolation levels of each DBMS, I would have steered away from using one of Oracle or SQL Server as the Commercial DBMS. The user agreement terms that prevent either from being identified takes away from the transparency of the report and prevents or limits discussion on a number of key issues related to performance. One of the experts also brought to my attention that it was not a suitable comparison anyway given their differences in serializable isolation level from StrongDBMS and PostgreSQL. I would recommend anyone pursuing further work in this field to pick a DBMS that does not restrict disclosure of benchmark testing and that also employs similar isolation characteristics to StrongDBMS for an effective comparison.

The TPC-C application and StrongDBMS server and client application are both written in Visual Studio using .NET and C#. Any student attempting a related study to this in future would benefit hugely from prior C# or .NET experience using MS Visual Studio and would be able to contribute more than I was able to, with greater autonomy. I was restricted in what I could offer in terms of the development of the code due to these limitations. I was able to change the TPC-C application code to fit the syntax required for PostgreSQL from existing TPC-C application files on the GitHub repository (Crowe, et al., 2019). However, this was only possible due my experience in using SQL rather than C# or .NET and I received a lot of help from Malcolm Crowe in terms of editing and debugging the source files.

Joining in Strong is more restrictive than that of conventional DBMSs in that the "where" clause can't be used to make joins, just constraints. I was able to implement this fairly easily due to experience using MS Access in the Database Design course at UWS as it uses similar joining syntax. Therefore, I would recommend users practice this method of joining to become proficient with querying using StrongDBMS's SQL. The syntax is available in the StrongDBMS user's manual (Crowe, 2019a).

As MS Excel is a very effective tool for data analysis and display, I would highly recommend it for collecting data in an experiment such as this. However, it is not without its issues. The TPC-C tests don't run at even intervals: the initial state is referred to as having 0 clerks, then it goes up to 1 clerk, and then the clerk incrementations vary depending on each DBMS and what increments seemed appropriate. Conventional line graphs are useless for displaying this data, as I found initially, as they display every increment, no matter the difference, evenly. This is not an effective way of displaying data as it skews the appearance of the data. A scatter graph is required to display the increments evenly and on the same graph. The legend of the scatter graphs became an issue as I had 17 different data series, so it became very crowded. To overcome this, I deleted the legend and edited individual data point labels. These had the legend key displayed and the leader lines disabled to improve appearance. The data point labels were then individually dragged into position, one by one, to form a make-shift legend. This was not a very effective method because when the size or shape of the graph is changed, the legend keys move around in a manner that is not organised. Because of this I recommend leaving formatting changes such as this until the end of testing otherwise, users will have to keep formatting again and again. This also occurs when it is copied from Excel into a MS Word file. To stop this from happening, a screenshot was taken of the graph in an organised state and that was pasted into the word file. This is very time consuming. It was also difficult to display the 2000 New Order times in a box plot. In the end I opted for the scatter graph again to display the information in a similar make-shift manner. I have passed the MS Office Specialist exam for Excel 2016 and still found it frustrating to use at times. I recommend any future students undertaking an experiment in this manner to have had significant experience in creating, modifying and formatting Excel charts.

I found communicating with my supervisors over email and Skype (Microsoft, 2019d) to be very effective and efficient. Skype allowed for the meetings to take place without the need to travel into university for face-to-face meetings. I recorded the meetings, with permission from my supervisors, with Screencast-O-Matic (Screencast-O-Matic, 2019) rather than take minute notes by hand as I would have done in face-to-face meetings. This was very useful as I could refer back to the recordings to make notes or simply to remember solutions to issues that I had forgotten. Using Skype's screen share feature proved extremely helpful when editing C# or .NET files in Visual Studio as I could communicate my issues clearly to my supervisors and receive tutorials on them. As I am working from a PC instead of a laptop, I would have had to send my files to my supervisors and discuss it with them from their device at the university, but Skype allowed me to bypass this issue. Later, I would watch the recordings back to reaffirm techniques such as debugging the TPC-C applications files. It also allowed for quick calls when small issues arose which would not have been sufficient grounds for a conventional meeting and complex enough to warrant email an insufficient method for communication. I would highly recommend this approach to future students as it saved a lot of time and money that would have been spent traveling into university for conventional meetings.

11 ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my supervisors Prof Malcolm Crowe and Dr Santiago Matalonga. Their guidance, feedback, tutoring and support has been critical to the understanding and execution of this project. I have learned a lot and it has been a genuine pleasure working with them.

I am grateful to the members and associates of the DBTech community, who provided constructive feedback into the conclusion of this project and their expert analysis will help the development of StrongDBMS in the future. They did this through their own free time and their contributions have been very insightful.

Appreciation is due to Dr Junkang Feng who provided helpful feedback as moderator to this project. Dr Feng also initially sparked and nurtured my interest in the logic and theory behind relational database management systems through my classes in Database Design and Oracle Database Development.

12 APPENDIX

12.1 Benchmark Result Tables and Associated Graphs

		A	B	C	AVERAGE (s)
StrongDBMS	start	4:57:17 AM	5:06:36 AM	5:11:40 AM	
	end	5:00:14 AM	5:09:21 AM	5:14:35 AM	
	time	177	165	175	**172.3**
Commercial	start	4:13:40 AM	4:15:00 AM	4:16:10 AM	
	end	4:14:09 AM	4:15:27 AM	4:16:36 AM	
	time	29	27	26	**27.3**
PostgreSQL	start	3:42:55 AM	3:44:44 AM	3:47:15 AM	
	end	3:43:32 AM	3:45:22 AM	3:47:53 AM	
	time	37	38	38	**37.7**

Table 8: 2000 New Order Throughput Test

The rest of the raw data (with the exception of Commercial DBMS consistency checks as they would reveal the tested DBMS) is found at https://github.com/cfyffe1/TPC-C-Implementation-Raw-Data for convenience as it would not present well in this document.

12.2 Information and Questionnaire Sheet

Participant Information and Questions

Benchmarking StrongDBMS: An empirical evaluation of an immutable append only database

This research is being done to evaluate the development of an experimental Relational Database Management System called StrongDBMS. Please read the following to see what is involved. You may contact a member of the team for more information.

You are invited to comment on the results based on the experimental design of the research, which will be provided to you. It is hoped that this could assist in the development of Strong. Your thoughts on the tests/results are valued as a person with valuable database knowledge and experience. Your comments will be gathered along with other comments made by the DBTech community to provide insight into the experiment for discussion and highlight flaws or suggestions for future work. Comments will be accepted until Midnight on Sunday 11th August 2019.

Your details can be kept confidential if desired, and your comments will be attributed to "a member of the DBTech community".

Contact will be through email in an informal format. Participation is voluntary. Participants can withdraw at any moment from the experiment without any consequences. Data can also be withdrawn at any time without any consequences. The project is run by the University of the West of Scotland and has been reviewed by their ethics committee.

