UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INFORMATICS INSTITUTE
BACHELOR OF COMPUTER ENGINEERING

GILSON SOUZA DOS SANTOS

Analyzing the behavior of database system components with standard benchmarks

Graduation Thesis

Advisor: Prof. Dr. Renata Galante
Coadvisor: M. Sc. Caetano Sauer

Porto Alegre
2015

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitor: Prof. Carlos Alexandre Netto
Vice-Reitor: Prof. Rui Vicente Oppermann
Pró-Reitor de Graduação: Prof. Sérgio Roberto Kieling Franco
Diretor do Instituto de Informática: Prof. Luís da Cunha Lamb
Coordenador do Curso de Engenharia de Computação: Prof. Raul Fernando Weber
Bibliotecária-Chefe do Instituto de Informática: Beatriz Regina Bastos Haro

ACKNOWLEDGMENTS

First of all I would like to thank my mother Rosângela Alves de Souza, my grandmother Ana Izabel Alves Godoy, and my uncle Rogério Barbosa Alves for all support and encouragement during all my life and especially in these last years. I would like also to thanks all my family and relatives for all the comprehension, time, and attention that they always gave me.

I would like to thank the Informatics Institute of UFRGS for providing an education of excellence to all its students. Thanks for all employees, professors and students that work hard to make it one of the best informatics intitute of Brazil. In special I would like to acknowledge Prof. Dr. Raul Weber and Profa. Dr. Taisy Weber for all the incondicional attention and support to all the students from the Informatics Institute. I would like to thank Profa. Dr. Renata Galante for having accepted to advisor this work at UFRGS.

Thanks to the Technische Universität Kaiserslautern, for offering the opportunity of this exchange program with UFRGS and for receiving all the students with open arms. I would like to thank Lucas Lersch for being an exceptional colleague and to conceive the LerschPlots idea, which has a sensible importance in this work. Special thanks also to Prof. Dr. Theo Härder for receiving students from UFRGS in the Database and Information Systems group and to M.Sc. Caetano Sauer for his immense patience and kindness on advising my work during the time I spent in Kaiserslautern.

Finally, I would like to thank all my friends and colleagues that shared with me all the difficulties and glories from the past years. I am grateful for having such good friends that I can count on hard times and laugh together on good times.

ABSTRACT

TPC benchmarks are the most adopted benchmarks by academia and industry regarding database transactions performance. These benchmarks, together with the benchmark TATP, simulate a large range of the industry applications for databases. We use these benchmarks diversity to see how database behaves under different applications.

In this work we investigate low-level characteristics of database systems when the benchmarks TPC-B, TPC-C, TPC-E, TPC-H, and TATP are executed, in order to understand how different applications properties impact the internal behavior of the system. The goal of such investigation is to understand typical application patterns and how they affect individual system components. This helps developers and administrators of database systems to improve performance and find optimal configuration parameters.

We use in this work the storage manager Shore-MT to collect statistics of some internal components and detailed patterns of access to pages of the database. When we investigated the results collected, we saw how changes in these characteristics can have different impact in low-level components, and the application characteristics that can enhance or weaken this impact. We also identify which are the benchmarks that have more impact in each component and the benchmarks characteristics that affect these results.

Keywords: Database. Database benchmarks. Page access. Buffer. Locking. Transaction. Shore-MT.

RESUMO

Os benchmarks TPC são os mais adotados pela academia e indústria em relação a desempenho de transações de banco de dados. Esses benchmarks, juntamente com o TATP, simulam uma grande variedade de aplicações industriais para sistemas de banco de dados. Essa diversidade é usada para entender como o banco de dados se comporta sob diferentes aplicações.

Nesse trabalho são investigadas características de baixo-nível do sistema de banco de dados quando os benchmarks TPC-B, TPC-C, TPC-E, TPC-H, e TATP são executados, a fim de entender como propriedades de diferentes aplicações impactam no comportamento interno do sistema. O objetivo de tal investigação é entender padrões típicos de aplicações e como eles afetam componentes individuais do sistema. Tal investigação ajuda desenvolvedores e administradores de sistemas de banco de dados a melhorarem performance e encontrarem parâmetros de configurações ideais.

Nesse trabalho é usado o gerenciador de armazenamento Shore-MT para coletar estatísticas de componentes internos e padrões detalhados de acesso a paginas do banco de dados. Quando os resultados coletados são investigados, é verificado que mudanças nessas características podem ter impactos diferentes nos componentes de baixo-nível, e as características das aplicações que podem aumentar ou diminuir esse impacto. Também são identificados quais são os benchmarks que tem maior impacto em cada componente e as características deles que afetam esses resultados.

Palavras-chave: Banco de dados. Benchmarks de banco de dados. Acesso de paginas. Buffer. Locking. Transações. Shore-MT.

LIST OF FIGURES

FIGURE 2.1 – BENCHMARKS AND THE APPLICATIONS SIMULATED BY THEM 14
FIGURE 2.1 – DIAGRAM FOR TPC-B DATABASE 15
FIGURE 2.2 – DIAGRAM FOR TPC-C DATABASE 18
FIGURE 2.3 – LIST OF SYLLABLES TO GENERATE C_LAST 27
FIGURE 2.4 – BUSINESS OPERATIONS VS. BUSINESS ANALYSIS 28
FIGURE 2.5 – DIAGRAM FOR TPC-H DATABASE 29
FIGURE 2.5 – TPC-E BUSINESS MODEL 34
FIGURA 2.7 – TATP DATABASE DIAGRAM 49
FIGURE 2.8 – LEVEL OF ABSTRACTION IN A DBMS 55
FIGURE 2.9 – COMPONENTS OF A STORAGE MANAGER 56
FIGURE 4.1 – TPC-B: REQUESTS TO PAGES WITHOUT SKEW 69
FIGURE 4.2 – TPC-B: REQUESTS TO PAGES WITH SKEW 69
FIGURE 4.3 – TPC-B: READ-WRITE ACCESSES WITHOUT SKEW 70
FIGURE 4.4 – TPC-B: READ-WRITE ACCESSES WITH SKEW 71
FIGURE 4.5 – TPC-B: HISTOGRAM OF PAGE ACCESS WITHOUT SKEW 72
FIGURE 4.6 – TPC-B: HISTOGRAM OF PAGE ACCESS WITH SKEW 72
FIGURE 4.7 – TPC-B: HISTOGRAM OF UPDATE PAGE ACCESS WITHOUT SKEW 73
FIGURE 4.8 – TPC-B: HISTOGRAM OF UPDATE PAGE ACCESS WITH SKEW 74
FIGURE 4.9 – TPC-C: REQUESTS TO PAGES WITHOUT SKEW 75
FIGURE 4.10 – TPC-C: REQUESTS TO PAGES WITH SKEW 75
FIGURE 4.11 – TPC-C: READ-WRITE ACCESSES WITHOUT SKEW 76
FIGURE 4.12 – TPC-C: READ-WRITE ACCESSES WITH SKEW 76
FIGURE 4.13 – TPC-C: HISTOGRAM OF PAGE ACCESS WITHOUT SKEW 77
FIGURE 4.14 – TPC-C: HISTOGRAM OF PAGE ACCESS WITH SKEW 78
FIGURE 4.15 – TATP: REQUESTS TO PAGES WITHOUT SKEW 79
FIGURE 4.16 – TATP: REQUESTS TO PAGES WITH SKEW 79
FIGURE 4.17 – TATP: HISTOGRAM OF PAGE ACCESS WITHOUT SKEW 80
FIGURE 4.18 – TATP: HISTOGRAM OF PAGE ACCESS WITH SKEW 80
FIGURE 4.19 – TPC-E: REQUESTS TO PAGES 81
FIGURE 4.20 – TPC-E: HISTOGRAM OF PAGE ACCESS 82
FIGURE 4.21 – TPC-H: REQUESTS TO PAGES 83
FIGURE 4.22 – TPC-H: HISTOGRAM OF PAGE ACCESS 84
FIGURE 4.23 – BOXPLOTS OF ALL BENCHMARKS 86

LIST OF TABLES

TABLE 2.2 – DESCRIPTION OF TPC-B BRANCH TABLE 15
TABLE 2.3 – DESCRIPTION OF TPC-B ACCOUNT TABLE 16
TABLE 2.4 – DESCRIPTION OF TPC-B TELLER TABLE 16
TABLE 2.5 – DESCRIPTION OF TPC-B HISTORY TABLE 16
TABLE 2.6 – SCALING RULES FOR TPC-B 16
TABLE 2.7 – DESCRIPTION OF TPC-C WAREHOUSE TABLE 19
TABLE 2.8 – DESCRIPTION OF TPC-C DISTRICT TABLE 19
TABLE 2.9 – DESCRIPTION OF TPC-C CUSTOMER TABLE 20
TABLE2.10 – DESCRIPTION OF TPC-C HISTORY TABLE 21
TABLE 2.11 – DESCRIPTION OF TPC-C ORDER TABLE 21
TABLE 2.12 – DESCRIPTION OF TPC-C ORDER-LINE TABLE 22
TABLE 2.13 – DESCRIPTION OF TPC-C NEW-ORDER TABLE 22
TABLE 2.14 – DESCRIPTION OF TPC-C ITEM TABLE 22
TABLE 2.15 – DESCRIPTION OF TPC-C STOCK TABLE 23
TABLE 2.16 – SCALING RULES FOR TPC-C 24
TABLE 2.17 – DESCRIPTION OF TPC-H PART TABLE 30
TABLE 2.18 – DESCRIPTION OF TPC-H SUPPLIER TABLE 30
TABLE 2.19 – DESCRIPTION OF TPC-H PARTSUPP TABLE 30
TABLE 2.20 – DESCRIPTION OF TPC-H CUSTOMER TABLE 31
TABLE 2.21 – DESCRIPTION OF TPC-H ORDERS TABLE 31
TABLE 2.22 – DESCRIPTION OF TPC-H LINEITEM TABLE 32
TABLE 2.23 – DESCRIPTION OF TPC-H NATION TABLE 32
TABLE 2.24 – DESCRIPTION OF TPC-H REGION TABLE 32
TABLE 2.25 – SCALING RULES FOR TPC-H 33
TABLE 2.26 –TPC-E CUSTOMER TABLES 35
TABLE 2.27 – TPC-E BROKER TABLES 36
TABLE 2.28 – DESCRIPTION OF TPC-E MARKET TABLES 37
TABLE 2.29 – TPC-E DIMENSION TABLES 38
TABLE 2.30 – DESCRIPTION OF VARIABLES USED IN TABLE 2.30 39
TABLE 2.31 – SCALING RULES FOR TPC-E 39
TABLE 2.32 – DESCRIPTION OF TPC-H REGION TABLE 40
TABLE 2.33 – TPC-E BROKER-VOLUME TRANSACTION INPUTS. 41
TABLE 2.34 – TPC-E CUSTOMER-POSITION TRANSACTION INPUTS. 42
TABLE 2.35 – TPC-E MARKET-FEED TRANSACTION INPUTS 42
TABLE 2.36 – TPC-E MARKET-WATCH TRANSACTION INPUTS 43
TABLE 2.37 – TPC-E SECURITY-DETAIL TRANSACTION INPUTS 44
TABLE 2.38 – TPC-E TRADE-LOOKUP TRANSACTION INPUTS 44
TABLE 2.39 – TPC-E TRADE-LOOKUP TRANSACTION INPUTS 45
TABLE 2.40 – TPC-E TRADE-RESULT TRANSACTION INPUTS 46
TABLE 2.41 – TPC-E TRADE-STATUS TRANSACTION INPUTS 46
TABLE 2.42 – TPC-E TRADE-UPDATE TRANSACTION INPUTS 47
TABLE 2.43 – TPC-E DATA-MAINTENANCE TRANSACTION INPUTS 48
TABLE 2.44 – TPC-E TRADE-CLEANUP TRANSACTION INPUTS 49
TABLE 2.45 – RELATIONSHIP AND DESCRIPTION OF MAIN TATP TABLE COLUMNS 50
TABLE 2.46 – SCALING RULES FOR TATP 51
TABLE 2.47 – PROBABILITY FOR TATP TRANSACTIONS SUCCEED 52
TABLE 2.48 – PROBABILITY FOR TATP READ TRANSACTIONS EXECUTE 52
TABLE 2.49 – PROBABILITY FOR TATP WRITE TRANSACTIONS EXECUTE 52
TABLE 2.50 – RULES TO GENERATE TRANSACTION INPUTS IN TATP BENCHMARK 53
TABLE 2.51 – BENCHMARK COMPARISON 54
TABLE 4.1 – TRANSACTION COUNTERS 87
TABLE 4.2 – COUNTERS OF BUFFER MANAGER 89
TABLE 4.3 – COUNTERS OF LOCK MANAGER 90

LIST OF ABBREVIATIONS AND ACRONYMS

DBMS	Database Management System
OLAP	Online Analytical Processing
OLTP	Online Transaction Processing
TPC	Transaction Processing Performance Council

CONTENTS

1 INTRODUCTION 11

2 THE BENCHMARKS AND THE DBMS 14

2.1 BENCHMARKS AND TOOLS OF THIS WORK 14
2.2 TPC-B 15
2.1.1 Entities, Relationship and Characteristics 15
2.2.2 Scaling Rules 16
2.2.3 Transactions 17
2.2.4 Transaction Input Generation 17
2.3 TPC-C 17
2.3.1 Entities, Relationship and Characteristics 18
2.3.2 Scaling Rules 23
2.3.3 Transactions 24
2.3.4 Transactions Input Generation 26
2.4 TPC-H 27
2.4.1 Entities, Relationship and Characteristics 28
2.4.2 Scaling Rules 33
2.4.3 Transactions 33
2.5 TPC-E 33
2.5.1 Entities, Relationship and Characteristics 34
2.5.2 Scaling Rules 38
2.5.3 Transactions 39
2.5.4 Transactions Input Generation 41
2.6 TATP 49
2.6.1 Entities, Relationship and Characteristics 49
2.6.2 Scaling Rules 51
2.6.3 Transactions 51
2.6.4 Transactions Input Generation 53
2.7 COMPARISON OF THE BENCHMARKS 54
2.8 SHORE-MT, THE STORAGE MANAGER 55
2.8.1 Log Manager 57
2.8.2 Buffer Pool Manager 57
2.8.3 Lock Manager 57

3 GATHERING STATISTICS 60

3.1 METHODOLOGY OVERVIEW 60
3.1 THE LOG 60
3.2 THE LOGINSPECT TOOL 62
3.3 PAGE ACCESS CHARACTERISTICS 62
3.4 COUNTERS 63
3.5 TRANSACTION COUNTERS 63
3.6 BUFFER MANAGER COUNTERS 64
3.7 LOCKING COUNTERS 65

4 EXPERIMENTS 67

4.1 EXPERIMENTAL SETUP 67
4.2 PAGE ACCESS ANALYSIS 68
4.2.1 TPC-B 68
4.2.2 TPC-C 74
4.2.3 TATP 78

4.2.4 TPC-E *81*
4.2.5 TPC-H *82*
4.2.6 Comparing skew among benchmarks *84*
4.3 COUNTERS OF TRANSACTION EVENTS 86
4.4 COUNTERS OF BUFFER MANAGER EVENTS 87
4.5 COUNTERS OF LOCKING EVENTS 89
5 CONCLUSION 92

1 INTRODUCTION

Benchmarks are tools used to verify performance characteristics of computer hardware and software. In the database context, there are standard benchmarks used to measure and compare the performance of those systems. These benchmarks simulate industry applications and give a perspective of how these applications would perform in a specific hardware and software setup. By understanding the patterns of the application, and how it affects individual components of the system, developers and maintainers of database systems could improve database configurations and software.

Our study analyzes the behavior of internal database system components – such as the buffer, log, and lock managers – when executing different benchmarks. Small variations in workload characteristics sometimes drastically affect the behavior and performance of internal components in a way that is not easy to predict. This can lead to misconfigured system parameters and/or wrong implementation choices, which deliver sub-optimal performance. Our goal is to understand the effect of such variations in order to assist developers and administrators in making the correct decisions.

To achieve such goal, we enhance the DBMS code with analysis tools to collect internal statistics within each internal component and perform a wide range of benchmarks under varying configurations. These statistics collected allow us to identify bottlenecks in the system, which occur due to certain workload characteristics. We also perform a qualitative analysis to determine which workload characteristics affect which components and in which way.

Our experiment workload consists of a subset of the TPC suite of benchmarks for business applications (TPC-B, TPC-C, TPC-E, and TPC-H) as well as the TATP benchmark for telecommunication applications. These are implemented in the Shore-MT research prototype, which is available as open-source.