Questions can be answered by contacting b00343094@studentmail.uws.ac.uk

All the code, and documentation for StrongDBMS and the TPC-C implementation can be found at https://github.com/MalcolmCrowe/ShareableDataStructures

1 BACKGROUND

StrongDBMS is an experimental SQL RDBMS that is currently in development at the University of the West of Scotland (Crowe, 2019a). The transaction log is append-only allowing for admin and user transparency. In Strong, Shareable data structures are set up so that when a new data structure is created, the old data structure is not destroyed (immutable). This type of database structure design means you can access the older versions of the data structure which is known as persistent. The structure is partially persistent if you can only make changes to the current version of the data structure while maintaining access to older versions. It is fully persistent if you can modify every version and have access to them as well (Driscoll, et al., 1989). StrongDBMS currently supports partial persistence as the previous state of the database can be reconstructed using the Log table and truncating the transaction log. This could be utilised by a client that requires transparency with its data so that any appends can be traced back to a user. This discourages any users from manipulating the data in a malicious way as they won't be able to cover their tracks. Common features in RDBMS, such as users, permissions and roles, are being added to StrongDBMS in its developmental stage that should allow it to compete on a performance level with established RDBMSs in a benchmark test. It is open source and free to use. (Crowe, 2019a; Crowe, 2019b).

StrongDBMS has a strict definition of Serializable and it is its only transaction level isolation setting. It allows a user to attempt to manipulate data that is already in mid-transaction by another user but will only allow one user to commit. The loser in this situation will have their transaction rolled back. This is an optimistic, competitive approach to transactions (Crowe, 2019a) compared to PostgreSQL, Oracle and SQL Server. These established DBMSs use the pessimistic concurrency control of locking to deny access to any data in use by a transaction. StrongDBMS does not require any locking. This optimistic approach means that first committer wins. In locking, the first person to begin a transaction using the data wins by locking the affected data. This can lead to delays in data entry as a transaction can lock data for long periods of time unless a timeout period is set for transactions. Strong's optimistic approach may be more frustrating for concurrent users as there is no guarantee when the user starts their transaction that they will be the first to commit. However, it does encourage more efficient data entry at higher levels of concurrency.

The TPC-C implemented in these benchmark tests varies in a number of ways from the official TPC-C benchmark (Transaction Processing Performance Council, 2010). In this particular adaption of TPC-C there is only 1 warehouse. An official TPC-C benchmark may run with 1 warehouse, but it can incorporate several to test the ability of a DBMS to handle large amount of data and allows for more complex transactions to take place. This TPC-C implementation is run over the course of 10 minutes and the amount of New Orders that are processed in the 10 minutes are used to evaluate which RDBMS has superior concurrency handling capabilities in a saturated, data warehouse environment. In TPC-C there is a method for calculating the price of running the database for 3 years, this has been discarded for these benchmark tests. Since the TPC-C implementation is used mainly for concurrency testing there is little justification for running it at TPC-C's required minimum duration of 120 minutes. 10 minutes is deemed a sufficient amount of time to analyse conflict within each RDBMS. In TPC-C, each warehouse has 10 terminals that request transactions whereas, in the TPC-C implementation, terminals are referred to as clerks. The number of clerks is a variable rather than a fixed number to test the transaction conflict resolution and concurrency capabilities of the DBMS that is being tested. The TPC-C implementation does not take an average throughput measurement by TPC-C standards. This is because the number of New Orders generated includes New Order rows acquired in the "Ramp-Up" phase which occurs upon initialisation. This "Ramp-Up" phase is where tpmC (New Order transactions per minute) are still increasing due to the TPC-C application "warming up”. A steady tpmC is needed to measure throughput which occurs after “Ramp-Up” and before “Ramp-Down”, in a period of time where the test is generating a fairly consistent throughput. The results from the 10-minute New Order concurrency test are seen in Figure 2.

There is another mode in this TPC-C implementation which tests how quickly each DBMS server can process 2000 New Orders. This is shown in Figure 1.

The tests are run on a PC with a 64-bit Windows 10 operating system on a PNY CS900 SATA SSD, i5-4460 3.2 GHz processor with 4 cores, 8 GB of RAM, with a NVIDIA GeForce GTX 750 Ti graphics card. The RDBMS’s servers, client applications and TPC-C data will be accessed from the drive on a Seagate ST1000DM003 Barracuda HDD.

2 BIBLIOGRAPHY

Crowe, M., 2019a. StrongDBMS, Paisley, UK: University of The West of Scotland.

Crowe, M., 2019b. Shareable Data Structures, Paisley, UK: University of The West of Scotland.

Driscoll, J. R., Sarnak, N., Sleator, D. D. & Tarjan, R. E., 1989. Making Data Structures Persistent. Journal of Computer and System Sciences, 38(1), pp. 86 - 124.

Transaction Processing Performance Council, 2010. TPC Benchmark C. TPC Benchmark C, Volume 5.11.

3 QUESTIONS

1. What recommendations would you make for the development of StrongDBMS based on these results?

2. Would you be able to comment on the concurrency of the tested DBMSs based on the 10-Minute New Orders test results as seen in Figure 2?

3. Would you be able to comment on the performance of each DBMS based on the 2000 New Orders test results as seen in Figure 1?

4. How would you expect the results to differ had an official TPC-C been carried out on StrongDBMS?

5. What benchmarks, other than TPC-C, would benefit the development of StrongDBMS?

6. Any other comments/insights you have?

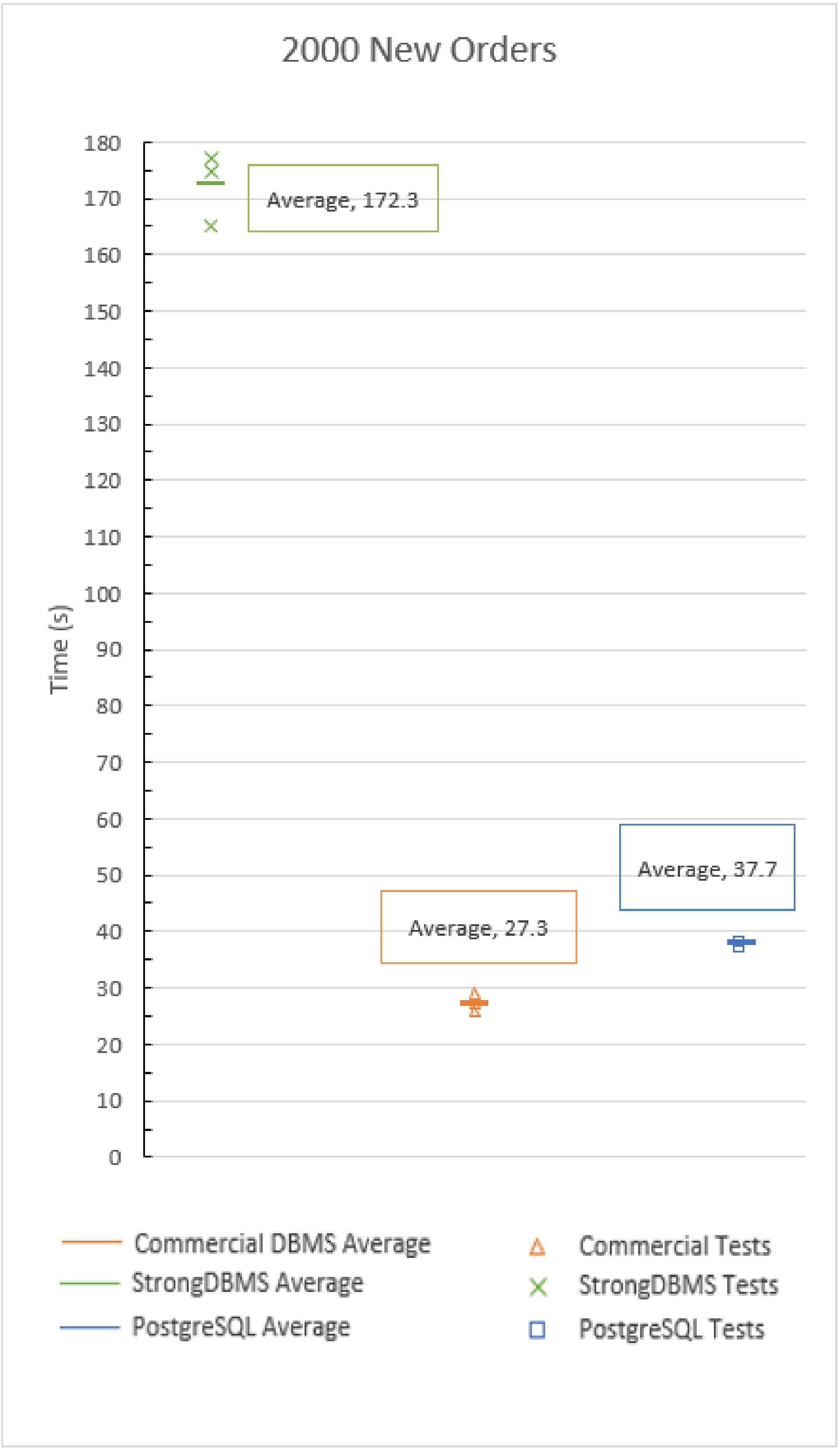


Figure 1: 2000 New Order Test for a Single User

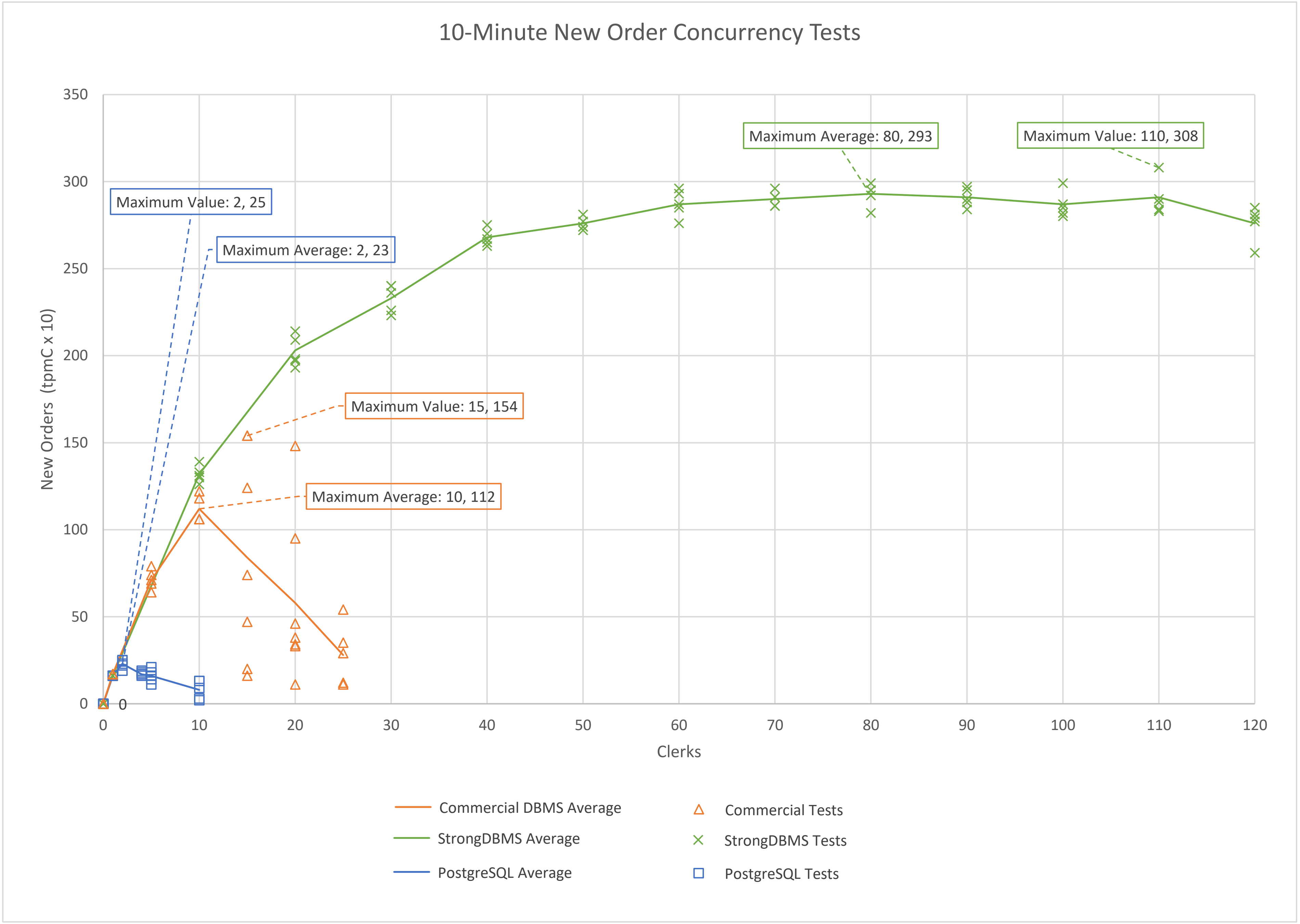


Figure 2: 10-Minute New Order Concurrency Tests

Thank you for your time and co-operation. It is very much appreciated, and I am very interested in your expert opinion.

Callum Fyffe

MSc IT Student at University of the West of Scotland

12.3 Email Reponses From Expert Community

12.3.1 Expert A

What recommendations would you make for the development of StrongDBMS based on these results?

The results are promising and show a better approach to data sharing and transaction isolation than other products. But the development of StrongDBMS was merely to demonstrate that “even a relational DBMS” could be implemented using shareable data structures. The demonstration chosen was to use the TPCC benchmark to verify ACID in a basic relational database even in conditions of high transaction conflict. By limiting the implementation to this purpose, the current implementation fails to demonstrate other features commonly associated with relational DBMS products, such as views, triggers, stored procedures and structured types, leaving room for doubt on the claim made for shareable data structures.

There are several possible directions for development.

First, work could begin to extend the implementation of StrongDBMS to include such features. The differences between StrongDBMS syntax and standard SQL are not inconsiderable, could become a nuisance in this process. It would become ever more awkward that the parsing engine (on the client) has no idea of the significance of identifiers used in the SQL statements it is parsing. This is no doubt one of the reasons why Strong DBMS is slower than its rivals in the 2000-new orders test. See also the answer to Q5 below.

Second, rather than build on the StrongDBMS codebase, the basic principles of shareable structures could be used to redevelop an existing SQL-compliant DBMS. This is currently happening for Pyrrho[1], whose performance in the TPCC benchmark has declined badly since the results Crowe reported[2] in 2005.