When we executed the benchmarks in the database system and varied some of their high-level characteristics, we could verify that the DBMS components behavior does not proportionally follow these different characteristics. For example, when we increase the skew of a workload in order to force it to increase the access to a restrict set of data, the impact suffered in components, such as buffer manager, depends more on the benchmarks properties than on the restriction that we forced. By analyzing these aspects in this study, we fill this important gap between this behavior of higher-level applications and that of lower-level internal components of the system.

The remainder of this work is organized as follows: Chapter 2 describes the five benchmarks studied, as well as the storage manager used and the components to be studied. Chapter 3 presents the methodology used to compare database behavior according to each benchmark and what was implemented to gather statistics. Chapter 4 explains the methodology used to run the benchmarks, describes the hardware environment, and presents the results of this study. Finally, conclusions and remarks are outlined in Chapter 5.

2 THE BENCHMARKS AND THE DBMS

This chapter aims to review and define important concepts required for the clear understanding of the work. At first we are going to give a brief overview of the benchmarks and tools that we used. Later we describe the main characteristics of each benchmark and a brief comparison among them. Finally we present the storage manager and concepts of the analyzed internal components.

2.1 Benchmarks and tools of this work

The chosen benchmarks for this study are TPC-B (TPC Benchmark B Specification, 1994), TPC-C (TPC Benchmark C Specification, 2010), TPC-E (TPC Benchmark E Specification, 2014), TPC-H (TPC Benchmark H Specification, 2014) and TATP (TATP Benchmark description, 2009). These benchmarks simulate, respectively, applications for banking, warehouse, brokerage house, business analysis of a wholesale supplier and mobile network operations, and give us different perspectives of design and stress of database systems. TPC-B is a more update-intensive benchmark with only one OLTP transaction, TPC-C is a more complex application than TPC-B and has 7 OLTP transactions, TPC-H contains only OLAP queries, TPC-E is a mix of OLAP and OLTP transactions in a much more complex application and TATP is chosen to simulate a different application from the others benchmarks. Table 2.1 present the benchmarks and the applications simulated by them.

Figure 2.1 – Benchmarks and the applications simulated by them

Benchmark	TPC-B	TPC-C	TPC-E	TATP	TPC-H
Application simulated	Banking	Wholesale supplier	Brokerage house	Mobile network operators	Wholesale supplier
Type of transactions	OLTP	OLTP	OLTP and OLAP	OLTP	OLAP

The DBMS chosen to benchmark is Shore-MT. It is a storage manager developed as a scientific project, which is focused on enhancing performance for multi-thread processors. All benchmarks are already implemented for Shore-MT in a suite of database benchmarks called Shore-Kits.

Below a description of each benchmark is given, as well as the description of Shore-MT and the components that we study. These descriptions give us the necessary knowledge to compare and understand the behavior of the system components when we perform these different benchmarks on it. Measurements of performance, among other characteristics of

each benchmark, will not be described since they are irrelevant for the main propose of this study.

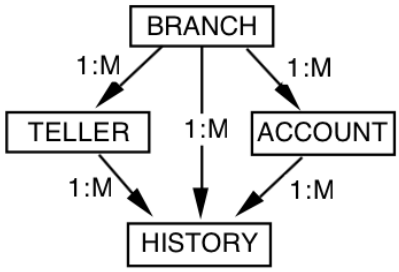
2.2 TPC-B

TPC-B (TPC Benchmark B Specification, 1994) is a benchmark that simulates a banking application. It is a good database engine stress test, with significant I/0, moderate system application execution time, and good measure for transaction integrity (since it is possible to see the a record of each executed transaction in database). As previous studies have already discussed, it has a design similar to TPC-A, which makes the study of the benchmark TPC-A unnecessary, since it shows similar results (Tözün P, 2013).

2.1.1 Entities, Relationship and Characteristics

This benchmark is composed of four separated and individual tables: Account, Branch, Teller and History, where each branch has the same number of tellers and accounts, each teller and account are related to 1 Branch, and table History keeps records of all operations and the identification of the Teller, Branch and Account involved on this. The relationships among these tables are represented in Figure 2.1.

Figure 2.1 – Diagram for TPC-B database



Source: TPC Benchmark B Specification (1994, p. 13).

The database tables have the columns described in Tables 2.2 – 2.5:

Table 2.2 – Description of TPC-B Branch Table

Column Name	Column Description
Branch_ID	Must uniquely identify the row across the range of branches.
Branch_Balance	Must represent at least 10 significant decimal digits and the sign.

Table 2.3 – Description of TPC-B Account Table

Column Name	**Column Description**
Account_ID	Must uniquely identify the row across the range of accounts and must be unique across the entire database.
Branch_ID	Branch where account is held.
Account_Balance	Must represent at least 10 significant decimal digits and the sign.

Table 2.4 – Description of TPC-B Teller Table

Column Name	**Column Description**
Teller_ID	Identify the row across the range of tellers and must be unique.
Branch_ID	Branch where the teller is held.
Teller_Balance	Must represent at least 10 significant decimal digits and the sign.

Table 2.5 – Description of TPC-B History Table

Column Name	**Column Description**
Account_ID	Account updated by transaction.
Teller_ID	Teller involved in transaction.
Branch_ID	Branch associated with Teller.
Amount	Amount specified by transaction. Must represent at least 10 significant decimal digits and the sign.
Time_Stamp	A date and time taken between BEGIN_TRANSACTION and COMMIT_TRANSACTION. Must represent date as YY:MM:DD and time with a resolution of at least HH:MM:SS:r3.3.

2.2.2 Scaling Rules

Table 2.6 shows the minimum amount of rows for each database table. The ratio between these values must be maintained (i.e. when any value is increased, the others must be increased proportionally). For running the tests, we use the Branch table as the base unit of scaling.

Table 2.6 – Scaling rules for TPC-B

Account	100000
Teller	10
Branch	1

The table History is not shown in the scaling rules because it grows while the workload is executing. Its size depends on how many transactions per second the system executes and how long the test lasts.

2.2.3 Transactions

The TPC-B workload consists of one debit-credit transaction. This transaction receives four parameters: Branch_ID, Teller_ID, Account_ID, and Amount. The transaction is considered a home transaction when the account belongs to the same branch of the teller which initiated the transaction. Otherwise it is considered a remote transaction.

Remote transactions may be processed entirely in a single node or be distributed in two separated nodes. It occurs if the account branch and the teller branch exist on different nodes. The node containing the teller is referred to as the native node, and the node containing the account branch is referred to as the foreign node.

2.2.4 Transaction Input Generation

The rules that must be followed in order to generate the four parameters of the transaction are as follows:

Teller_ID Generation: The Teller must be chosen at random from some range of Teller_ID values, without regard to transactions being processed in parallel.

Branch_ID Generation: As a Teller resides in a known Branch, the Branch_ID selected is the one associated with the Teller generated.

Account_ID Generation: In order to generate the Account_ID, it is necessary to follow two steps: First a random number X is generated within [0,1]. After that, if X is less than 0.85 or there is only one branch, a random Account_ID is selected over all accounts in the same Branch_ID. However, if X is greater than 0.85 and there are more than 2 branches, a random Account_ID is generated over all accounts there are not held in the branch selected.

This algorithm assures that, if there is more than one branch, an average of 15% of all transactions will be remote.

Delta Amount: This parameter is a random value within [-999999, +999999], selected independently for each transaction.

2.3 TPC-C

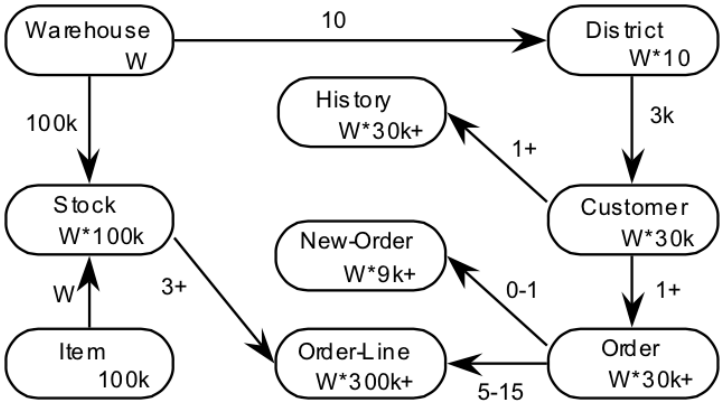
For the last two decades, TPC-C (TPC Benchmark C Specification, 2010) has been the OLTP benchmark mostly used by industry and academia. It models a database for a wholesale supplier with a number of geographically distributed sales districts and associated

warehouses, and consists of five simple short-running transactions with frequent updates, which follow customer orders from initial creation to final delivery and payment.

2.3.1 Entities, Relationship and Characteristics

The TPC-C benchmark is composed of nine tables. The relationship between these tables can be seen in the entity-relationship diagram shown in Figure 2.2.

Figure 2.2 – Diagram for TPC-C database



Source: TPC Benchmark C Specification (2010, p. 11).

The columns of these tables are following described. They can be implemented in any order using any physical representation. The relationship between the tables are defined in the Relations field as PK for a Primary Key, PK+ for a composed Primary Key, FK () for a Foreign Key and FK+ () for a composed foreign key. As the purpose of this section is to show simply the relationship between columns, their exact specification is not provided. We refer to the official documentation for such details (TPC Benchmark C Specification, 2010).

Table 2.7 – Description of TPC-C Warehouse Table

Column Name	**Key Constraints**
W_ID	PK
W_NAME	
W_STREET_1	
1_STREET_2	
W_CITY	
W_STATE	
W_ZIP	
W_TAX	
W_YTD	

Table 2.8 – Description of TPC-C District Table

Column Name	**Key Constraints**
D_W_ID	PK+ FK (W_ID)
D_ID	PK+
D_NAME	
D_STREET_1	
D_STREET_2	
D_CITY	
D_STATE	
D_ZIP	
D_TAX	
D_YTD	
D_NEXT_O_ID	

Table 2.9 – Description of TPC-C Customer Table

Column Name	**Key Constraints**
C_ID	PK+
C_W_ID	PK+ FK+ (D_W_ID, D_ID)
C_D_ID	PK+ FK+ (D_W_ID, D_ID)
C_FIRST	
C_MIDDLE	
C_LAST	
C_STREET_1	
C_STREET_2	
C_CITY	
C_STATE	
C_ZIP	
C_PHONE	
C_SINCE	
C_CREDIT	
C_CREDIT_LIM	
C_DISCOUNT	
C_BALANCE	
C_YTD_PAYMENT	
C_PAYMENT_CNT	
C_DELIVERY_CNT	
C_DATA	

Table2.10 – Description of TPC-C History Table

Column Name	**Key Constraints**
H_C_W_ID	FK+ (C_W_ID, C_D_ID, C_ID)
H_C_D_ID	FK+ (C_W_ID, C_D_ID, C_ID)
H_C_ID	FK+ (C_W_ID, C_D_ID, C_ID)
H_W_ID	FK+ (D_W_ID, D_ID)
H_D_ID	FK+ (D_W_ID, D_ID)
H_DATE	
H_AMOUNT	
H_DATA	

** The History table has no primary key, since for this benchmark there is no need to uniquely identify a row within this table.*

Table 2.11 – Description of TPC-C Order Table

Column Name	**Key Constraints**
O_ID	PK+
O_W_ID	PK+ FK+ (C_W_ID, C_D_ID, C_ID)
O_D_ID	PK+ FK+ (C_W_ID, C_D_ID, C_ID)
O_C_ID	FK+ (C_W_ID, C_D_ID, C_ID)
O_ENTRY_D	
O_CARRIER_ID	
O_OL_CNT	
O_ALL_LOCAL	

Table 2.12 – Description of TPC-C Order-Line Table

Column Name	**Key Constraints**
OL_W_ID	PK+ FK+ (O_W_ID, O_D_ID, O_ID)
OL_D_ID	PK+ FK+ (O_W_ID, O_D_ID, O_ID)
OL_O_ID	PK+ FK+ (O_W_ID, O_D_ID, O_ID)
OL_NUMBER	PK+
OL_SUPPLY_W_ID	FK+ (S_W_ID, S_I_ID)
OL_I_ID	FK+ (S_W_ID, S_I_ID)
OL_DELIVERY_D	
OL_QUANTITY	
OL_AMOUNT	
OL_DIST_INFO	

Table 2.13 – Description of TPC-C New-Order Table

Column Name	**Key Constraints**
NO_O_ID	PK+ FK+ (O_W_ID, O_D_ID, O_ID)
NO_D_ID	PK+ FK+ (O_W_ID, O_D_ID, O_ID)
NO_W_ID	PK+ FK+ (O_W_ID, O_D_ID, O_ID)

Table 2.14 – Description of TPC-C Item Table

Column Name	**Key Constraints**
I_ID	PK
I_IM_ID	
I_NAME	
I_PRICE	
I_DATA	

Table 2.15 – Description of TPC-C Stock Table

Column Name	**Key Constraints**
S_W_ID	PK+ FK (W_ID)
S_I_ID	PK+ FK (I-ID)
S_QUANTITY	
S_DIST_01	
S_DIST_02	
S_DIST_03	
S_DIST_04	
S_DIST_05	
S_DIST_06	
S_DIST_07	
S_DIST_08	
S_DIST_09	
S_DIST_10	
S_YTD	
S_ORDER_CNT	
S_REMOTE_CNT	
S_DATA	

2.3.2 Scaling Rules

For this benchmark, the Warehouse is the unit of scaling and, except for the Item table that has a fixed cardinality, the ratio between these tables should be maintained following the rules described in Table 2.15.

Table 2.16 – Scaling rules for TPC-C

WAREHOUSE	1
DISTRICT	10
CUSTOMER	30k
HISTORY	30k
ORDER	30k
NEW-ORDER	9k
ORDER-LINE	300k
STOCK	100k
ITEM	100k

2.3.3 Transactions

The TPC-C benchmark consists of five transactions, where two of them are ready-only transactions (Order-Status and Stock-Level) and the other three are read-write transactions (New-Order, Payment and Delivery). These transactions emulate operations that would be executed in a wholesale supplier system.

Below, a brief description of each transaction is given. In these descriptions, we use light-weight, mid-weight and heavy as a subjective classification for transactions in terms of contention and number of operations executed by them.

2.3.3.1 The New-Order Transaction

The New-Order transaction is the backbone of the workload and consists of creating a complete new order from a customer. It represents a mid-weight transaction and it is executed with a high frequency.

A new order receives as input a warehouse number (W_ID), a district number (D_W_ID, D_ID), a customer number (C_W_ID, C_D_ID, C_ID) and a variable set of items with cardinality between 5 and 15. The number of items generated for each new order is randomly selected within [5..15]. The inputs necessary for each item of this list are: The item identification (OL_I_ID), the supplying warehouse (OL_SUPPLY_W_ID) and the quantity (OL_QUANTITY). The order is considered home if it is supplied by the home warehouse (i.e. when OL_SUPPLY_W_ID equals O_W_ID) otherwise it is considered as remote. In this transaction, the supplying warehouse (OL_SUPPLY_W_ID) is selected as the home warehouse 99% of the time and as a remote warehouse 1% of the time.

Summarizing, a new order is executed in two steps: First, it creates an order header, which is stored in the Order table. In the second step, it orders a set of items and, for each item ordered, a new record is created in the Order-Line table.

2.3.3.2 The Payment Transaction

The Payment transaction executes payments of orders, affecting the customer's balance and reflecting the payment on the district and warehouse sales statistics. It represents a lightweight transaction with a high frequency of execution.

This transaction receives as input the district number (D_W_ID, D_ID), the customer number (C_W_ID, C_D_ID, C_ID) or customer last name (C_W_ID, C_D_ID, C_LAST), and payment amount (H_AMOUNT). The customer is selected 60% of the time by last name and 40% of the time by his customer number. This simulates the situation where a customer does not use his unique customer number. Furthermore, the probability of the warehouse selected being a home warehouse is 85%.

2.3.3.3 The Order-Status Transaction

The Order-Status transaction queries the status of a customer's last order and represents a mid-weight transaction and it is executed with low frequency.

It receives as input the warehouse number (W_ID), the district number (D_W_ID, D_ID), in 60% of the times the customer number (C_W_ID, C_D_ID, C_ID), and in 40% of the cases the customer last name (C_W_ID, C_D_ID, C_LAST).

2.3.3.4 The Delivery Transaction

The Delivery transaction consists of processing a batch of 10 not-yet-delivered orders. It has a low frequency of execution and it is intended to be executed in deferred mode through a queueing mechanism. The result of the deferred execution is recorded in a result file.

As input, this transaction receives 2 parameters: the warehouse number (W_ID), which is associated with the Warehouse table, and the carrier number (O_CARRIER_ID), which exist in the Order table only to simulate the existence of a carrier. However, the table Carrier does not exist.