The opportunities for applying these principles to other DBMS are more limited. StrongDBMS and Pyrrho share a very literal, idiosyncratic definition of transaction isolation. It is very doubtful that shareable data structures would be as useful a basis for developing DBMS with a weaker isolation mode such as READ UNCOMMITTED, or even REPEATABLE READ. These are the isolation levels most commonly used for application development, so that such a change to an existing DBMS, other than the above-mentioned, would probably break the vast majority of database applications.

Would you be able to comment on the concurrency of the tested DBMSs based on the 10-Minute New Orders test results as seen in Error! Reference source not found.**?**

Clearly StrongDBMS performs better than the other DBMS in this benchmark. But the model of OLTP processing found in TPCC is no longer the typical use case for DBMS. Almost all DBMS are now behind a web site. Today, web applications have an option to gather user input so that a lengthy sequence of user interactions with a web page are followed by a time-limited transaction with the database(s). Thus, the DBMS concurrency is between a small number of web server threads rather than the potentially very large number of users of the web site. The results of the TPCC benchmark for the DBMS would assist in tuning this balance.

[1] http://pyrrhodb.blogspot.com/ post of 24 June 2019

[2] Crowe, M.K. (2005): Transactions in the Pyrrho Database Engine, in Hamza M.H. (ed) *Proceedings of the IASTED International Conference on Databases and Applications, Innsbruck, February 2005* (IASTED, Anaheim) ISBN 0-88986-460-8 ISSN 1027-2666, p.71-76

Moreover, today, compensation processes in the event of transaction conflict are preferred over transaction aborts. The resulting architecture can solve the problems of distributed transactions: if more than one DBMS is involved the web servers are able to verify that all branch transactions have succeeded, and can apply compensation as required in the rare cases where heuristic completion is required.

Would you be able to comment on the performance of each DBMS based on the 2000 New Orders test results as seen in Error! Reference source not found.**?**

These clearly show that StrongDBMS is much slower than the other databases tested. Many commercial users of DBMS would regard this aspect as crucial, and in view of the answers to 2 above, would prefer faster DBMS response, and tune concurrency as described above.

How would you expect the results to differ had an official TPC-C been carried out on StrongDBMS?

It is estimated that the TPCC as set out in the specification (one clerk per warehouse) results in only 4% of conflicting transactions. That would suggest that commercial DBMS would handle everything perfectly until the number of warehouses exceeded 250. Moreover, the TPCC specification does not specify the SERIALIZABLE isolation mode, which these DBMS products actually advise against. Existing commercial DBMS regard their performance on the TPCC benchmark are entirely satisfactory.

What benchmarks, other than TPC-C, would benefit the development of StrongDBMS?

Although TPCC is one of the best-known DBMS benchmarks, there are benchmarks that target other use cases. As a minimum StrongDBMS could be shown to work well in web site development. In such a use case, all DBMS requests would be known in advance and prepared in binary form with the server's internal identifiers which would bring performance benefits.

Any other comments/insights you have?

StrongDBMS achieved its main purpose in demonstrating that shareable data structures are a practical alternative to data structures based on lists and arrays. There are associated demonstrations that shareable data structures can also scale well, so that there is an opportunity to include shareable data structures in mainstream Computing curricula, as an example of how to avoid locking in multithreaded programming.

It is less clear that traditional DBMS should be re-implemented using shareable data structures. There would be more benefit in other DBMS products adopting other innovations, such as RESTView technology[3].

[3] Crowe, M.K., Laux, F (2018).: DBMS Support for Big Live Data, https://www.iaria.org/conferences2018/filesDBKDA18/MalcolmCrowe_DBMS_Support.pdf

12.3.3 Expert B

Reply to “Participant Information and Questions"
Title: Benchmarking StrongDBMS:
An empirical evaluation of an immutable append only database
Author: Callum Fyffe
“Expert B”
Date: 08 August 2019

Initial Remarks

I am not very knowledgeable on benchmarking, so I will leave it to others to comment on how appropriate the benchmarks which you have used are. I do know a little about transaction isolation, and I believe that there are issues in your study which require paying careful attention to isolation issues, so I will comment some on that. Also, I have considerable experience as a user of PostgreSQL, and I will comment on its use in your studies.

I feel that it is necessary to provide some background before answering your questions, which I do in the sections entitled “Transaction Isolation" and “Performance of Reference Systems" below.

Transaction Isolation

First of all, I assume that all tests using X and PostgreSQL were run using SERIALIZABLE isolation for all transactions. The level of isolation used for the reference systems should definitely be stated explicitly in any report.

It is very important to note and to understand that SERIALIZABLE levels of the various commercial and open-source systems differ from each other as much, if not more, than, they do from that of StrongDBMS. In fact, I would argue that, in terms of SERIALIZABLE isolation, StrongDBMS and PostgreSQL have more in common with each other than either does with “*Commercial DBMS”*. From my perspective, this makes “*Commercial DBMS”* a questionable choice for a reference system. Since you paint a somewhat different picture, I believe that some clarification is appropriate.

“SECTION REMOVED FOR COMMERCIAL DBMS ANONYMITY”

SERIALIZABLE isolation in PostgreSQL The SERIALIZABLE isolation level of PostgreSQL is implemented using serializable snapshot isolation, or SSI. It provides true conflict serializability. It is a recent approach, which builds upon standard snapshot isolation (SI). Ordinary SI, while providing a relatively high level of isolation, does not detect read-write conflicts (rw-conflicts), and so cannot provide true conflict serializability. In SSI, so called dangerous structures (consisting of two or three transactions with special properties) are prohibited. A special process, built on top of SI, detects such structures and eliminates them, usually at commit time, by aborting a participating transaction.

The SI isolation level prohibits writes of the same data object x by concurrent transactions. Since SSI is an extension of SI, it exhibits the same behavior. In PostgreSQL, such write-write conflicts (ww-conflicts) are resolved using first-updater wins (FUW). That is, the first transaction to request an update of x is the winner. If the winner commits, the loser must be aborted and be rerun. If the winner does not commit, the loser may continue. Thus, with FUW, writers may block other writers. On the other hand, it is important to note that since dangerous structures are generally detected at commit time, readers and writers cannot block each other under the implementation of SSI employed by PostgreSQL.

SERIALIZABLE isolation in StrongDBMS As you have noted, StrongDBMS provides only SERIALIZABLE isolation. As is the case with PostgreSQL, it is implemented as an extension of SI. However, it differs in two ways. First, ww-conflicts are resolved via first-committer wins (FCW), in which such conflicts are resolved at commit time, so writers do not block each other. Second, instead of detecting and eliminating dangerous structures, serializability is guaranteed by prohibiting so-called backward rw-dependencies; that is, if T1 reads a data object x and T2 writes x concurrently, then T1 must commit before T2. Otherwise, at least one of T1 or T2 must be aborted and rerun. It is provable that SSI results in strictly fewer false positives (hence more concurrency) than does the strategy of StrongDBMS.

Comparison of SERIALIZABLE in the reference systems In view of the above descriptions, the following can be concluded. *"SECTION REMOVED FOR COMMERCIAL DBMS ANONYMITY".* I believe that it is a poor choice for a benchmark comparison to StrongDBMS.