2.3.2.5 The Stock-Level Transaction

The Stock-Level transaction verifies the number of recently sold items that have a stock level below a specified threshold. It is a heavy database transaction and it has a low frequency of execution.

As input, the transaction receives a unique warehouse and a district number (W_ID and D_ID), plus a threshold selected at random within [10..20].

2.3.4 Transactions Input Generation

The transactions described above run in terminals, which have an emulated user generating the inputs for each transaction. Terminals are associated with warehouses and, for each warehouse created, there are 10 terminals associated with it. These terminals send the warehouse identification (W_ID) associated with them to each transaction as input.

For each terminal, there is an emulated user, who generates the others inputs necessary for each transaction. The rules to be followed by the emulated user during the input generation and the NURand function, used in some of these rules, are described below.

NURand function: This function is used in the process of selecting the district number, the customer last name, and the item number inputs. NURand is designed to generate a non-uniformely distributed random number over the specified range of values [x .. y]. The following formula describes this function:

NURand(A, x, y) = (((random(0, A) | random(x, y)) + C) % (y - x + 1)) + x

Where:

- | stands for the bitwise logical OR operation
- % stands for the modulo operation
- A is a constant chosen according to the size of the range [x .. y]
- C is a run-time constant randomly chosen within [0 .. A] that can be varied without altering performance.

District number (D_ID): It is randomly selected within [1..10].

Customer number (C_ID): The non-uniform random customer number is generated using the NURand(1023,1,3000) function.

Customer last name (C_LAST): Generated by the concatenation of three syllables from the list showed on Figure 2.3:

Figure 2.3 – List of syllables to generate C_LAST

0	1	2	3	4	5	6	7	8	9
BAR	OUGHT	ABLE	PRI	PRES	ESE	ANTI	CALLY	ATION	EING

Source: TPC Benchmark C Specification (2010, p. 64).

The syllables are selected using the NURand(255,0,999) function, what turns out in possible several customers with the same last name.

Item number (OL_I_ID): Selected using NURand(8191, 1, 100000) function.

Quantity (OL_QUANTITY): Randomly selected within [1..10].

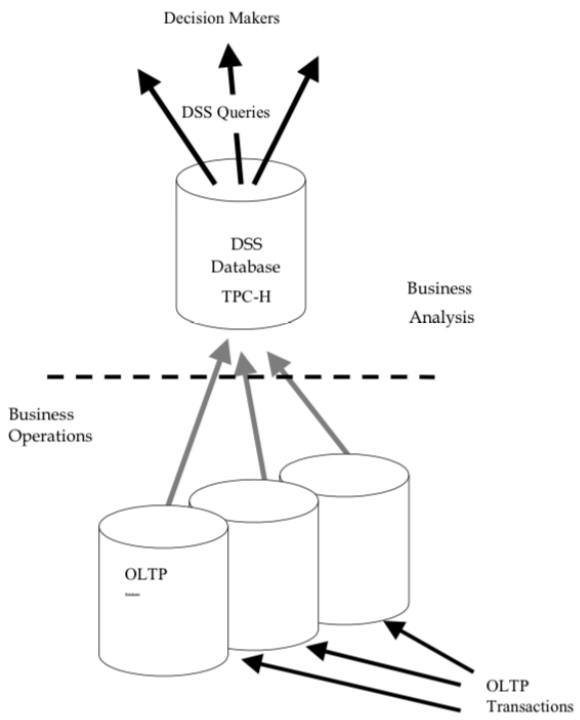
Carrier number (O_CARRIER_ID): it is randomly selected within [1..10].

2.4 TPC-H

TPC-H (TPC Benchmark H Specification, 2014) is a decision support benchmark. Different from the other benchmarks described in this document, TPC-H has only OLAP queries. It runs complex, long-running, read-only queries and its main performance measure is how many queries are executed per second (Tözün P, 2013).

Figure 2.4 shows the main difference between business analysis and business operations and help us to understand the main difference between TPC-H and the other benchmarks. While the other benchmarks aim at the business operations side of the image, using OLTP transactions and simulating daily activities of a database in production, TPC-H is on the business analysis side, i.e., the workload goal is to analyze data to support business decisions.

Figure 2.4 – Business operations vs. Business Analysis

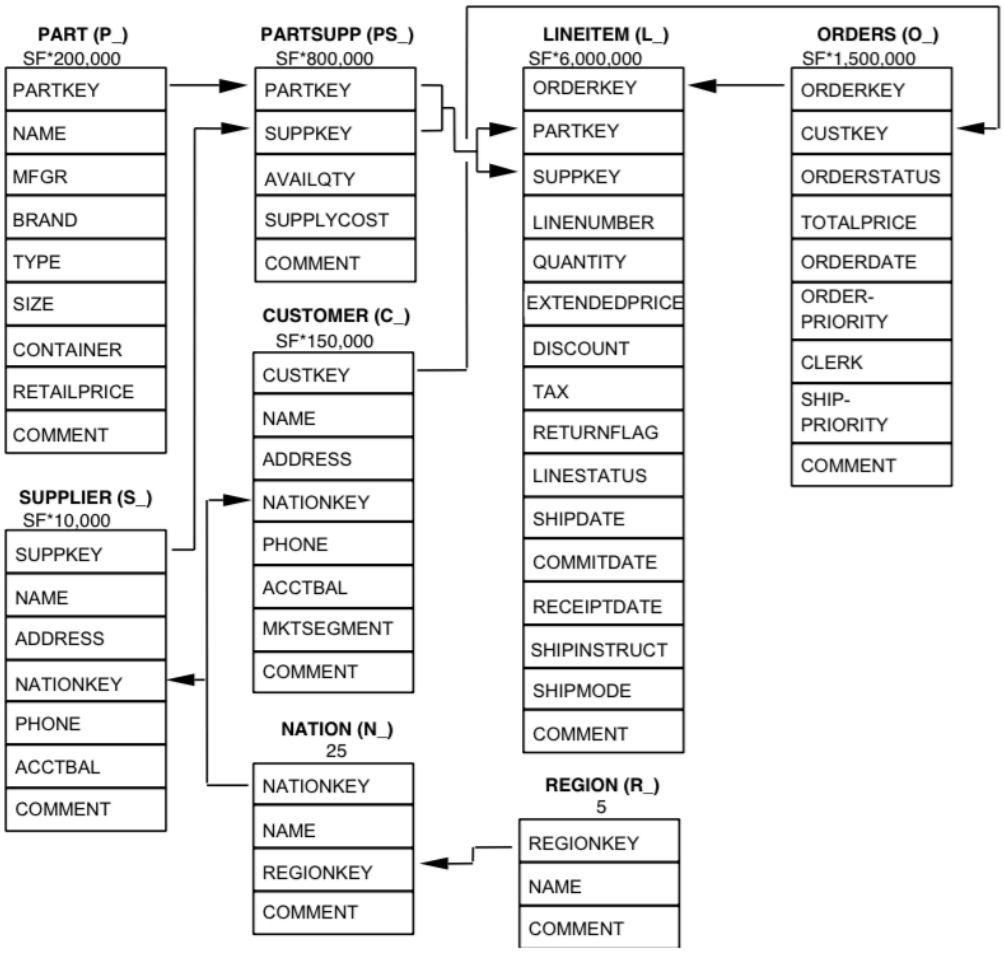


Source: TPC Benchmark H Specification (2014, p. 12).

2.4.1 Entities, Relationship and Characteristics

In order to represent a realistic context, TPC-H models a database for a wholesale supplier, which consists of eight tables. These tables and the relationships between them are described in Figure 2.5.

Figure 2.5 – Diagram for TPC-H database



Source: TPC Benchmark H Specification (2014, p. 13).

In Figure 2.5, the parentheses following each table name contain the prefix of the columns names for that table. The number or formula below the name of the table yields the number of rows of the table as a function of the scale factor (SF). The columns of and the relationships between TPC-H tables are described in Tables 2.17 - 2.24.

Table 2.17 – Description of TPC-H Part Table

Column Name	**Key Constraints**
P_PARTKEY	PK
P_NAME	
P_MFGR	
P_BRAND	
P_TYPE	
P_SIZE	
P_CONTAINER	
P_RETAILPRICE	
P_COMMENT	

Table 2.18 – Description of TPC-H Supplier Table

Column Name	**Key Constraints**
S_SUPPKEY	PK
S_NAME	
S_ADDRESS	
S_NATIONKEY	FK (N_NATIONKEY)
S_PHONE	
S_ACCTBALL	
S_COMMENT	

Table 2.19 – Description of TPC-H PartSupp Table

Column Name	**Key Constraints**
PS_PARTKEY	PK+ FK (P_PARTKEY)
PS_SUPPKEY	PK+ FK (S_SUPPKEY)
PS_AVAILQTY	
PS_SUPPLYCOST	
PS_COMMENT	

Table 2.20 – Description of TPC-H Customer Table

Column Name	**Key Constraints**
C_CUSTKEY	PK
C_NAME	
C_ADDRESS	
C_NATIONKEY	FK (N_NATIONKEY)
C_PHONE	
C_ACCTBAL	
C_MKTSEGMENT	
C_COMMENT	

Table 2.21 – Description of TPC-H Orders Table

Column Name	**Key Constraints**
O_ORDERKEY	PK
O_CUSTKEY	FK (C_CUSTKEY)
O_ORDERSTATUS	
O_TOTALPRICE	
O_ORDERDATE	
O_ORDERPRIORITY	
O_CLERK	
O_SHIPPRIORITY	
O_COMMENT	

Table 2.22 – Description of TPC-H LineItem Table

Column Name	**Key Constraints**
L_ORDERKEY	PK+ FK (O_ORDERKEY)
L_PARTKEY	FK+ (PS_PARTKEY, PS_SUPPKEY)
L_SUPPKEY	FK+ (PS_PARTKEY, PS_SUPPKEY)
L_LINENUMBER	PK+
L_QUANTITY	
L_EXTENDEDPRICE	
L_DISCOUNT	
L_TAX	
L_RETURNFLAG	
L_LINESTATUS	
L_SHIPDATE	
L_COMMITDATE	
L_RECEIPTDATE	
L_SHIPINSTRUCT	
L_SHIPMODE	
L_COMMENT	

Table 2.23 – Description of TPC-H Nation Table

Column Name	**Key Constraints**
N_NATIONKEY	PK
N_NAME	
N_REGIONKEY	FK (R_REGIONKEY)
N_COMMENT	

Table 2.24 – Description of TPC-H Region Table

Column Name	**Key Constraints**
R_REGIONKEY	PK
R_NAME	
R_COMMENT	

2.4.2 Scaling Rules

The scaling rules for the TPC-H benchmark are based on a variable called scale factor (SF), which must assume one of the following values: 1, 10, 20, 100, 300, 1000, 3000, 10000, 30000, 100000. The user of this benchmark defines the value for SF, but any number different of those mentioned is not allowed on the official published results. Table 2.25 shows how many lines must exist in each table, and the tables where the Scale Factor must be multiplied in order to find the final number of rows.

Table 2.25 – Scaling Rules for TPC-H

PART	SF*200,000
PARTSUPP	SF*800,000
SUPPLIER	SF*10,000
LINEITEM	SF*6,000,000
ORDERS	SF*1,500,000
CUSTOMER	SF*150,000
NATION	25
REGION	5

2.4.3 Transactions

The TPC-H benchmark consists of twenty-two complex queries, which examine a large volume of data to simulate the business analysis of a wholesale supplier application, and two refresh functions, which are executed always as a pair and in sequence. These simulate a sequence of batched data modifications executing against the database to bring it up to date with its operational data source.

Since the number of transactions is quite extensive in TPC-H, and our focus on this work is on OLTP workloads, we omit the description of each transaction. Instead, we refer to the official documentation (TPC Benchmark H Specification, 2014).

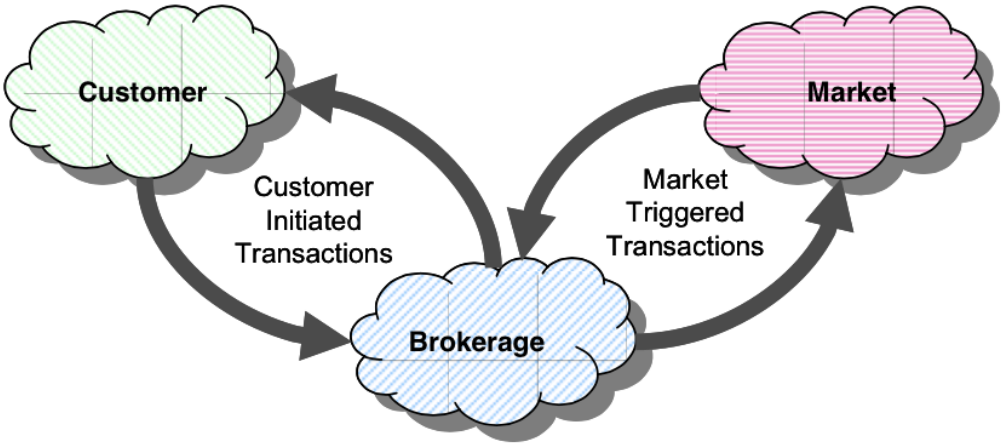
2.5 TPC-E

The TPC-E (TPC Benchmark E Specification, 2014) benchmark is the latest benchmark introduced by the Transaction Processing Performance Council (TPC). It was designed to represent current database applications with more accuracy than TPC-C. It is a hybrid of a typical OLTP and an OLAP benchmark, with a set of twelve transactions mixing short update transactions with more complex, long-running, read-only transactions (Tözün P, 2013).

Although the first version of TPC-E was released in 2007, its adoption by industry and academia is still slow. It is possible to see in TPC-E results publications that, until these days, there are only publications using Microsoft SQL Server as a database product, proving the slow adoption of this benchmark by the database software industry.

The TPC-E business model is of a brokerage company, which manages customers' accounts and executes trade orders that interact with the financial market. Figure 2.6 presents this business model.

Figure 2.5 – TPC-E business model



Source: TPC Benchmark E Specification (2014, p. 43).

2.5.1 Entities, Relationship and Characteristics

The TPC-E schema is composed of 33 tables, with 9 of them having a fixed size, 16 scaling based on the number of customers but not growing or shrinking (i.e., updates only), and the remaining 8 growing during workload execution (TPC Benchmark E Specification, 2014, p. 68). The tables can be divided in four categories: Customer, Market, Brokerage and Dimension. The schema is given in Tables 2.26 - 2.29, each describing a category.

Table 2.26 –TPC-E Customer Tables

Table (table prefix)	Column Name	Key Constraints	Column Description
ACCOUNT_PERMISSION (AP_)	AP_CA_ID	PK+ FK (CA_)	Customer account identifier.
	AP_TAX_ID	PK+	Tax account identifier of the person with access to the customer account.
CUSTOMER (C_)	C_ID	PK	Customer identifier.
	C_ST_ID	FK (ST_)	Customer status type identifier.
	C_AD_ID	FK (AD_)	Address identifier of the customer's address.
CUSTOMER_ACCOUNT (CA_)	CA_ID	PK	Customer account identifier.
	CA_B_ID	FK (B_)	Identifier of the broker who manager this account.
	CA_C_ID	FK (C_)	Identifier of the customer who owns this account.
CUSTOMER_TAXRATE (CX_)	CX_TX_ID	PK+ FK (TX_)	Tax rate identifier.
	CX_C_ID	PK+ FK(C_)	Identifier of the customer who must pay this tax rate.
HOLDING (H_)	H_T_ID	PK FK (T_)	Identifier of the trade.
	H_CA_ID	FK+(HS_)	Customer account identifier.
	H_S_SYMB	FK+(HS_)	Symbol for the security held.
HOLDING_HISTORY (HH_)	HH_H_T_ID	PK+ FK (T_)	Identifier of the trade that create the holding row.
	HH_T_ID	PK+ FK (T_)	Identifier of the current trade.
HOLDING_SUMMARY (HS_)	HS_CA_ID	PK+ FK (CA_)	Customer account identifier.
	HS_S_SYMB	PK+ FK (S_)	Symbol for the security held.
WATCH_ITEM (WI_)	WI_WL_ID	PK+ FK (WL_)	Watch list identifier.
	WI_S_SIMB	PK+ FK (S_)	Symbol for the security held.

WATCH_LIST (WL_)	WL_ID	PK	Watch list identifier.
	WL_C_ID	FK (C_)	Identifier of the customer who created this watch list.

Table 2.27 – TPC-E Broker Tables

Table (table prefix)	Column Name	Key Constraints	Column Description
BROKER (B_	B_ID	PK	Broker identifier.
	B_ST_ID	FK (ST_)	Broker status type identifier. Identifies if this broke is active or not.
CASH_TRANSACTION (CT_)	CT_T_ID	PK FK (T_)	Trade identifier.
CHARGE (CH_)	CH_TT_ID	PK+ FK (TT_)	Trade type identifier.
	CH_C_TIER	PK+	Customer's tier.
COMMISSION_RATE (CR_)	CR_C_TIER	PK+	Customer's tier. Valid values 1,2 or 3.
	CR_TT_ID	PK+ FK (TT_)	Identifies the type of trade.
	CR_EX_ID	PK+ FK (EX_)	Identifier the exchange the trade is against.
	CR_FROM_QTY	PK+	Lower bound of quantity being traded to match this commission rate.
SETTLEMENT (SE_)	SE_T_ID	PK FK (T_)	Trade identifier.
TRADE (T_)	T_ID	PK	Trade identifier.
	T_ST_ID	FK (ST_)	Identify the status of this trade.
	T_TT_ID	FK (TT_)	Identifier of the type of this trade.
	T_S_SYMB	FK (S_)	Security symbol of the security that was traded.
	T_CA_ID	FK (CA_)	Customer account identifier.
TRADE_HISTORY (TH_)	TH_T_ID	PK+ FK (T_)	Trade identifier.
	TH_ST_ID	PK+ FK (ST_)	Status type identifier.
TRADE_REQUEST	TR_T_ID	PK	Trade request identifier.

(TR_)		FK (T_)	
	TR_TT_ID	FK (TT_)	Trade request type identifier.
	TR_S_SYMB	FK (S_)	Security symbol of the security the customer wants to trade.
	TR_B_ID	FK (B_)	Identifies the broker handling the trade.
TRADE_TYPE (TT_)	TT_ID	PK	Trade type identifier.

Table 2.28 – Description of TPC-E Market Tables

Table (table prefix)	Column Name	Key Constraints	Column Description
COMPANY (CO_)	CO_ID	PK	Company identifier.
	CO_ST_ID	FK (ST_)	Company status type identifier.
	CO_IN_ID	FK (IN_)	Industry identifier.
	CO_AD_ID	FK (AD_)	Address identifier.
COMPANY_COMPETITOR (CP_)	CP_CO_ID	PK+ FK (CO_)	Company identifier.
	CP_COMP_CO_ID	PK+ FK (CO_)	Identifier of the competitor company.
	CP_IN_ID	PK+ FK (IN_)	Industry identifier.
DAILY_MARKET (DM_)	DM_DATE	PK+	Date of last completed training day.
	DM_S_SYMB	PK+ FK (S_)	Security symbol of this security.
EXCHANGE (EX_)	EX_ID	PK	Exchange identifier.
	EX_AD_ID	FK (AD_)	Mailing address of exchange.
FINANCIAL (FI_)	FI_CO_ID	PK+ FK (CO_)	Company identifier
	FI_YEAR	PK+	Year of the quarter end.
	FI_QTR	PK+	Quarter number that the financial information is valid for.
INDUSTRY (IN_)	IN_ID	PK	Industry identifier.
	IN_SC_ID	FK (SC_)	Identifier of the sector the industry is in.
LAST_TRADE (LT_)	LT_S_SYMB	PK	Security symbol.

		FK (S_)	
NEWS_ITEM (NI_)	NI_ID	PK	News item identifier.
NEWS_XREF (NX_)	NX_NI_ID	PK+ FK (NI_)	News item identifier.
	NX_CO_ID	PK+ FK (CO_)	Identifier of the company mentioned in the news item.
SECTOR (SC_)	SC_ID	PK	Sector identifier.
SECURITY (S_)	S_SYMB	PK	Security symbol.
	S_ST_ID	FK (ST_)	Security status type identifier.
	S_EX_ID	FK (EX_)	Identifier of the exchange the security is traded on.
	S_CO_ID	FK (CO_)	Identifier of the company this secured is issued by.

Table 2.29 – TPC-E Dimension Tables

Table (table prefix)	**Column Name**	**Key Constraints**	**Column description**
ADDRESS (AD_)	AD_ID	PK	Address identifier.
	AD_ZC_CODE	FK (ZC_)	Zip or postal code.
STATUS_TYPE (ST_)	ST_ID	PK	Status type identifier.
TAXRATE (TX_)	TX_ID	PK	Tax rate identifier.
ZIP_CODE (ZC_)	ZC_CODE	PK	Postal code.

2.5.2 Scaling Rules

As already mentioned, there are 16 scale tables which are based on the number of customers. Table 2.31 shows, for an amount of 5,000 customers, the relation between rows in the tables and customers in the system. Table 2.30 describes the variables used in formulas of Table 2.31.

Table 2.30 – Description of variables used in table 2.30

Variable	Description
Customers	Number of rows in CUSTOMER table.
Accounts	Number of rows in CUSTOMER_ACCOUNT table.
Companies	Number of rows in COMPANY table.
Securities	Number of rows in SECURITY table.

Table 2.31 – Scaling rules for TPC-E

Table	Cardinality	Formula
CUSTOMER	5,000	Scale based on transaction rate.
CUSTOMER_TAXRATE	10,000	Customers * 2.
CUSTOMER_ACCOUNT	25,000	Customers * 5.
ACCOUNT_PERMISSION	˜35,500	Accounts *1,42 (Average ˜1,42 permission per account).
COMPANY	2,500	Customers * 0.5.
COMPANY_COMPETITOR	7,500	Companies * 3;
ADDRESS	7,504	Companies + EXCHANGE (4) + Customers.
BROKER	50	Customers * 0.01.
FINANCIAL	50,000	Companies * 20.
NEWS_ITEM	5,000	Companies * 2.
NEWS_XREF	5,000	Companies * 2.
SECURITY	3,425	Customers * 0,685.
LAST_TRADE	3,425	Securities * 1.
DAILY_MARKET	4,469,625	Securities * 1,305.
WATCH_LIST	5,000	Customers * 1.
WATCH_ITEM	˜500,000	Customers * 100 (Average ˜100 securities per watch list).

2.5.3 Transactions

TPC-E contains 12 different transactions. Ten of them are executed in the regular transaction mix, while the two others, DataMaintenance and TradeCleanup, are executed separately. DataMaintenance is executed periodically, every minute, alongside with the transaction mix, whereas TradeCleanup is not considered as part of the test, it is used to bring the database to a known state before the workload test starts, in order to clean up the submitted or pending trades from a previous workload execution.

Table 2.32, copied from (Tözün P, 2013), shows the 10 transactions in the mix and the percentage of each transaction in total workload execution.

Table 2.32 – Description of TPC-H Region Table

Transaction	Weight	Access	Category	% in Mix
BrokerVolume	Mid to Heavy	RO	BI	4.9
CustomerPosition	Mid to Heavy	RO	CI	13
MarketFeed	Medium	RW	MT	1
MarketWatch	Medium	RO	CI	18
SecurityDetail	Medium	RO	CI	14
TradeLookup	Medium	RO	BI/CI	8
TradeOrder	Heavy	RW	CI	10.1
TradeResult	Heavy	RW	MT	10
TradeStatus	Light	RO	CI	19
TradeUpdate	Medium	RW	BI/CI	2
BI: Brokerage Initiated, CI: Customer Initiated, MT: Market Triggered				

The summary of each transaction is given as follows:

Broker-Volume: This transaction is designed to emulate a brokerage house's "up-to-the-minute" internal business processing. An example of a Broker-Volume Transaction would be a manager generating a report on the current performance potential of various brokers.

Customer-Position: Retrieves a customer's profile and summarizes their overall standing based on current market values for all assets.

Market-Feed: Tracks the current market activity.

Market-Watch: Monitors the overall performance of the market by allowing a customer to track the current daily trend (up or down) of a collection of securities.

Security-Detail: Accesses detailed information on a particular security.

Trade-Lookup: Retrieves information by either a customer or a broker to satisfy their questions regarding a set of trades.

Trade-Order: Buys or sells a security by a Customer, Broker, or authorized third-party.

Trade-Result: Completes a stock market trade. This is representative of a brokerage house receiving from the market exchange the final confirmation and price for the trade.

Trade-Status: Provides an update on the status of a particular set of trades.

Trade-Update: Makes minor corrections or updates to a set of trades. This is analogous to a customer or broker reviewing a set of trades, and discovering that some minor editorial corrections are required.

Data-Maintenance: Performs the periodic modifications to data that is mainly static and used for reference. This is analogous to updating data that seldom changes.

Trade-Cleanup: Used to cancel any pending or submitted trades from the database.

2.5.4 Transactions Input Generation

TPC-E scales with customers and allows customer information to be partitioned into groups of related customers. This process is called Customer Partitioning. The advantage of Customer Partitioning is that it increases locality of reference within each sub-group of Customers.

If Customer Partitioning is being used, which is the case in our tests, the following rules will be applied to generate a customer identifier, account identifier or customer tax identifier:

- 50% of the time the data is selected from the partition's range of customers.
- 50% of the time the data is selected from the entire range of customers.

When Customer Partitioning is not used, the customer identifier, account identifier and customer tax identifier are generated from the entre range of customers.

The inputs for each transaction and their descriptions are presented in Tables 2.33 – 2.44.

Table 2.33 – TPC-E Broker-Volume transaction inputs.

Transaction	**Input parameter**	**Description**
Broker-Volume	broker_list [ ]	A list of twenty to forty distinct broker name strings refers to the column B_NAME in BROKER table. Names are randomly selected from the broker range, with, uniform
	sector_name	A randomly selected sector name string refers to the column SC_NAME in SECTOR table using uniform distribution

Table 2.34 – TPC-E Customer-Position transaction inputs.

Transaction	Input parameter	Description
Customer-Position	acct_id_idx	Index to one of the customer's account.
	cust_id	Customer id.
	get_history	Selected as 1 to get the history of the customer, and 0 otherwise.
	tax_id	Customer tax id.

Table 2.35 – TPC-E Market-Feed transaction inputs

Transaction	Input parameter	Description
Market-Feed	price_quote[ ]	A list of numeric prices generated for each entry on the ticker list. Each security prices fluctuates between a low and a high price. The fluctuation has a predefined frequency.
	Status_submitted	The string ID value for the STATUS_TYPESubmitted status.
	symbol[ ]	A list of strings containing the Security Symbol for each security on the ticker. The security symbol string refers to the column LT_S_SYMB in the LAST_TRADE table.
	trade_qty[]	A list of numbers representing the number of shares of a security that were traded for this ticker entry. The trade_qty is the same as the trade_qty requested in the Trade Request.
	Type_limit_buy	The string id value for the TRADE_TYPELimit-Buy type.
	Type_limit_sell	The string id value for the TRADE_TYPELimit-Sell type.
	Type_stop_loss	The string id value for the TRADE_TYPELimit-Loss type.
	Unique_symbols	The number of unique security symbols in the ticker stream.

Table 2.36 – TPC-E Market-Watch transaction inputs

Transaction	Input parameter	Description
Market-Watch	acct_id	A single customer id is selected and, after that, a account id from this selected customer is randomly selected.
	cust_id	The customer id.
	ending_co_id	Company identifier of the last company in the range of 5,000 companies to be searched for companies in IN_NAME industry. The value will be starting_co_id + 4,999. This input will only be used when industry_name is used which is 5% of the time. The other 95% of the time when this input is not being used its value will be zero.
	industry_name	A randomly selected industry name string, which refers to column IN_NAME in INDUSTRY table. This input will be used 5% of the time. The securities collection will be all the securities of companies in this industry. The other 95% of the time when this input is not being used its value will be zero.
	start_date	A date non-uniformly selected from the 1305 days in the DAILY_MARKET table.
	starting_co_id	A number randomly selected from the range of possible company identifiers minus 4,999. Company identifier of the first company in the range of 5,000 companies to be searched for companies in IN_NAME industry. This input will only be used when industry_name is used which is 5% of the time. The other 95% of the time when this input is not being used its value will be zero.

Table 2.37 – TPC-E Security-Detail transaction inputs

Transaction	Input parameter	Description
Security-Detail	access_lob_flag	If 1, access the complete news articles for the company. If 0, access just the news headlines and summaries.
	max_rows_to_return	An integer value, randomly selected between 5 and 20 with a uniform distribution. This value determines how many rows must be returned from the DAILY_MARKET table for this security.
	start_day	A date randomly selected from a uniform distribution of dates between 3 January 2000 and max_rows_to_return days before 1 January 2005.
	symbol	Security symbol, randomly selected from a uniform distribution

Table 2.38 – TPC-E Trade-Lookup transaction inputs

Transaction	Input parameter	Description
Trade-Lookup	acct_id	Customer account id.
	end_trade_dts	Used as the end point in time to identify trades.
	frame_to_execute	This transaction is divided in 4 different parts, each of these parts is called Frame. Two of these Frames are mutually exclusive. This parameter informs which of the mutually exclusive frames must be executed.
	max_acct_id	Used to identify the maximum customer account ID.
	max_trades	Used to find the number of trades.
	start_trade_dts	Used as the start point in time to identify trades.
	symbol	Used as the security symbol for which to find trades
	trade_id[ ]	Array of non-uniform randomly chosen trade IDs used to identify a set of particular trades.

Table 2.39 – TPC-E Trade-Lookup transaction inputs

Transaction	Input parameter	Description
Trade-Order	acct_id	Customer account id.
	co_name	Company name. This parameter can be used together with the parameter issue to specify the security being trade in this transaction. The other option to do so is to use the parameter symbol, but in this case co_name and issue are empty strings.
	exec_f_name	First name of the person executing the trade.
	exec_l_name	Last name of the person executing the trade.
	exec_tax_id	Tax identifier for the person executing this trade.
	is_lifo	If this flag is set to 1 then this trade will process against existing holdings from newest to oldest (LIFO order). If this flag is set to 0, then this trade will process against existing holdings from oldest to newest (FIFO order).
	issue	This parameter can be used together with the parameter co_name to specify the security being trade in this transaction. The other option to do so is to use the parameter symbol, but in this case co_name and issue are empty strings.
	requested_price	For a limit order, this is the requested price for triggering the trade.
	roll_it_back	If this flag is 1 then an intentional rollback is executed. If 0, then a commit is executed.
	st_pending_id	Identifier for the “Pending” order status.
	st_submitted_id	Identifier for the “Submitted” order status
	symbol	This parameter can be used to specify the security being trade in this transaction. The other option to do so is to use the parameters co_name and issue together, but in this case symbol is an empty strings.

	trade_qty	The number of shares to be traded for this order.
	trade_type_id	Identifier indicating the type of trade
	type_is_margin	If this flag is set to 1, then the order will be done on margin. If the flag is set to 0, then this trade will be done with cash.

Table 2.40 – TPC-E Trade-Result transaction inputs

Transaction	**Input parameter**	**Description**
Trade-Result	trade_id	The Trade ID for the trade to be settled. Trade ID is the primary key of the TRADE table.
	trade_price	The price of the trade.

Table 2.41 – TPC-E Trade-Status transaction inputs

Transaction	**Input parameter**	**Description**
Trade-Status	acct_id	A single customer is chosen non-uniformly from the range of available customers. After that, a single customer account id, which refers to CA_ID in CUSTOMER_ACCOUNT table, is chosen at random, uniformly, from the range of customer account ids for the chosen customer.

Table 2.42 – TPC-E Trade-Update transaction inputs

Transaction	Input parameter	Description
Trade-Update	acct_id	Customer account ID.
	end_trade_dts	Used as the end point in time to identify trades.
	frame_to_execute	This transaction is divided in 4 different parts, each of these parts is called Frame. Two of these Frames are mutually exclusive. This parameter informs which of the mutually exclusive frames must be executed.
	max_acct_id	Used to identify the maximum customer account ID.
	max_trades	Max number of trades to find.
	max_updates	Maximum number of trades to be modified.
	start_trade_dts	Used as the start point in time to identify trades.
	symbol	Used as the security symbol for which to find trades
	trade_id[ ]	Array of non-uniform randomly chosen trade IDs used to identify a set of particular trades.

Table 2.43 – TPC-E Data-Maintenance transaction inputs

Transaction	Input parameter	Description
Data-Maintenance	acct_id	Customer account ID selected using non-uniform distribution.
	c_id	Customer ID randomly selected using uniform distribution.
	co_id	Company ID, which refers to CO_ID in COMPANY table. This parameter is randomly selected using uniform distribution
	day_of_month	A number randomly selected from 1 to 31 with a uniform distribution.
	symbol	A string containing a security symbol.
	table_name	A string containing the name of the table to be altered. Valid values are "ACCOUNT_PERMISSION", "ADDRESS", "COMPANY", "CUSTOMER", "CUSTOMER_TAXRATE", "DAILY_MARKET", "EXCHANGE", "FINANCIAL", "NEWS_ITEM", "SECURITY", "TAXRATE", "WATCH_ITEM".
	tx_id	A string containing a tax identifier. This parameter refers to TX_ID in TAXRATE table.
	vol_incr	A randomly selected positive or negative number. This number is only used when the table_name is "DAILY_MARKET", otherwise vol_incr is set to 0 and ignored. When table_name is "DAILY_MARKET" this number is added to DM_VOL.