PostgreSQL employs the state-of-the-art strategy SSI for the implementation of SERIALIZABLE isolation. While the FUW policy for conflict resolution of PostgreSQL does result in some blocking for ww-conflicts, it must be remembered that with such a conflict, only one transaction can commit anyway. The question of which of FCW and FUW is better is a difficult empirical question, and likely depends upon transaction mix. Both work well, and I would not expect to see a huge difference in the performance of StrongDBMS and PostgreSQL due to the choice of conflict-resolution strategy. As for rw-conflicts, PostgreSQL will have fewer false positives, although the degree to which this matters is open to study. The bottom line is that while I feel that PostgreSQL is a good choice for a comparison to StrongDBMS, I do not see any reason, based upon their implementations of SERIALIZABLE isolation, why StrongDBMS would greatly outperform it. I will comment more on this in the next section.

SERIALIZABLE isolation in other major systems IBM Db2 does not provide a SERIALIZABLE isolation level. "SECTION REMOVED FOR COMMERCIAL DBMS ANONYMITY". Similarly, MySQL/MariaDB implements its SERIALIZABLE level of isolation as SI. Whether this is in compliance with the SQL standard is debatable, but in any case there are major DBMSs which do not implement SERIALIZABLE isolation in a way which provides what is technically called serializable isolation. Thus, none of these systems would seem to provide a good benchmark comparison to StrongDBMS. PostgreSQL is clearly the best choice.

"SECTION REMOVED FOR COMMERCIAL DBMS ANONYMITY"

Performance of the Reference Systems

Performance of PostgreSQL The extremely poor performance of PostgreSQL, particularly as documented in Figure 2, is almost unbelievable. It is a mature, highly optimized system, which is widely used in diverse and demanding applications. Before these results can be published, it is necessary to make absolutely certain that a programming error is not involved. Given the way that performance tails o_ to almost nothing beyond a modest number of clerks, my suspicion is that there is a subtle bug with the result that aborted transactions are not always restarted correctly. As a test strategy, I suggest tracing a number of transactions, to make certain that they are rerun correctly after an abort. This will be tedious work, but it is necessary. Also, scrutinize the PostgreSQL logs carefully to see whether other problems are flagged, such as exceeding the maximum number of connections. Perhaps new connections are being created without terminating the finished ones properly. If PostgreSQL indeed performs poorly on this benchmark, it will be necessary to show why it fails when StrongDBMS succeeds. However, given the absolutely dismal results, I feel that it is far more likely that a programming error or system misconfiguration is involved.

"SECTION REMOVED FOR COMMERCIAL DBMS ANONYMITY"

Answers to Questions Posed

Here I will try to answer your questions, but in many cases they have already been discussed in more detail above.

What recommendations would you make for the development of StrongDBMS based on these results?

I cannot recommend much of anything until the possibility of errors in the profiles of the reference systems, particularly PostgreSQL, have been ruled out. I do not have confidence in the validity of the results at this point.

Would you be able to comment on the concurrency of the tested DBMSs based on the 10-Minute New Orders test results as seen in Figure 2?

As noted above, I have a strong suspicion that the results for PostgreSQL in particular are in error. The data show it collapsing almost completely under the pressure of high concurrency. Since it is a mature system, used in many demanding applications with great success, it seems that the possibility of a programming error is a far more likely explanation.

Would you be able to comment on the performance of each DBMS based on the 2000 New Orders test results as seen in Figure 1?

The comments for Figure 2 also apply here. I would need additional, convincing assurances that programming or configuration errors are not involved before accepting the results.

How would you expect the results to differ had an official TPC-C been carried out on StrongDBMS?

This issue lies outside of my expertise; I cannot answer that.

What benchmarks, other than TPC-C, would benefit the development of StrongDBMS?

This issue lies outside of my expertise; I cannot answer that.

Any other comments/insights you have?

The idea of using immutable data structures in a DBMS is a very interesting one. However, even after any errors in the profiles of the reference systems are corrected, I do not see any strong grounds to conclude that the performance differences between StrongDBMS and the reference systems are due primarily to this aspect of StrongDBMS. The systems differ in many other ways as well. To evaluate the benefits of immutable data structures would require other, far different studies.

12.3.4 Expert C

Dear Callum,

The results are surprising and highly interesting. Especially the more than
10 times higher throughput than PostgreSQL under concurrency is remarkable.
Malcolm and you have done a great job!

I find it difficult to make recommendations as I do not know sufficient details about the TPC-C benchmark and your implementation. For example the TPC-C document states in 5.1.2 "... New-Order transactions that are rolled back, as required by Clause 2.4.1.4, are considered as completed transactions". Where 2.4.1.4 requests 1% of New-Order transactions (tx) should have entry errors which leads to a rollback but are still considered as "completed transactions". How do you handle concurrency conflicts?

So, first of all I need to speculate to comment the results:

As the main focus was on concurrency testing, it is important how you deal with concurrency conflicts. Are these aborted tx retried or just ignored? If they are ignored I suspect that the next transaction has a chance to not conflict with other concurrent tx because the keys and other possibly conflicting data is chosen randomly.
In this case StrongDB would have an advantage over locking because in the latter case the tx are queued until the conflict disappears leading to lower throughput/concurrency. This might be a possible explanation for figure 2.

The lower batch performance (2000 New Orders, Figure 1) is surprising as there is no concurrency at all and it is only a matter of how many operations you need for a New Order. I would have guessed that StrongDB has a leaner implementation because some functions of commercial system are not implemented yet and therefore should outperform the other DBMS.
As this is not the case a detailed performance analysis is needed. (Does VisualStudio/C# allow to measure how long a method executes or how long a system call takes?) If I am allowed to speculate again, it could be the case that with nearly each new transaction being written to the end of the logfile, the file system may need to allocate more disc space for the growing file. Is it possible to control the file extension mechanism in C#?
Commercial Systems usually preallocate large file extents to avoid such time consuming allocation. Oracle e.g. even allocates a contiguous disc space for its database files.

Recommendations:
1. Find the reason why StrongDB is 4-5 times slower to process New Orders
sequentially. Then tune the implementation accordingly.
2. Figure 2 shows that StrongDB delivers not only a higher concurrency
but also a better stability when overloaded. Find an explanation why this is the case and why *"Commercial DBMS"* shows such a great variance in concurrency (esp. with high concurrency)
3. It seems that *"Commercial DBMS"* and PostgreSQL are optimized for conflict-free
throughput.
4. Distributing data to multiple discs is a common tuning measure as disc
access is 3 orders of magnitude slower than memory access. Therefore I expect that the commercial products already support this possibility (multiple writer processes). This could lead to better results with New Orders but have no benefit for higher concurrency.
5. The TPC-E benchmark would be interesting. It is also a OLTP benchmark
but the transactions and the DB-Schema are more complex.
6. I would also be interested in a hot-spot scenario. E.g. Inventory

management, seat reservation in a aircraft, trading, etc.

Best regards
"Expert C"

12.4 PROJECT SPECIFICATION

University of the West of Scotland

School of Engineering and Computing

MSc Project Specification

Student name: Callum Fyffe

Banner ID: B00343094

Email: B00343094@studentmail.uws.ac.uk

MSc Programme/stream: Information Technology

MSc Programme Leader: Costas Iliopoulos

Project Title:

Benchmarking StrongDBMS: An empirical evaluation of an immutable append only database

Research Question to be answered:

How does Strong deal with a strict TPC-C benchmark in comparison to other major RDBMS? Why does Strong and the other major RDBMS perform as they do when benchmarked using TPC-C?

Outline (overview) and overall aim of project:

StrongDBMS (Strong) is currently in development and has unique features that make it stand out against other well-established relational database management systems (RDBMS). Compared to other database system providers, it has extremely strict Atomicity, Consistency, Isolation and Durability (ACID) characteristics. Therefore, the data in a Strong database should be more consistent than data in its established competitors as their default configurations tend to allow for anomalies to occur within the data. However, it is still in the early stages of development, having less overheads than other systems so it is likely to perform better until more complex features, that other products already boast, are added. It is designed with append-only, sharable data structures, allowing any changes made to the database to be monitored and accessed. This allows for greater transparency when looking into the history of a database. It also treats transaction isolation and locking very differently to other database software. It has already been tested against the Transaction Processing Performance Council's (TPC) TPC-C benchmark, but it has had a number of new features added to it since which may further slow down its transaction processing time. This benchmark is a good indicator of whether a database can perform adequately in a situation that requires a large amounts of data generation and gathering, that can handle multiple

transactions, and that demonstrates strong ACID properties. Strong and two highly regarded RDBMS products will be tested in identical TPC-C tests to compare results and gauge the development of Strong. This is relevant due to the advancement of Non-Volatile Memory express (NVMe) Solid-State Drives (SSD) as data storage and retrieval is about to become much faster, and there is a gap in the database market for smaller, specialised DBMS. This project will demonstrate the advantages of a stream-lined RDBMS that's strengths lie in a limited field, compared to major RDBMS products that offer a wide range of features but have too large a scope to be effective in particular scenarios.

Keywords: StrongDBMS, ACID, Shareable Data Structures, Relational Database Management System, TPC-C

Objectives (list of tasks to be undertaken to achieve overall aim of the project and to answer the research question posed):

Some literature research will be done into the documentation of the TPC-C benchmark, StrongDBMS, shareable data structures, the other benchmarked RDBMS and previous benchmark attempts on them. Other benchmarking methods other than TPC-C will also be investigated as a consideration for future work to be carried out.

To understand the background of StrongDBMS and compare the ACID characteristics of it against more popular RDBMS. The under-development StrongDBMS will be evaluated to see what steps need to be taken to improve it. As more features are added it will be retested to determine the effect of new features on performance. This comparison can also be used to highlight strengths and weaknesses of the RDBMSs and recommend their use for specific situations. This research will be integrated into the development of StrongDBMS rather than a finished product summary.

To highlight how far StrongDBMS has progressed under development by comparing the results of an identical TPC-C benchmark run on StrongDBMS against other RDBMS with the collaboration of Academics from the University of the West of Scotland (UWS).

After the results are analysed, the literature will be revised to provide suggestions on what market StrongDBMS could eventually target based on its advantages over more popular, established RDBMS.

Relationship of proposed project to MSc programme/stream:

MSc Information Technology teaches the students a lot to do with SQL relational database theory and implementations that are mostly Oracle related. This project will compliment and contrast the PgDip coursework by looking into other RDBMS and their inner workings, especially when referring to ACID properties and the handling of transactions while relying on the previous knowledge gained in courses such as Database Design, Oracle Database Development, and Database Applications for Business. It also gives an insight into the choices a database administrator might have to face in a data warehouse scenario when trying to compromise between performance and consistency as well as picking the right DBMS for the job.

Indicative reading list and resources:

12.4.1 Bibliography

Chen, Y., Raab, F., & Katz, R. (2012). From TPC-C to Big Data Benchmarks: A functional Workload Model. *Specifying Big Data Benchmarks*, 28 - 43.

Crowe, M. (2019a). *StrongDBMS.* Paisley, UK: University of The West of Scotland.

Crowe, M. (2019b). *Shareable Data Structures.* Paisley, UK: University of The West of Scotland.

Crowe, M. (2019c). *MalcolmCrowe/ShareableDataStructures*. Retrieved April 18, 2019, from https://github.com/MalcolmCrowe/ShareableDataStructures

Crowe, M., Matalonga, S., & Laiho, M. (2019). StrongDBMS: a DBMS built from immutable components. Retrieved April 18, 2019, from https://www.researchgate.net/publication/332071384_Strong_a_DBMS_built_from_immutable_components

Llanos, D. R. (2006). TPCC-UVa: An Open-Source TPC-C Implementation for Global Performance Measurement of Computer Systems. *SIGMOD Record, 35*(4).

Micorsoft. (2019, April 23). *SQL Server Documentation.* Retrieved from Microsoft.com: https://docs.microsoft.com/en-us/sql/sql-server/sql-server-technical-documentation?view=sql-server-2017

Ports, D. R., & Grittner, K. (2012). Serializeable Snapshot Isolation in PostgreSQL. *Proceedings of the VLDB Endowment, 5*(12), 1850-1861.

PostgreSQL Global Development Group. (2019). *PostgreSQL 11.3 Documentation.* Retrieved May 13, 2019, from https://www.postgresql.org/files/documentation/pdf/11/postgresql-11-A4.pdf

Transaction Processing Performance Council. (2019). *TPC*. Retrieved April 08, 2019, from http://www.tpc.org/default.asp

Warszawski, T., & Bailis, P. (2017). ACIDRain: Concurrency-Related Attacks on Database-Backed Web Applications. *Proceedings of the 2017 ACM International Conference on Management of Data*, 5 - 20.

Marking scheme:

- Rigour of Literature Review 20% including
  - Method, documentation, and report in the dissertation
- Experimental evaluation of StrongDB 25%
  - Documentation of the Evaluation strategy
  - Presentation of results.
- Discussion 20%
  - Critical discussion that adds to the body of knowledge

- Validation with the DBTech Community 10%
- Conclusions / Recommendations 10%
- Critical Evaluation 10%
    - With lessons learned about the research process and recommendation for other students performing a similar research.
- Coherence of the technical report 5% including
    - Readability; internal structure (objective, aims, scope, etc), English language and grammar.

Supervisor:

Santiago Matalonga; Malcolm Crowe

Moderator:

Junkang Feng

Programme Leader:

Costas Iliopoulos

Date specification submitted:

10/06/2019

Please complete the 'ethics' form below for all projects.

School of Engineering and Computing

MSc PROJECT - ETHICAL CONSIDERATION DOCUMENT

SECTION 1: TO BE COMPLETED BY THE STUDENT

Does your research involve: research with human subjects, access to company documents/records, questionnaires, surveys, focus groups and/or other interview techniques? (please enter Y in the appropriate box)

YES		complete the rest of the form **and** submit an application for approval to the Ethics Review Manager
NO	Yes	skip to the end of the form for supervisory sign off

Please answer ALL of the following questions concerning your proposed research:

	(please enter either Y or N in each box)	Y/N
1.	Are the participants and subjects from any vulnerable group or (e.g. NHS patients, children) or from a 'captive' audience (e.g. students)	N
2.	Are the participants and subjects of the study in any way unable to give free and informed consent within the meaning of the Mental Capacity Act 2005 to the best of your knowledge?	N
3.	Are you asking questions that are likely to be considered impertinent or to cause distress to any of the participants?	N
4.	Are any of the subjects in a special relationship with the applicant (e.g. colleagues)?	N
5.	Does your project pose any risk to either yourself or the participant?	N

If you have answered **Y** to **any** of the above questions, or are **unsure**, please contact your supervisor (students) or any member of the School Ethics Committee (Staff).