Table 2.44 – TPC-E Trade-Cleanup transaction inputs

Transaction	Input parameter	Description
Trade-Cleanup	st_canceled_id	Identifier for the "Canceled" trade order status.
	st_pending_id	Identifier for the "Pending" trade order status.
	st_submitted_id	Identifier for the "Submitted" trade order status.
	trade_id	The trade identifier to be used as the start for handling outstanding submitted and/or pending limit trades.

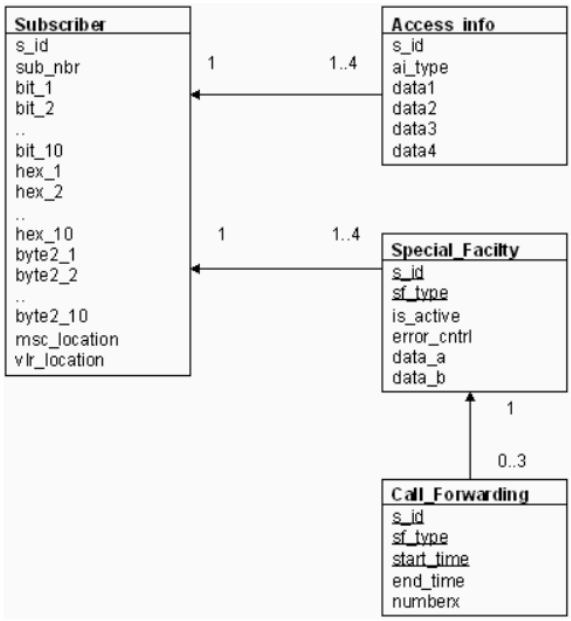
2.6 TATP

TATP (TATP Benchmark description, 2009) is a benchmark designed to test the interplay of DBMS, OS, and hardware in a typical Home Location Register (HLR) database of a mobile phone network. According to (TATP Benchmark description, 2009), HLR is a database that mobile network operators use to store information about their customers and the services to which they have subscribed.

2.6.1 Entities, Relationship and Characteristics

The TATP database is composed of four tables. Their columns and relationships are shown in Figure 2.7.

Figura 2.7 – TATP database diagram



Fonte: TATP Benchmark Description (2009, p. 4).

Table 2.45 gives the relationships between the tables, specifying it by columns, and a brief description of the columns of each table. Some columns are omitted here—their values are randomly generated following their own data type.

Table 2.45 – Relationship and description of main TATP table columns

Table	Column	Key Constraints	Description
SUBSCRIBER	S_ID	PK	The unique number that identifies the subscriber.
	SUB_NBR		15-digit string generated from S_ID by transforming S_ID to string and padding it with leading zeros. For example, if S_ID is 123 then SUB_NBR is "000000000000123".
ACCESS_INFO	S_ID	FK (S_ID in SUBSCRIBER table.)	
	AI_TYPE		Number between 1 and 4 and it is randomly chosen, but there can be only one record of each AI_TYPE per each subscriber. So if there are four Access_Info records for a certain subscriber they have values 1,2,3 and 4.
SPECIAL_FACILITY	S_ID	PK+ FK (S_ID in SUBSCRIBER table.)	
	SF_TYPE	PK+	Number between 1 and 4 and it is randomly chosen, but there can be only one record of each AI_TYPE per each subscriber. So if there are four Access_Info records for a certain subscriber they have values 1,2,3 and 4

	IS_ACTIVE		It is either 0 or 1. IS_ACTIVE is chosen to be 1 in 85% of the cases and 0 in 15% of the cases.
CALL_FORWARDING	S_ID	PK+ FK+(S_ID and SF_TYPE in SPECIAL_FACILITY table)	
	SF_TYPE	PK+ FK+(S_ID and SF_TYPE in SPECIAL_FACILITY table)	
	START_TIME	PK+	It can have values 0, 8 and 16, representing, respectively, midnight, 8 o'clock and16 o'clock.

2.6.2 Scaling Rules

There is a fixed relation between the amounts of rows in all tables. Table 2.46 shows this relation for an amount of 1,000,000 rows in the Subscriber table.

Table 2.46 – Scaling rules for TATP

SUBSCRIBER	1,000,000
ACCESS_INFO	~ 2,500,000
SPECIAL_FACILITY	~ 2,500,000
CALL_FORWARDING	~ 3,750,000

The number of rows in the Subscriber table is used to scale the population of the other tables. The official population sizes used in TATP are: 100,000, 200,000, 500,000, 1,000,000, 2,000,000 and 5,000,000 subscribers.

2.6.3 Transactions

TATP consists of 7 transactions, which perform insert, update, delete, and query operations. Transactions may not always succeed because the keys are randomly generated,

and some of these values randomly chosen may not be present in the database. Table 2.47 gives the probabilities for each transaction succeeding.

Table 2.47 – Probability for TATP transactions succeed

Transaction	**Probability of success (%)**
GET_SUBSCRIBER_DATA	100
GET_NEW_DESTINATION	23.9
GET_ACCESS_DATA	62.5
UPDATE_SUBSCRIBER_DATA	62.5
UPDATE_LOCATION	100
INSERT_CALL_FORWARDING	31.25
DELETE_CALL_FORWARDING	31.25

Each TATP has assigned a probability to be executed. Tables 2.48 and 2.49 show this probability for each transaction.

Table 2.48 – Probability for TATP read transactions execute

Read Transactions (80%)	
GET_SUBSCRIBER_DATA	35%
GET_NEW_DESTINATION	10%
GET_ACCESS_DATA	35%

Table 2.49 – Probability for TATP write transactions execute

Write Transactions (20%)	
UPDATE_SUBSCRIBER_DATA	2%
UPDATE_LOCATION	14%
INSERT_CALL_FORWARDING	2%
DELETE_CALL_FORWARDING	2%

The summary of each transaction is given as follows:

GET_SUBSCRIBER_DATA: Receives s_id as input parameter and retrieves one row from the SUBSCRIEBER table.

GET_NEW_DESTINATION: Searches for the current call-forwarding destination. It receives s_id, sf_type and start_time as input parameters..

GET_ACCESS_DATA: Used to consult the access validation data. Its input parameters are s_id and ai_type.

UPDATE_SUBSCRIBER_DATA: Updates the service profile data and receives s_id, sf_type, bit_rnd and data_a as input parameters.

UPDATE_LOCATION: Changes the location of a subscriber identifier by the input parameter sub_nbr. The new location is received in the input parameter vlr_location.

INSERT_CALL_FORWARDING: Adds a new call forwarding info. The input parameters necessary for this transaction are sub_nbr, sf_type, start_time, end_time and number.

DELETE_CALL_FORWARDING: It receives sub_nbr, sf_type and start_time as parameters and removes a call forwarding info.

2.6.4 Transactions Input Generation

The inputs are generated according to the rules specified in Table 2.50.

Table 2.50 – Rules to generate transaction inputs in TATP benchmark

Parameter	**Rule to generate parameter**
s_id	It is randomly selected from [1..P], where P is the size of the SUBSCRIBER table.
sf_type	A number randomly selected from [1..4].
start_time	It can assume the values 0, 8 and 16. This parameter is randomly generated using one of these values.
ai_type	Randomly selected from [1..4].
bit_rnd	It is a parameter of 1 bit, which can assume the values 0 or 1. The value is randomly selected.
data_a	Assumes values from 0 to 255.
vlr_location	Simulates an address and it can assume values from 1 to 2ˆ32-1. The value is randomly chosen.
sub_nbr	The sub_nbr column holds a string representation of the s_id number. Its value range is [1..P], where P is the size of the SUBSCRIBER table. It is randomly generated from its value range.
end_time	This input parameter is randomly generated from 1 to 24.
numberx	String of length 15 characters. A number between [1..P], where P is the size of the SUBSCRIBER table.

2.7 Comparison of the benchmarks

Below, we present a short comparison between all presented benchmarks. Table 2.51 is adapted from (Tözün P, 2013) in order to include TATP and TPC-H. The table presents the following information:

- First release: When each benchmark was launched.
- Business model: The type application simulated by each benchmark.
- Tables: How many tables have fixed size independent of the scale factor used, how many tables are affected by the scale factor, how many tables grow while the benchmark is executing, and the total of tables in the benchmark.
- Transactions: The amount of transactions type which perform read-write operations and the amount of transactions type which perform read-only operations.
- Transaction mix: The percentage of transactions executed by the benchmark performing read-write and read-only operations.
- Transactions using secondary indexes: From the total of transactions type, how many of them use secondary index.
- Data population: How the database is populated. In this case is used or random data or pseudo-real data.

Table 2.51 – Benchmark comparison

		TPC-B	TPC-C	TPC-E	TATP	TPC-H
First release		Aug 1990	Aug 1992	Feb 2007	May 2003	Feb 1999
Business model		Banking	Wholesale supplier	Brokerage house	Mobile network operators	Wholesale supplier
Tables	Fixed	0	1	9	0	2
	Scaling	3	4	16	4	6
	Growing	1	4	8	0	0
	Total	4	9	33	4	8
Transactions	RW	1	3	6	4	22
	RO	0	2	6	3	0
Transactions Mix	RW	100%	92%	23.1%	20%	0%
	RO	0%	8%	76.9%	80%	100%
Transactions Using Secondary Indexes		None	2	10	1	n/a
Data population		Random	Random	Pseudo-Real	Random	Random

When the percentage of read-write and read-only are presented, TPC-H shows only read-only transactions, since it contains only consulting queries.

2.8 Shore-MT, the Storage Manager

In this subsection we present how the levels of abstraction are divided in a DBMS, the reasons to choose a storage manager to perform the tests, and Shore-MT as the storage manager chosen. We also give some concepts about the internal components of a storage manager, which are used in the analysis of the experiments.

According to Härder, T (2005), the Database Management System is divided into 5 levels of abstraction, as we can see in Figure 2.8, copied from (Härder, T, 2005). Functionalities provided by DBMS, such as Triggers, Functions and Procedures, SQL front-end, application programming interfaces (e.g. JDBC and ODBC), among many others, exist only in modules that interact with the levels of abstraction L4 and L5. The levels L1, L2 and L3 are implemented in the DBMS by the storage manager module. At these 3 levels, there is no relational database, tuples, or records. All data are treated only as physical records, segments, pages, files and blocks. This module is considered the heart of the DMBS because it is the responsible for communicate the application layer with the hardware. It manages the transactions, the memory access, the disk storage, the recovery from failures and everything that involves these operations.

Figure 2.8 – Level of abstraction in a DBMS

	Level of abstraction	Objects	Auxiliary mapping data
L5	Nonprocedural or algebraic access	Tables, views, tuples	Logical schema description
L4	Record-oriented, navigational access	Records, sets, hierarchies, networks	Logical and physical schema description
L3	Record and access path management	Physical records, access paths	Free space tables, DB-key translation tables
L2	Propagation control	Segments, pages	DB buffer, page tables
L1	File management	Files, blocks	Directories, VTOCs, etc.

Source: Härder T (2005, p. 5).

As our study is focused only on the characteristics of the storage manager module, there is no need to use a complete DBMS in our tests and, for this reason, we chose to work only with a storage manager.

Shore-MT (Johnson R, 2009) is the storage manager chosen to perform the workloads. It is based on the Shore storage manager developed at the University of Wisconsin, Madison

(Carey M, 1994). Shore-MT is an experimental test-bed library, what means that there is no executable file to run directly the storage manager. In order to perform any operations, it is necessary first to create an implementation using this library..

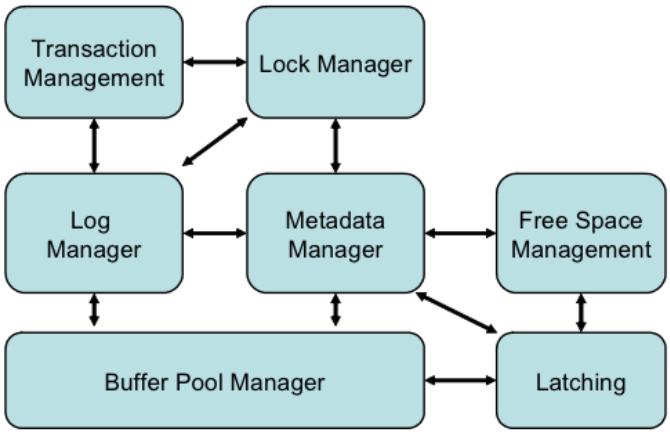
Shore-MT was developed to provide the scalability required for modern multicore hardware, since its open-source peers, such as MySQL and PostgreSQL, according to (Johnson R, 2009), do not attend this requirement. The strategy to create Shore-MT was to eliminate the scalability bottlenecks found in Shore when running on multicore hardware, focusing first on scalability, with single-thread performance as a secondary goal. As some tests have shown (Johnson R, 2009), this approach allowed Shore-MT to have a better performance than its peers when running eight threads or more.

In order to verify Shore-MT performance, a suite of database benchmarks, known as Shore-Kits, was implemented for Shore-MT.

Today, the benchmarks supported by Shore-Kits are TPC-B, TPC-C, TPC-E, TATP, TPC-H and SSB. In order to implement these benchmarks, Shore-Kits implements some of the functionalities of the levels of abstraction L4 and L5 (listed in Figure 2.8), such as create queries, and a relational database with tables records, tuples, etc. Since Shore-MT does not have an SQL front-end, these benchmarks are implemented in C++ through Shore-MT storage manager API calls. This API is linked as a static library to the Shore-Kits executable.

Figure 2.9, copied from (Johnson R, 2009), shows the main components of the storage manager and how they interact with each other. In order to analyze workload behavior, we focus on three major components, which concentrate most of the storage manager's activities: Log Manager, Lock Manager and Buffer Pool Manager.

Figure 2.9 – Components of a storage manager



Source: Johnson R (2009, p. 26).

2.8.1 Log Manager

The main goal of the Log Manager is to log all operations which alter user data, in order to ensure that they are not lost or corrupted in case of a failure.

The Shore-MT Log Manager was developed, as most of current database engines, based on the ARIES logging and recovery method (Mohan C, 1992). It basically makes use of a technique called WAL (Write Ahead Logging), where before the changes made by a transaction modify the stored pages, they are first written in records that are appended to the log. As the log is written sequentially, it is much faster to write the changes in the log (i.e., a no-force policy) than to write database pages which are randomly accessed (force).

As the log manager assures that all modifications produced by a transaction are saved, a transaction is not considered complete until its all log records are stored in the disk, avoiding loss of information and guaranteeing the atomicity and durability properties (Härder, T, 1983).

2.8.2 Buffer Pool Manager

The Buffer Pool Manager creates the illusion for the database engine that all pages are in main memory, just like the virtual memory manager in an operating system (Johnson R, 2009).

When a page is requested, and this page is not in the buffer pool, the buffer manager reads this page from disk and fixes it in the buffer pool, after that it latches the page as shared, for read operations, or as exclusive, for update operations (Mohan C, 1992). Once the page is in the buffer pool, it can be accessed directly with no need of access to the disk. If a page in buffer pool must be latched with a different mode, a re-fix operation is performed, changing the latch to the new mode.

When a page in buffer pool has a different version from the page in disk, because this page suffered an update operation, it is considered as dirty and must be written into the disk before being evicted.

2.8.3 Lock Manager

The Lock manager is the component responsible to ensure the Isolation property of ACID, forbidding transactions to interfere with the execution of the other concurrent transactions.

When a transaction acquires a lock, it may be obtained in different modes, such as shared, exclusive, intention shared, intention exclusive and shared intention exclusive and in different granularities, such as volume, store, page, key value and record (tuple) (Mohan C, 1992). The Lock manager in Shore-MT, as in most of modern database engines, uses the concept of hierarchical (or intention) locks. When a transaction wants to modify a row, for example, the transaction acquires a database lock, table lock and row lock (Johnson R, 2009).

3 GATHERING STATISTICS

In this chapter is given a description of the methodology used to analyze the page access skew and internal component statistics. Furthermore, the implementation steps required to collect the information necessary for our analysis are described.

3.1 Methodology overview

By benchmarking a database system, it is possible to focus on specific characteristics, which allow a better understanding of the system behavior. The first characteristic analyzed is the access to pages. In the analysis of this characteristic, we present the page access skew and find patterns of access and updates for each benchmark. By analyzing the page access pattern we can understand how access in logical layer (tables, columns, etc…) are related to the access in physical layer (pages) and also how the page access skew is presented, since a higher degree of skew implies in more concurrency and performance loss. Another study realized is about how the benchmarks properties stress the components buffer manager, lock manager, and transaction manager. We realize such study by counting the amount of page fixes performed during the benchmark execution, and the type of each fix, we also compare locks acquired in the different granularities available in the DBMS, and we verify how many transactions were created and how many of them aborted.