Please answer ALL of the following questions concerning your proposed research:

		Y/N
1.	Participants will be/have been advised that they may withdraw at any stage if they so wish.	Y
2.	Issues of confidentiality and arrangements for the storage and security of material during and after the project and for the disposal of material have been considered.	Y
3.	Arrangements for providing subjects with research results if they wish to have them have been considered.	Y
4.	The arrangements for publishing the research results and, if confidentiality might be affected, for obtaining written consent for this have been considered.	Y

5.	Information Sheets and Consent Forms had been prepared in line with University guidelines for distribution to participants.	Y
6.	Arrangements for the completed consent forms to be retained upon completion of the project have been made.	Y

If you have answered **N** or you **cannot confirm** the answer to **any** of the above questions, contact your supervisor (students) or any member of the School Ethics Committee (Staff).

SECTION 2: APPLICATION CHECK LIST FOR SUPERVISORS

Please enter Y, N or TBD* in all of these boxes. If you use TBC, then the documents must be produced to submit to the Ethics Review Manager (ERM). Without consent from the ERMyour project cannot proceed. Any project requiring Ethical Approval that does not receive Approval from the ERM will not be accepted for marking and will be deemed to have 'failed'.

I confirm that an Information Sheet and a Consent Form have been prepared and will be made available to all participants. These contain details of the project, contact details of the researcher and advise subjects that their privacy will be protected, their participation is voluntary and that they may withdraw at any time without reason.	
I confirm that research instruments (questionnaires, interview guides, etc) have been reviewed against the policies and criteria noted in The University Research Ethics Committee Notes for Guidance.	
I confirm that **all** related documents, including any questionnaires, interview schedules and copies of the Information Sheet and Consent Form, are attached and submitted with this application.	

*TBD issues need to be resolved during project management and prior to submission to the ERM

Supervisor (print name): Santiago Matalonga

Signature: SM

Date: 10/06/2019

IMPORTANT: please note that by signing this form all signatories are confirming that any potential ethical issues have been considered and necessary actions undertaken, and that Daune West (MSc Project Co-ordinator) and Malcolm Crowe (Chair of the School of Engineering and Computing Ethics Committee) have been advised of any potential ethical issues relating to the project proposed in the attached MSc Project specification.

BIBLIOGRAPHY

Abadi, D. & Faleiro, M., 2018. An Overview of Deterministic Database Systems. *Communications of the ACM,* 61(9), pp. 78 - 88.

Alexandrov, A., Brucke, C. & Markl, V., 2013. Issues in Big Data Testing and Benchmarking. *Proceedings of the Sixth International Workshop on Testing Database Systems.*

Alomari, M., Cahill, M., Fekete, A. & Rohm, U., 2008. The cost of serializability on platforms that use snapshot isolation. *2008 IEEE 24th International Conference on Data Engineering,* pp. 576 - 585.

Baru, C. et al., 2012. Setting the Direction for Big Data Benchmark Standards. *Selected Topics in Performance Evaluation and Benchmarking,* pp. 197 - 208.

Begg, C., Connolly, T. & Crowe, M., 2015. Introduction to Pyrrho: A Lightweight RDBMS. In: *Database Systems: A practical Approach to Design, Implementation, and Management.* Harlow, UK: Pearson, pp. E1-E18.

Cave, J., 2018. *Re: Implementing Optimistic Locking in Oracle [Online Discussion Group].* [Online]
Available at: https://stackoverflow.com/questions/41006941/implementing-optimistic-locking-in-oracle
[Accessed 7 August 2019].

Chen, Y., Raab, F. & Katz, R., 2012. From TPC-C to Big Data Benchmarks: A functional Workload Model. *Specifying Big Data Benchmarks,* pp. 28 - 43.

Cole, R. et al., 2011. The mixed workload CH-benCHmark. *Proceedings of the Fourth International Workshop on Testing Database Systems,* pp. 8 - 13.

Connolly, T. & Begg, C., 2015. *Database Systems: A Practical Approach to Design, Implementation and Management.* 6th ed. Harlow, UK: Pearson.

Crowe, M., 2015. *The Pyrrho Database Management System.* [Online]
Available at: https://pyrrhodb.uws.ac.uk/ThePyrrhoBook.pdf
[Accessed 6 August 2019].

Crowe, M., 2019a. *StrongDBMS,* Paisley, UK: University of The West of Scotland.

Crowe, M., 2019b. *Shareable Data Structures,* Paisley, UK: University of The West of Scotland.

Crowe, M., 2019c. *PYRRHO DBMS: Version 7 is coming (soon).* [Online]
Available at: http://pyrrhodb.blogspot.com/2019/06/version-7-is-coming-soon.html
[Accessed 10 August 2019].

Crowe, M., 2019d. *MalcolmCrowe/SharableDataStructures.* [Online]
Available at: https://github.com/MalcolmCrowe/ShareableDataStructures
[Accessed 07 July 2019].

Crowe, M., Begg, C., Laux, F. & Laiho, M., 2017. Data Validation for Big Live Data. *DBKDA ,* pp. 30 - 33.

Crowe, M. & Laux, F., 2018. *DBMS Support for Big Live Data.* [Online]
Available at:

https://www.iaria.org/conferences2018/filesDBKDA18/MalcolmCrowe_DBMS_Support.pdf
[Accessed 12 August 2019].

Crowe, M., Matalonga, S. & Laiho, M., 2019. StrongDBMS: Built from Immutable Components. *DBKDA,* pp. 11 - 16.

Darmont, J., Bentayeb, F. & Boussaid, O., 2017. *Benchmarking Data Warehouses,* Lyon, FRA: University of Lyon.

DBTech NET, 2019. *DBTechNet.* [Online]
Available at: http://blogit.haaga-helia.fi/dbtechnet/
[Accessed 26 June 2019].

Difallah, D. E., Pavlo, A., Curino, C. & Cudre-Mauroux, P., 2013. Oltp-bench: An extensible testbed for benchmarking relational databases. *Proceedings of the VLDB Endowment,* 7(4), pp. 277 - 288.

Driscoll, J. R., Sarnak, N., Sleator, D. D. & Tarjan, R. E., 1989. Making Data Structures Persistent. *Journal of Computer and System Sciences,* 38(1), pp. 86 - 124.

Fekete, A. et al., 2005. Making Snapshot Isolation Serializable. *ACM Transactions on Database Systems,* 30(2), pp. 492 - 528.

Fekete, A., O'Neil, E. & O'Neil, P., 2004. A Read-Only Transaction Anomaly Under Snapshot Isolation. *ACM SIGMOD Record ,* 33(3), pp. 12 - 14.

Feuerstein, S., 2014. *Oracle PL/SQL Programming.* 6th ed. Sebastopol, USA: O'Reilly Media.

Fiannaca, A. J. & Huang, J., 2015. *Benchmarking of relational and nosql databases to determine constraints for querying robot execution logs.,* Washington, USA: Computer Science & Engineering, University of Washington.

Giles, D., 2019. *All Things Database: Education, Best Practices, Use Cases & More.* [Online]
Available at: https://blogs.oracle.com/database/oracle-database-19c-available-exadata
[Accessed 01 August 2019].

Harding, R., Van Aken, D., Pavlo, A. & Stonebreaker, M., 2017. An Evaluation of Distriibuted Concurrency Control. *Proceedings of the VLDB Endowment,* 10(5), pp. 553 - 564.

IARIA, 2019. *The Ninth International Conference on Smart Grids, Green Communications and IT Energy-aware Technologies.* [Online]
Available at: https://www.iaria.org/conferences2019/ProgramENERGY19.html
[Accessed 30 July 2019].

IBM Software Group Information Management, 2009. *Telecom Application Transaction Processing Benchmark.* [Online]
Available at: http://tatpbenchmark.sourceforge.net/TATP_Description.pdf
[Accessed 19 June 2019].

Laiho, M., Dervos, D. A. & Silpio, K., 2013. *SQL Transactions.* 1 ed. s.l.:DBTech VET Teachers Project.

Laiho, M. & Laux, F., 2011. *On SQL Concurrency Technologies,* Helsinki, Finland; Reutlingen, Germany: DBTechNet.

Linaker, J. & de Mello, R. M., 2015. *Guidelines for Conducting Surveys in Software Engineering.* [Online]

Available at:
https://www.researchgate.net/publication/276062061_Guidelines_for_Conducting_Surveys_in_Software_Engineering
[Accessed 13 August 2019].

Llanos, D. R., 2006. TPCC-UVa: An Open Source TPC-C Implementation for Global Performance Measurement of Computer Systems. *ACM SIGMOD Record,* 35(4), pp. 6 - 15.

Lüttgau, J. et al., 2018. Survey of Storage Systems for High-Performance Computing. *Supercomputing Frontiers and Innovations,* 5(1), pp. 31 - 58.

Microsoft, 2019a. *SQL Server Documentation.* [Online]
Available at: https://docs.microsoft.com/en-us/sql/sql-server/sql-server-technical-documentation?view=sql-server-2017
[Accessed 23 April 2019].

Microsoft, 2019b. *What's new in SQL Server 2019 preview.* [Online]
Available at: https://docs.microsoft.com/en-us/sql/sql-server/what-s-new-in-sql-server-ver15?view=sqlallproducts-allversions
[Accessed 11 June 2019].

Microsoft, 2019c. *Services Provider Use Rights,* s.l.: Microsoft.

Microsoft, 2019d. *Skype makes it easy to stay in touch.* [Online]
Available at: https://www.skype.com/en/
[Accessed 29 July 2019].

Nambiar, R. et al., 2012. TPC Benchmark Roadmap 2012. *Selected Topics in Performance Evaluation and Benchmarking,* pp. 1 - 20.

Oracle, 2013. *TPC Benchmark C Full Disclosure Report,* s.l.: Oracle America.

Oracle, 2018. *Oracle Technology Network Developer License Terms.* [Online]
Available at: https://www.oracle.com/technetwork/licenses/db18c-express-license-5137264.html
[Accessed 11 April 2019].

Oracle, 2019a. *Oracle Help Center.* [Online]
Available at: https://docs.oracle.com/en/
[Accessed 10 April 2019].

Oracle, 2019b. *Oracle.* [Online]
Available at: https://www.oracle.com/index.html
[Accessed 10 April 2019].

Poess, M., 2012. TPC's Benchmark Development Model: Making the First Industry Standard Benchmark on Big Data a Success. *Specifying Big Data Benchmarks,* pp. 1 - 10.

Poess, M. & Nambiar, R. O., 2008. Energy Cost, The key Challenge of Today's Data Centers: A power Consumption Analysis of TPC-C Results. *Proceedings of the VLDB Endowment,* 1(2), pp. 1229 - 1240.

PostgreSQL Global Development Group, 2019a. *PostgreSQL 11.3 Documentation.* [Online]
Available at: https://www.postgresql.org/files/documentation/pdf/11/postgresql-11-A4.pdf
[Accessed 13 May 2019].

PostgreSQL Global Development Group, 2019b. *PostgreSQL 11 Realeased!.* [Online]
Available at: https://www.postgresql.org/about/news/1894/
[Accessed 10 June 2019].

Runeson, P. & Host, M., 2009. Guidelines for conducting and reporting case study research in software engineering. *Empir Software Eng,* Volume 14, pp. 131-164.

SAP, 2014. *TPC Benchmark C Full Disclosure Report,* s.l.: SAP.

Screencast-O-Matic, 2019. *Video creation for everyone.* [Online]
Available at: https://screencast-o-matic.com/
[Accessed 29 July 2019].

Shanley, K., 1998. *History and Overview of the TPC.* [Online]
Available at: http://www.tpc.org/information/about/history.asp
[Accessed 10 August 2019].

Shao, J., Liu, X., Li, Y. & Liu, J., 2015. Database performance optimization for SQL Server based on hierarchical queuing network model.. *International Journal of Database Theory and Application,* 8(1), pp. 187 - 196.

solid IT, 2019. *DB-Engines.* [Online]
Available at: https://db-engines.com/en/
[Accessed 18 June 2019].

Spenik, M. & Sledge, O., 2003. *Microsoft SQL Server 2000 DBA survival guide,* s.l.: Sams Publishing.

Stackshare, 2019a. *PostgreSQL: A powerful, open source object-relational database system.* [Online]
Available at: https://stackshare.io/postgresql
[Accessed 7 June 2019].

Stackshare, 2019b. *Microsoft SQL Server.* [Online]
Available at: https://stackshare.io/microsoft-sql-server
[Accessed 11 June 2019].

Stratikopoulos, A., Kotselidis, C., Goodacre, J. & Luján, M., 2018. FastPath: Towards Wire-speed NVMe SSDs. *2018 28th International Conference on Field Programmable Logic and Applications,* pp. 170 - 177.

Thomson, A. et al., 2012. Calvin: Fast Distributed Transactions for Partitioned Database Systems. *Proceedings of the 2012 ACM SIGMOD Internatinoal Conference on Management of Data,* pp. 1 - 12.

Tongkaw, S. & Tongkaw, A., 2016. Comparison of Database Performance of MariaDB and MySQL with OLTP Workload. *2016 IEEE Conference on Open Systems,* pp. 117 - 119.

Tranasaction Processing Performance Coucil, 2018. *TPC Benchmark H,* San Francisco, USA: s.n.

Transaction Processing Performance Council, 2010. TPC Benchmark C. *TPC Benchmark C,* Volume 5.11.

Transaction Processing Performance Council, 2019. *TPC.* [Online]
Available at: http://www.tpc.org/default.asp
[Accessed 08 April 2019].

Travassos, G. & Barros, M. d. O., 2003. Contributions of In Virtuo and In Silico Experiments for the Future of Empirical Studies in Software Engineering. *2nd Workshop on Empirical Software Engineering the Future of Empirical Studies in Software Engineering,* pp. 117 - 130.

Warszawski, T. & Bailis, P., 2017. ACIDRain: Concurrency-Related Attacks on Database-Backed Web Applications. *Proceedings of the 2017 ACM International Conference on Management of Data,* pp. 5 - 20.