3.1 The Log

In order to investigate how different configurations of each benchmark access the pages of the database, we analyze the log created by the Log Manager component. The log records all actions from transactions when there are changes to data records, ensuring that a committed action is durable even in the presence of failures.

Log records are identified by a log sequence number (LSN), which is assigned in ascending order to log records when they are appended to the log. Each log record has a header and a type-dependent body that describes the operation that generated it. The header is composed by the LSN, the ID of the transaction that wrote it, the previous LSN written by the same transaction, and the length of the log record body. The log record body contains information regarding the update that was performed. This information is used to determine the appropriate action to be executed by redo and/or undo (Gray J, 1993).

Transaction status is also stored in the log. When a transaction is being executed, its log records are written first in memory (log buffer). When a commit happens, the transaction generates a commit log record and all log records up to the last LSN of the committing

transaction are written to stable storage. The transaction is not considered completed until all its log records are written to disk.

The policy utilized to write the log is WAL (Write-Ahead Logging), which asserts that before a change from a transaction is effectively written to disk, it must be first logged on stable storage. Using WAL, the system is not allowed to proceed in the opposite way, saving the changed data in database stored version before the log describing this change is already stored. That is the only restriction on the buffer page replacement policy and it reassures the fact that if a failure occurs, all steps executed by transactions can be recovered from the log and reapplied to the system.

According to Gray and Reuter (Gray J, 1993), WAL is stated in the following rules:

- Each volatile page has a LSN field naming the log record of the most recent update to the page.
- Each update must maintain the page LSN field.
- When a page is about to be copied to disk, the copier must first ask the log manager to copy all log records up to and including the page's LSN to durable memory. This is called forcing the log up to that LSN.
- Once the force completes, the volatile version of the page (e.g., page in memory) can overwrite the persistent version of the page (e. g., page in disk)
- The page must be fixed during the writes and during the copies, to guarantee page-action consistency.

Every page in database contains a field called page LSN to track its version. Whenever a page is updated, its LSN field is also updated with the LSN of the log record that reflects this update. If a failure occurs, it enables the system to verify which was the last log record applied to that specific page and from which log record the recovery operation must begin.

It is possible to add custom log records to the log in order to track states of the database system in operation. We used such approach to identify the transactions being executed in the system with the benchmarks transactions described in Section 2. This relation allows us to identify which benchmark transaction performed which operation in the system.

We created in Shore-Kits a method to identify the transactions specified by the benchmarks. This method, which is called before each transaction starts, creates a custom log record informing the name of the benchmark transaction and the transaction id related to it.

As the log manager does not log page read operations, we also used custom log records to track these operations. These custom log records contain information about the

page being read and the ID of the transaction that is performing this operation, which allows us to collect for read operations the same statistics that we collect for update operations.

3.2 The Loginspect tool

For analyzing the log, including the custom log records described above, we developed a tool called Loginspect, which analyzes the log produced by a benchmark execution. The goal is to generate statistics and collect others information from our interest. From the Loginspect perspective, the log provides a detailed trace of actions performed during a benchmark. Furthermore, it also serves the purpose of debugging.

Loginspect is composed of a set of commands, where each command processes the log in a distinct way, giving different kinds of results. Below, a list of commands implemented in Loginspect for our analysis is given. These commands give a better perspective of how the information from log can be analyzed.

- SKEW: This command is used to verify the skew of page access. It accepts three parameters: "-s", to sort the result; "-t", to give the result in histogram format and "-b" to give the result in a boxplot format.
- TRACE: This command is used to trace all page accesses presented in log. This result is used in a graph to demonstrate how the access to pages is distributed, how the access occurs during the time and where accesses are concentrated in a skewed workload.
- TRACETRANSACTIONS: This command gives the same information of the command TRACE, but it separates the access by benchmark transaction (e.g., NewOrder and Payment in TPC-C), illustrating the different access patterns of each transaction.

3.3 Page access characteristics

With the results given by the Loginspect tool, we can analyze and compare the benchmarks regarding the page access characteristics. These properties mainly express the page access skew: how it is presented by each benchmark and how the page access skew behaves when we change high-level properties of the benchmark. For the analysis of the page access properties, we use 3 approaches.

In the first approach, we simply create a histogram showing the page access skew. This chart shows the distribution of the page access and helps us to understand how concentrated it is. By comparing the histograms of each benchmark, we can visualize how

different the skew are among them and, by changing high-level characteristics of the benchmark, we can visualize how these changes affect the page access skew.

In the second approach, we use a boxplot chart to see how concentrated the page access is. By using boxplots, we can easily compare how are the distributions of page access among the benchmarks.

The third and last approach uses the TRACE command from the loginspect tool to show how the page access occurs through time. When a new page is accessed, a new point is created in our chart. As there is a big set of points in the chart, in some cases it appears as there are more than one page access happening at the same time, however, it does not happen. What occurs is that those points are so close to each other that it is not possible to see the time difference between them. However, as we want to understand the behavior of the page access, this proximity of the points does not affect our analysis.

3.4 Counters

In order to study the behavior of the DBMS components, the Shore-MT code was modified to collect statistics while the benchmark is running. These statistics are collected through counters. These counters are used inside the code block of each specific operation of our interest. For example, there is a counter in Lock Manager which is incremented every time that the object page is locked. So, at the end of the benchmark execution, we have the statistics of the total amount of page locks performed. The results of these statistics are then analyzed and discussed in Section 4.

3.5 Transaction counters

The comparison of benchmarks behaviors enables a better understanding of DBMS components, which can help in decisions of application design. For example, benchmarks containing only OLAP queries, as it is the case for TPC-H, are expected to create fewer transactions than a benchmark containing only OLTP transactions (as we benchmark the system the same amount of time for each benchmark), especially if the transactions created are very simple like in TATP benchmark. We verify this statistic by comparing the values presented in the counter begin_xct_cnt, which is incremented when every transaction is created and, in the end, shows us the amount of transactions created by the system after each benchmark was executed.

As already described, TPC-B benchmark contains only an update-intensive transaction kind, what implies in significant I/O. Meanwhile TPC-E, TPC-C, and TATP are a mix of read-only transactions with read-write transactions. We compare how type of benchmark transactions affects the total of committed transactions created by each benchmark through the counter commit_xct_cnt.

Finally, TPC-E and TATP benchmarks are expected to have transactions failures and rollbacks, while other benchmarks do not mention this situation, even though some failures are expected due their specification. We verify the amount aborted transactions for each benchmark and analyze how it behaves according to their specifications. In order to do such analysis, we verify all aborted transaction showed in the abort_xct_cnt counter and we relate it to the benchmark specifications.

3.6 Buffer manager counters

The main goal of the buffer manager is to give access to pages by fixing them in the buffer pool and, when necessary, latching them in shared or exclusive mode, depending on the operation to be executed by the transaction requiring the page. Many transactions accessing the same pages can cause performance loss, because update operations in pages, or in objects smaller than a page, require exclusive latches.

For the statistics regarding the latching process, we count how many times the buffer has latched pages in Shared mode and in Exclusive mode. There are two moments where this process occurs: first when a page is fixed in the buffer and, in sequence, this page is latched. Second when a page is already in the buffer and a transaction needs to perform a different operation with this page, so the latch must be changed. For this moment, a function to change the latch is performed by the buffer. As we want to verify how many times the pages are latched in exclusive mode and in shared mode, we added counters to verify the amount latches acquired.

Regarding page fix, we count every time the code which fix a page in the buffer is executed, through a counter called page_fix_cnt. We use this counter to compare how the benchmarks stress the Buffer Manager and how it is affected when we vary properties of each benchmark.

Finally, we also collect statistics about how many dirty pages in the buffer are written in the disk. Since writing the pages in sequence in the disk is much faster than writing them randomly, we want to know how this write process is executed by each benchmark and how it

changes when we vary properties of the benchmark. These values are present in 9 different counters, but we divided them in only two groups: one page write and multiple pages write.

3.7 Locking counters

Lock Manager ensures the isolation of the executing transaction by locking the objects accessed by these transactions. One of the most important features introduced by ARIES is the concept of fine-granularity locking (Mohan C, 1992), which enables the database to lock objects with sizes smaller than pages, such as records and keys. This feature creates the possibility to concurrently access objects smaller than a page.

For this case where the objects are smaller than a page, the Buffer Manager fix the page in the buffer, latch it in Shared mode for read operations or in Exclusive mode for update operations, and the Lock Manager locks the necessary objects inside this page, such as records or keys. In case of the granularity is a page or something coarser than a page, there is no need to the Buffer Manager to use latches, since the lock is sufficient to ensure the isolation of the operations.

Lock Manager, in Shore-MT, locks 5 different types of objects: volume, store, page, key, and record. We collect the total amount of locks acquired for each type of object after the benchmark, and then we compare these results with all results obtained by the different benchmarks executed. We use these counters to see and compare which are the benchmarks that need to acquire more locks and the type of objects most locked by each benchmark.

Exclusive locks of a determined object cannot be acquired when others transactions have the lock from this object. When it occurs, the transaction must wait the others locks be released first. We verify how many times this process occurs through the lock wait counter and, with this information, we can see the benchmarks, and consequently the type of applications, which suffers more performance loss because of locking operations.

4 EXPERIMENTS

In this chapter we describe the setup to execute the experiments, and the results found with an analysis over page access and the counters of transactions, buffer manager, and locking events.

4.1 Experimental Setup

All benchmarks in this chapter were executed in Shore-Kits using 4 worker threads on the same hardware. Each benchmark runs for a total of 90 seconds on an already loaded database. Worker threads are used in Shore-Kits to simulate the terminals of clients which perform the benchmark operations that are executed as transactions in the system. As such, the number of worker threads determines the level of multi-programming in the system, potentially increasing parallelism and concurrency.

In Shore-Kits, a command called skew is used to alter the skew of data accesses in the benchmark. It defines how transaction inputs (e.g., the ID of an object) are distributed on the dataset. When a user transaction is initiated in Shore-Kits, its input parameters are generated by a function whose behavior is determined by the command skew. When the command is not used, this function uses a uniform distribution to generate such parameters. If the command skew is used, the function parameters are changed and the function returns the data according to the distribution set by the skew command. The set of parameters subject to the random generation are based in the unit of scale for each benchmark. In TPC-C, for instance, in a dataset of 10 warehouses, a 70-30 skew means that 70% of the accesses are restricted to 3 warehouses.

The dataset of benchmarks TPC-B, TPC-C, and TATP were loaded with scale factor 10. TPC-E had its database loaded with scale factor 1 and TPC-H with scale factor 0.1. In the latter two benchmarks, we did not change the default skew behavior due to difficulties with the benchmark implementation.

We ran all benchmarks first without applying the skew command, which means that every data accessed is selected randomly, with uniform probability, amongst all scale units. In a second moment, we use the skew command to run the benchmarks TPC-B, TPC-C, and TATP in order to have 70% of the accesses in a subset of 30% of the benchmarks dataset.

The server used to run the benchmarks has two Intel Xeon E5420 processors, each with 4 out-of-order cores running at 2.5 GHz and having a cache size of 6 MB. The server has 32 GB of RAM. The operating system running in the server is Ubuntu 14.04.1 LTS. Since our

goal is to analyze the application behavior and not performance metrics such as transaction response time or throughput, all data is stored in main memory.

4.2 Page access analysis

In order to analyze the page access patterns for each of the five benchmarks already presented and see how these patterns change when we alter the skew, we generated the charts showed in Figures 4.1 – 4.23. Throughout this section, we present each of them in detail and try to identify patterns and causes for some of the observed effects.

There are three types of charts used in our analysis. The first type shows the pages accessed by each request over time. In these charts, as the one presented in Figure 4.1, the x-axis refers to the sequence of page fix requests created by the system, while the y-axis shows the identifier of the database page accessed. Since the page identifier does not have any meaning in our graph and the position of the points is enough to make our analysis, there is no absolute value displayed in the y-axis. For each benchmark, we present versions of this chart for both read-only and read-write accesses.

The second type of chart used in this work shows the access skew by means of a histogram of page accesses, sorted from the least accessed to the most accessed page identifier. Figure 4.5 shows one of such charts. These charts show in the y-axis the total of accesses in the set of pages grouped in the x-axis. Each set of pages in the x-axis shows the percentage that they represent from the total amount of pages accessed during the benchmark execution.

Finally, the third type of chart compares skew between benchmarks using a normalized boxplot graph. The only example in this work is Figure 4.23, which we discuss in Section 4.2.6.

4.2.1 TPC-B

Figure 4.1 shows the pages being accessed by each request done by the system while TPC-B was running with uniform distribution of input parameters (i.e., without the skew command). Figure 4.2 shows the page access with skew.

When comparing the charts from Figure 4.1 and 4.2, it is possible to identify that there are fewer ranges of pages with accesses concentration, as seen by the less dense pattern of dots on the chart. We deduct that the pages which are more accessed in both graphs belong to the Branch table, which is a small table having only 10 records in these tests, and it is never updated but always read by the TPC-B transaction. The apparent decrease in the access

density is a consequence of the increased contention introduced by skewed data accesses. Since most transactions are restricted to a smaller subset of the data, more of the time is spent waiting for locks and latches, which decreases the transaction throughput and, consequently, the density observed in the chart.

Figure 4.1 – TPC-B: Requests to pages without skew

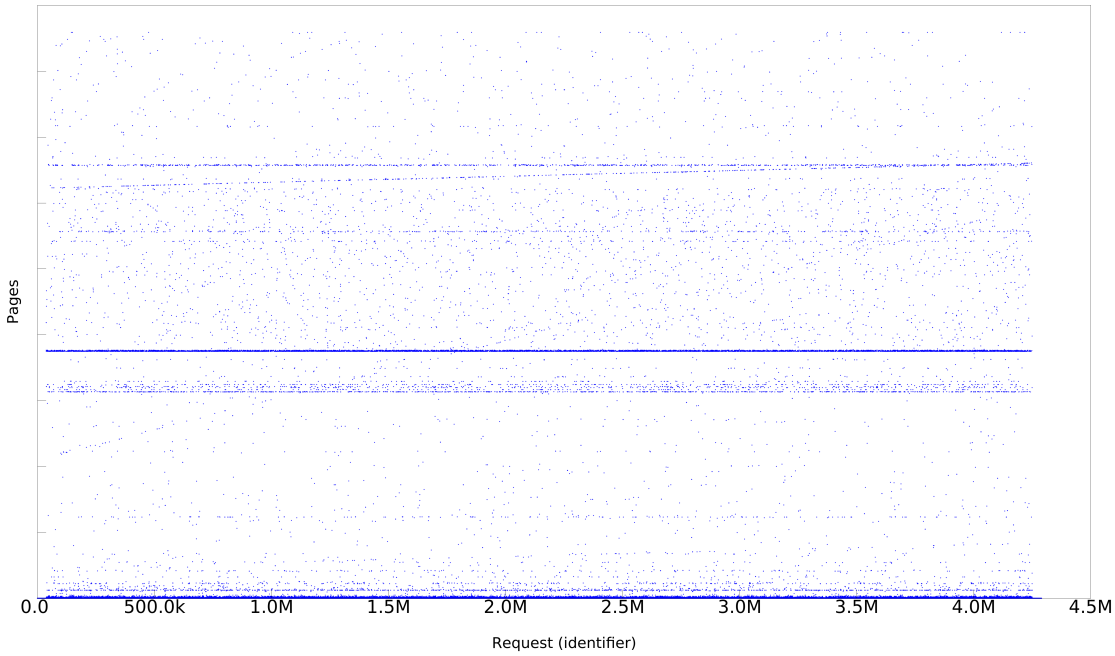
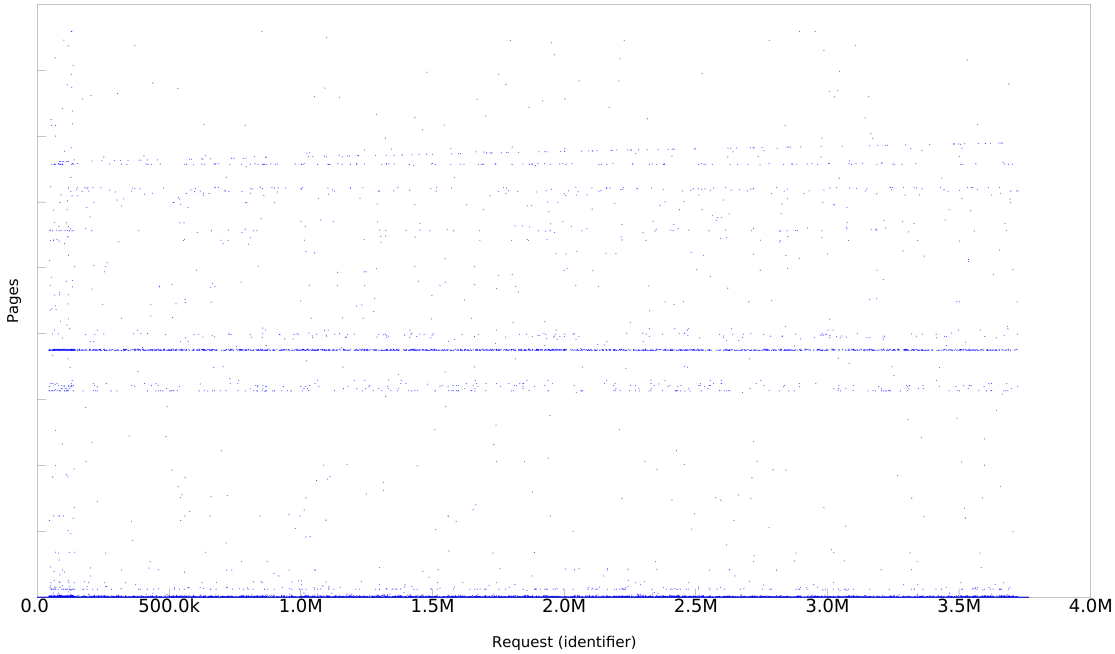


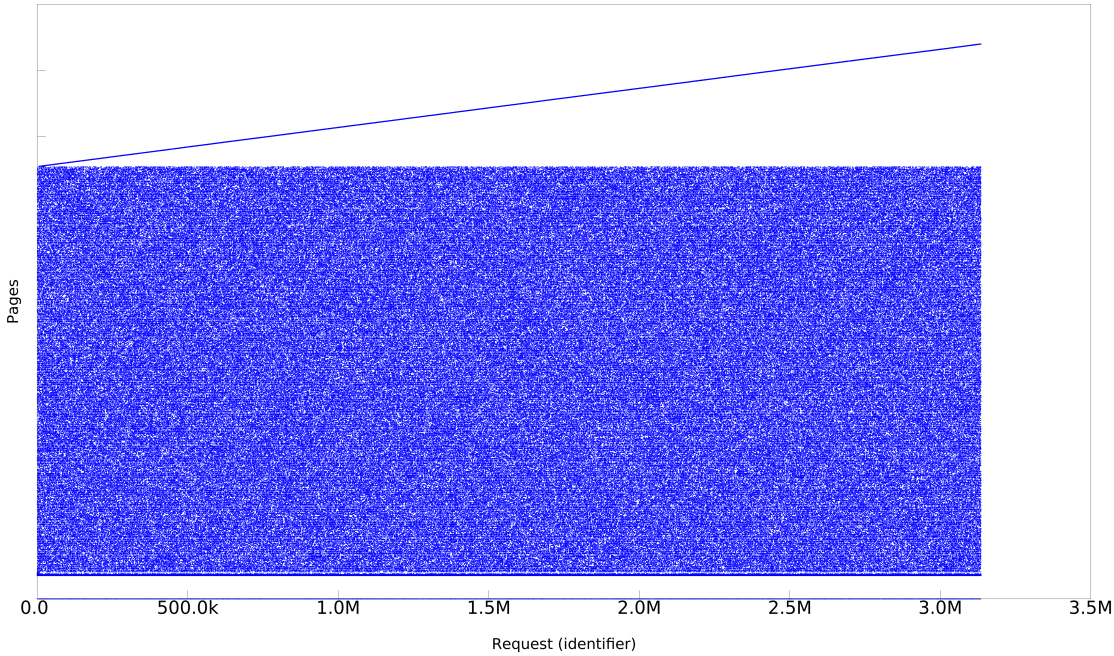
Figure 4.2 – TPC-B: Requests to pages with skew



As there are less read-write accesses than read-only, these update requests are hidden by the read requests. For this reason we create the charts presented in Figure 4.3 and 4.4 to show only read-write accesses, i.e., only pages with at least one write access are displayed. In

these charts, the domain of the y-axis was reduced to show only pages that are accessed. In Figure 4.3, we note two page ranges with interesting behavior. At the top, there is a line showing individual accesses to increasing page identifiers. These pages belong to the History table, which keeps a log of account operations and as such is an append-only table. At the bottom, there is a large range of dense access. We conclude that this range corresponds to the account table, which is the main target of update operations in the only TPC-B transaction.

Figure 4.3 – TPC-B: Read-write accesses without skew



In Figure 4.4 we see the chart for TPC-B generated when the skew command is used. In this chart, we see that the increase of skew is more evident. We observe that around 1/3 of the working set are accessed more densely than the rest, which is proportional to the skew 70-30 which we applied with the command skew. Also note that the density follows a strictly horizontal pattern in the chart, which is due to the fact that the TPC-B transaction does not update any variable-length field, thus maintaining a fixed assignment from record ID to page ID.

Figure 4.4 – TPC-B: Read-write accesses with skew

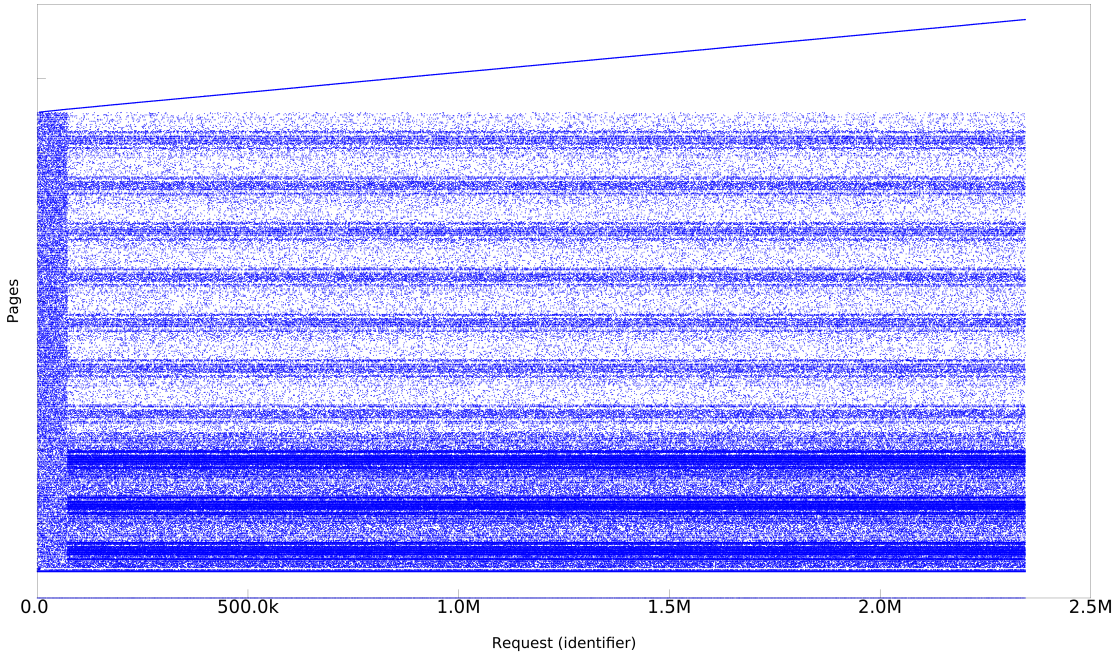


Figure 4.5 presents the distribution of accesses to pages without skew. When we compare the pattern presented in Figure 4.5 with the pattern presented in Figure 4.6, we see that it remains basically the same, but the bar with the 1% of the most accessed pages receive almost 1 million more accesses. This shows that the TPC-B benchmark has an inherent skew in terms of general page accesses. It is only when we look at read-write accesses that we observe the clear effect of the skew command.

Figure 4.5 – TPC-B: Histogram of page access without skew

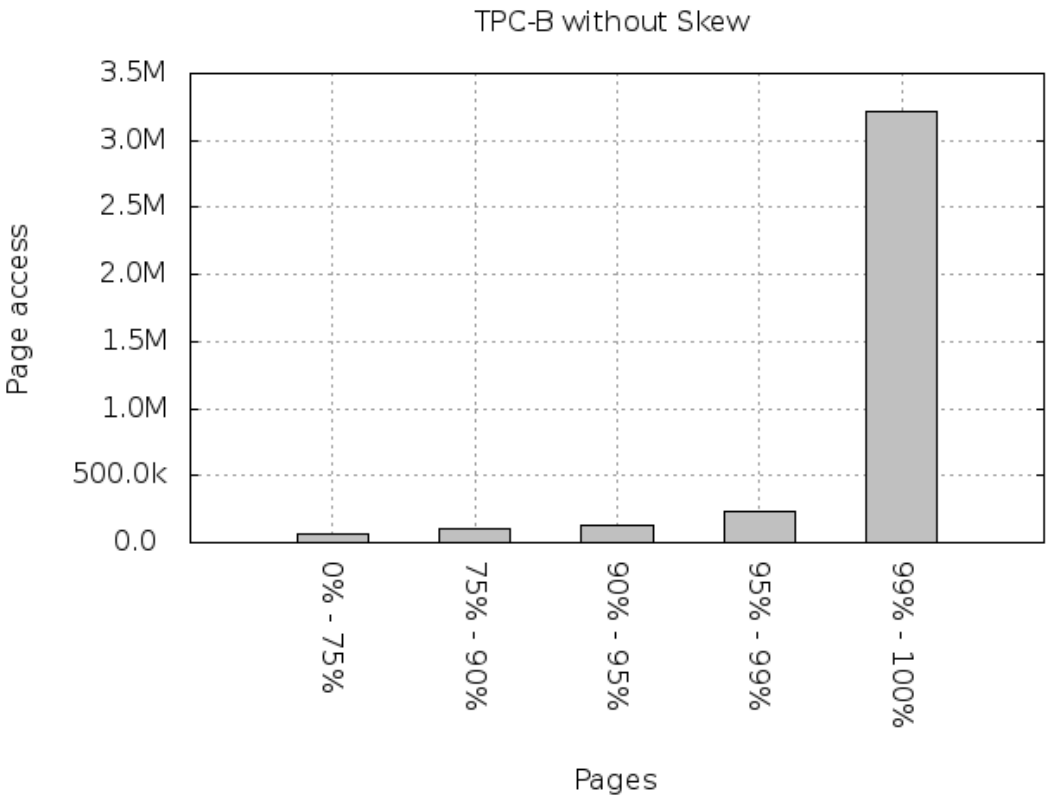
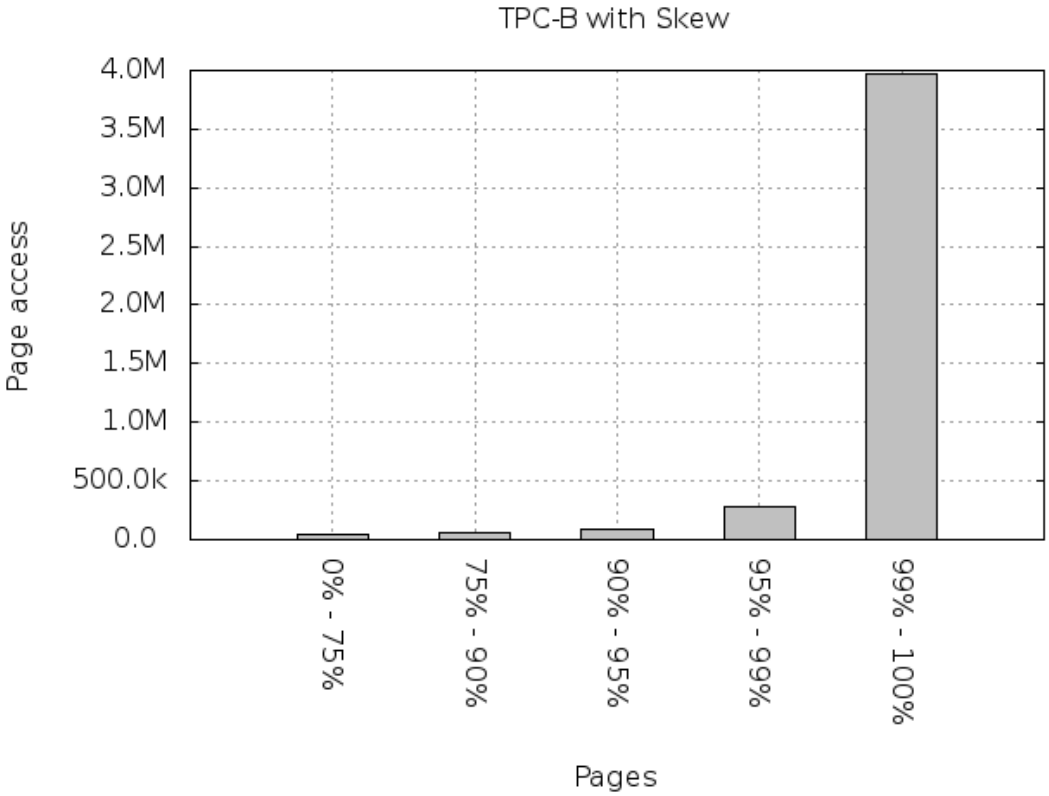


Figure 4.6 – TPC-B: Histogram of page access with skew



When we observe in Figure 4.7 the histogram of read-write accesses, we see that this kind of access is more uniformly distributed, showing almost no degree of skew. However, when skew is applied, as shown in Figure 4.8, we clearly see the effect in the histogram.

Figure 4.7 – TPC-B: Histogram of update page access without skew

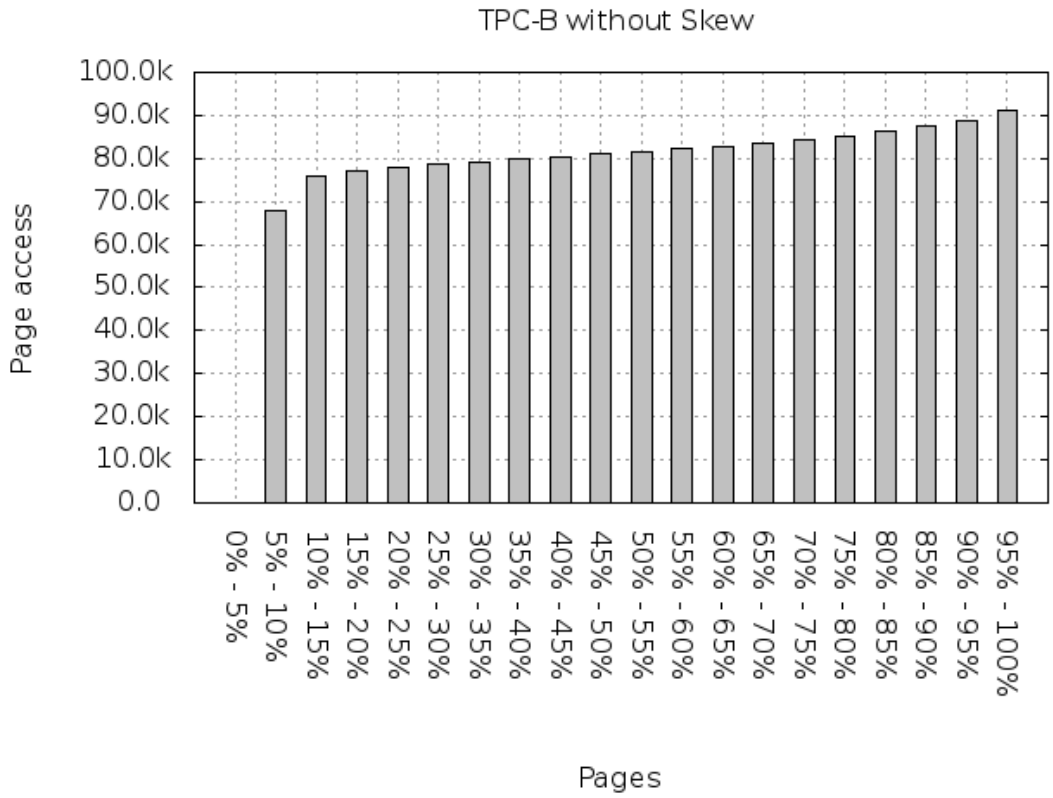
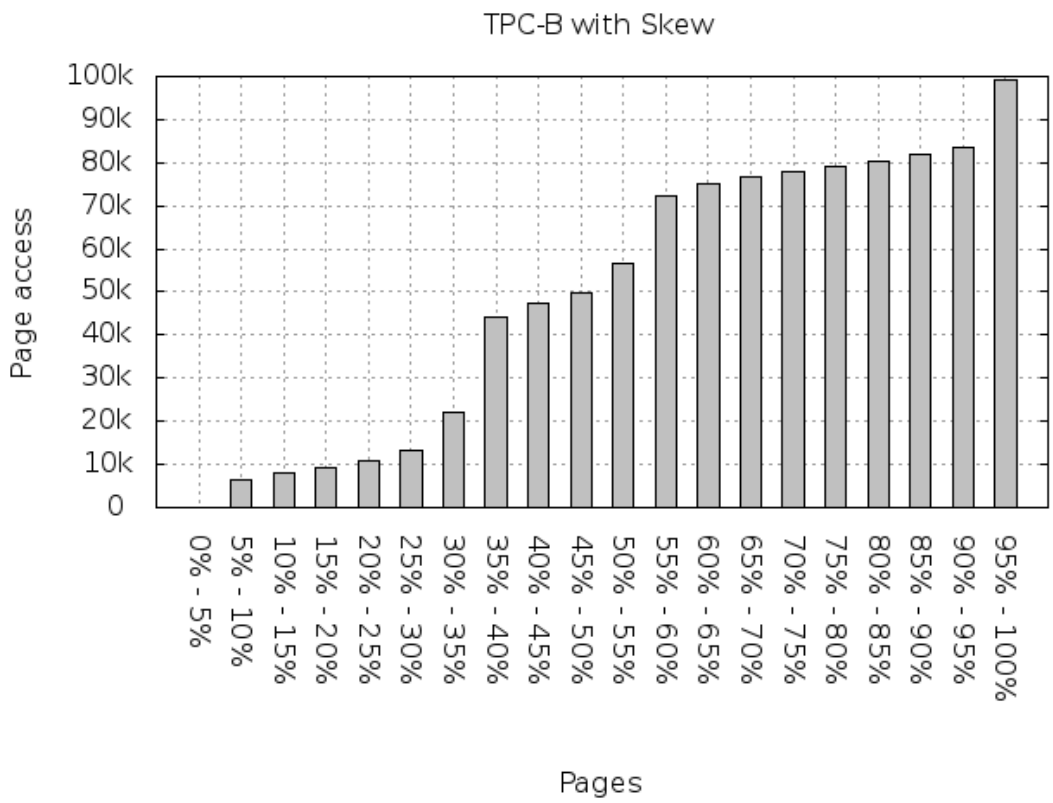


Figure 4.8 – TPC-B: Histogram of update page access with skew



4.2.2 TPC-C

Figure 4.9 shows the page access pattern of TPC-C. In this chart, the requests are divided basic in two ranges: a large range being highly accessed and a smaller range located in the bottom of the chart which is less frequently accessed. The range with few accesses could refer to the Item table. This table is used in 4 of the 5 TPC-C transactions, but these transactions read only a few lines from this table each time they are executed, which can explain so few accesses to this range of pages.

The chart of the page access pattern with skew is showed in Figure 4.10. When comparing this chart with the one presented in Figure 4.11, we see that both charts are very similar and the major difference between them is in the pages located at the top of the charts, which are less accessed in the chart generated with skew. We deduce these pages refer to the Stock table, which is used in only two transactions, but one of them is the heaviest transaction from TPC-C, which verifies the complete stock from a warehouse district. The other pages may not be so affected when running with skew due the business model of TPC-C. As it represents an application for a wholesale supplier with distributed sales districts, a large part

of TPC-C dataset is associated with more than one warehouse, which makes the subsets of data have associations with each other, and for this reason the dataset is not so susceptive to the change of skew.

Figure 4.9 – TPC-C: Requests to pages without skew

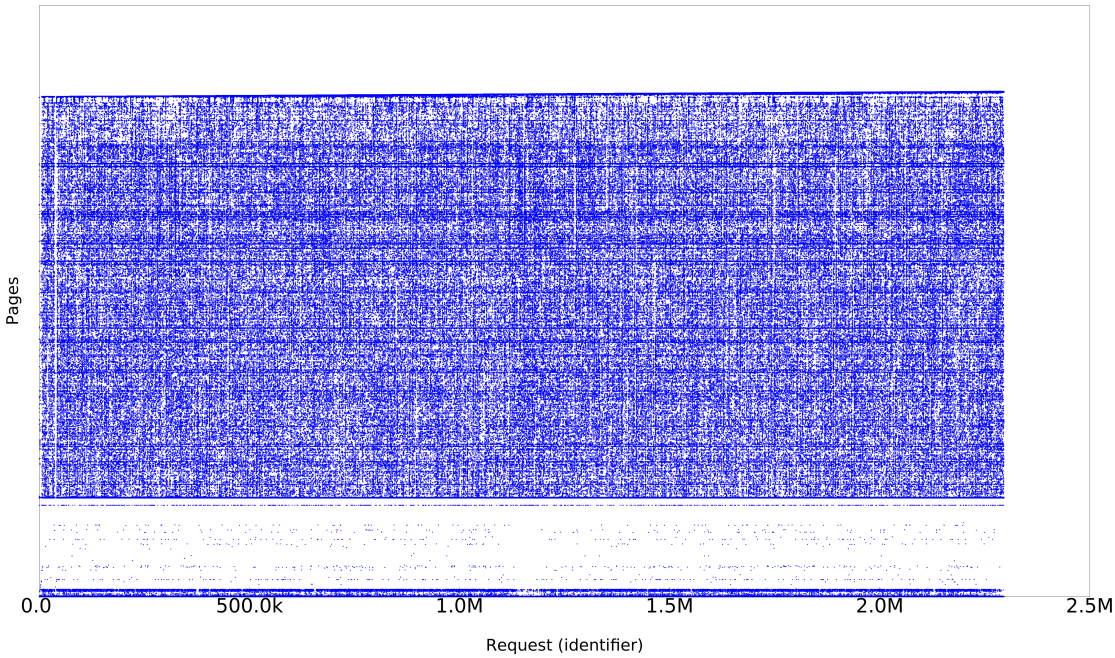
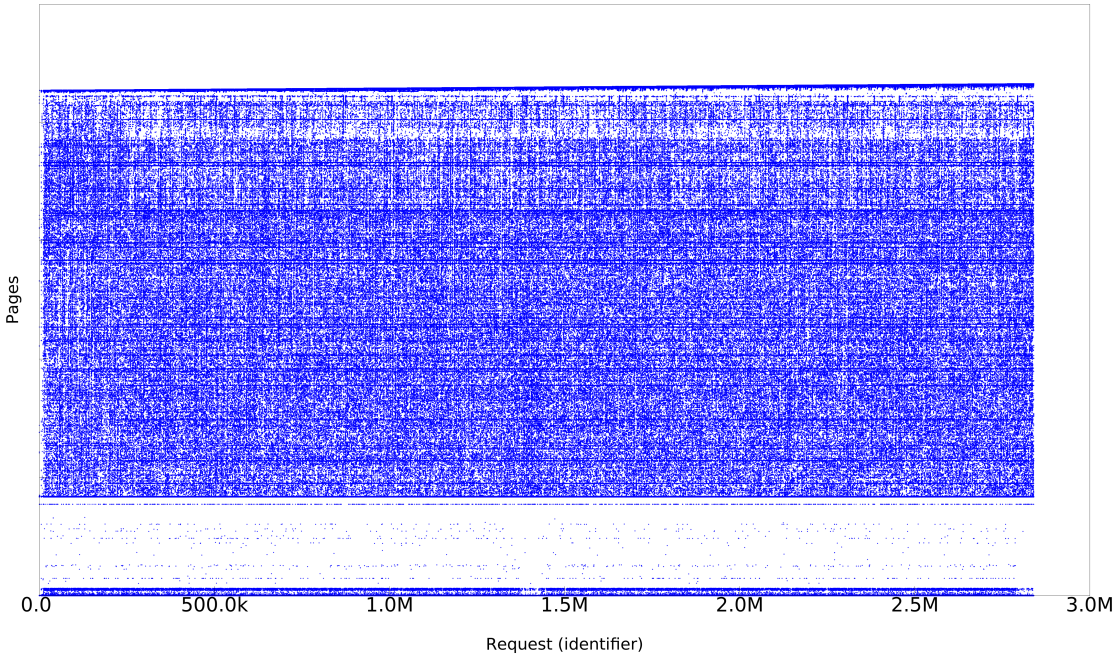


Figure 4.10 – TPC-C: Requests to pages with skew



When we analyze only read-write accesses, showed in Figure 4.11, we see that new pages are created, however not as clear as in TPC-B. We also see that, different from TPC-B, these new pages remain being accessed by transactions and that the accesses to update pages is evenly spread over the complete domain of page IDs.

The chart in Figure 4.12 shows the accesses to write accesses to pages after use the command skew. If we compare the charts from Figure 4.11 with the chart from Figure 4.12 we do not see a different pattern that could be produced with skew. We deduced that this behavior also occur due the business model which TPC-C emulates.

Figure 4.11 – TPC-C: Read-write accesses without skew

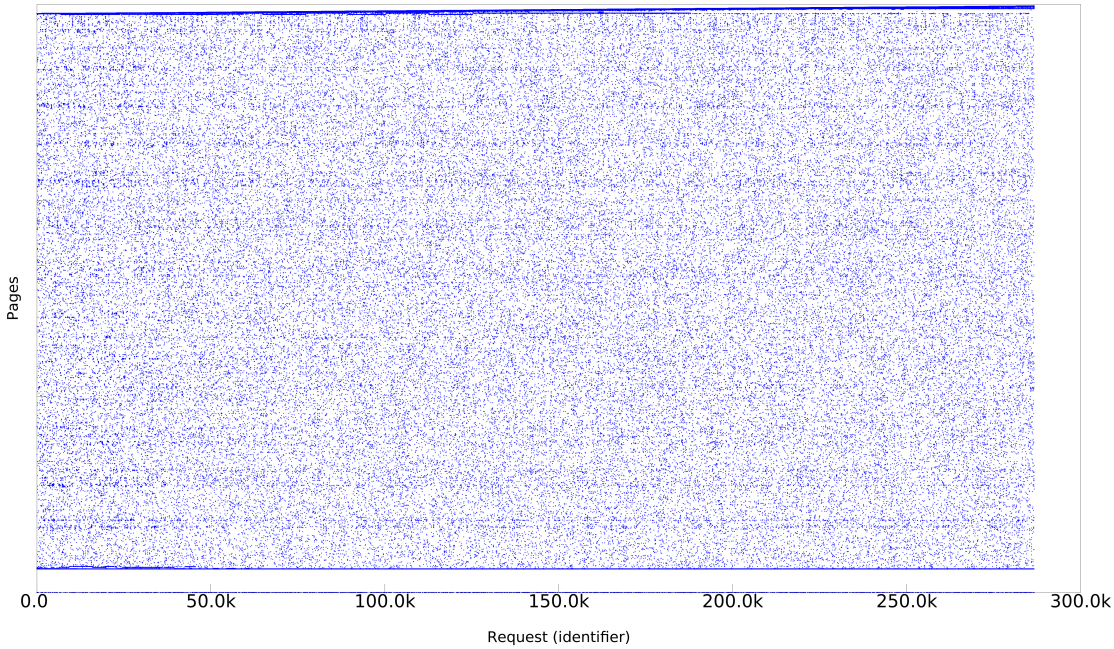
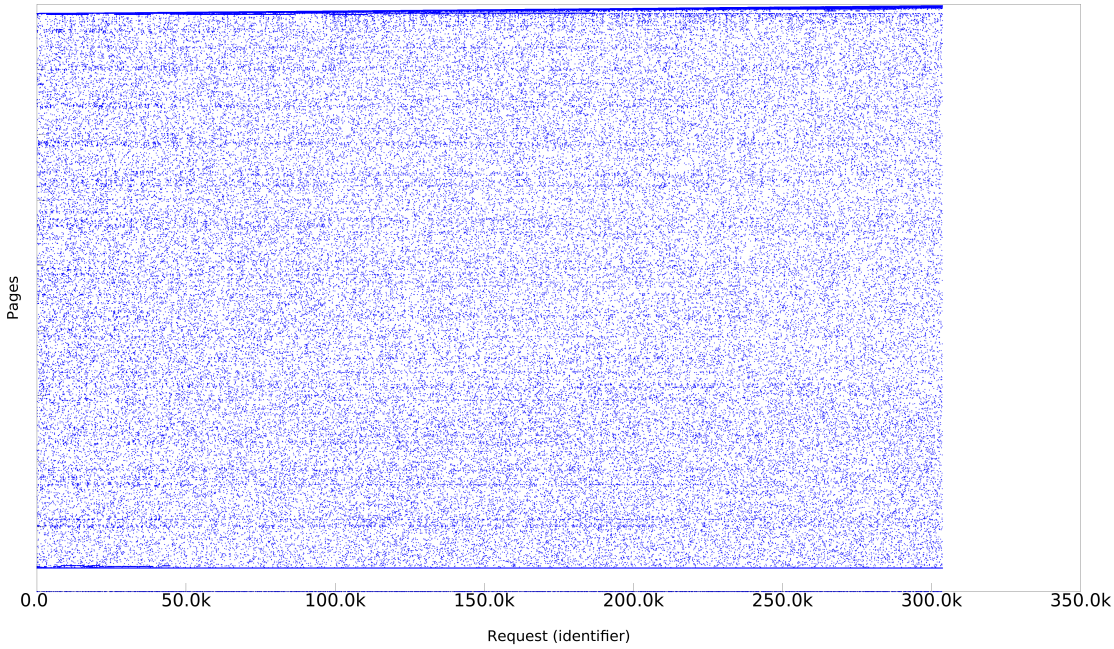


Figure 4.12 – TPC-C: Read-write accesses with skew



By analyzing the page accesses using the histogram chart presented in Figure 4.13, we see that TPC-C has a high degree of skew in pages even without any skew introduced. We see

that most of the pages receive few accesses, as we have notice in Figure 4.9, and only around 20% of the pages are highly accessed. Summing up all the accesses from the first 80% of pages result in less accesses than the range between 80% and 85%. We also notice that the 1% of pages more accessed receives 1.5 million less access when the benchmark is executed with skew, because skew causes more concurrency and longer transactions. However, even with this difference, the pattern of accesses to pages remains the same.

Figure 4.13 – TPC-C: Histogram of page access without skew

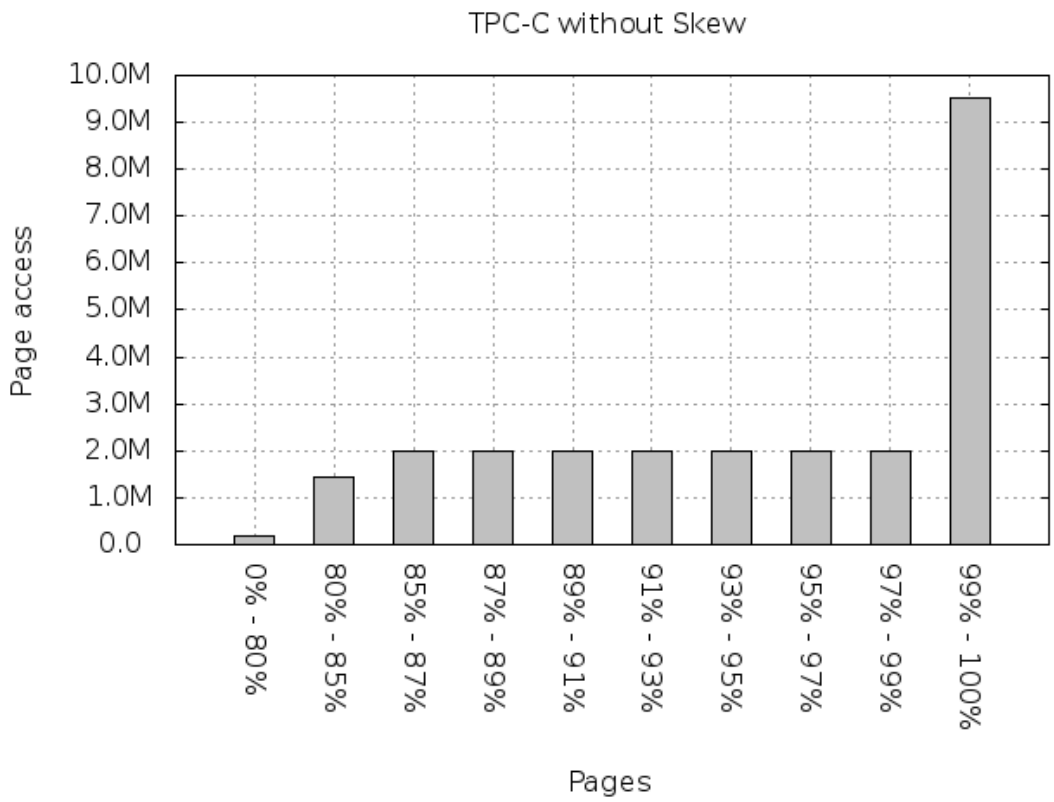
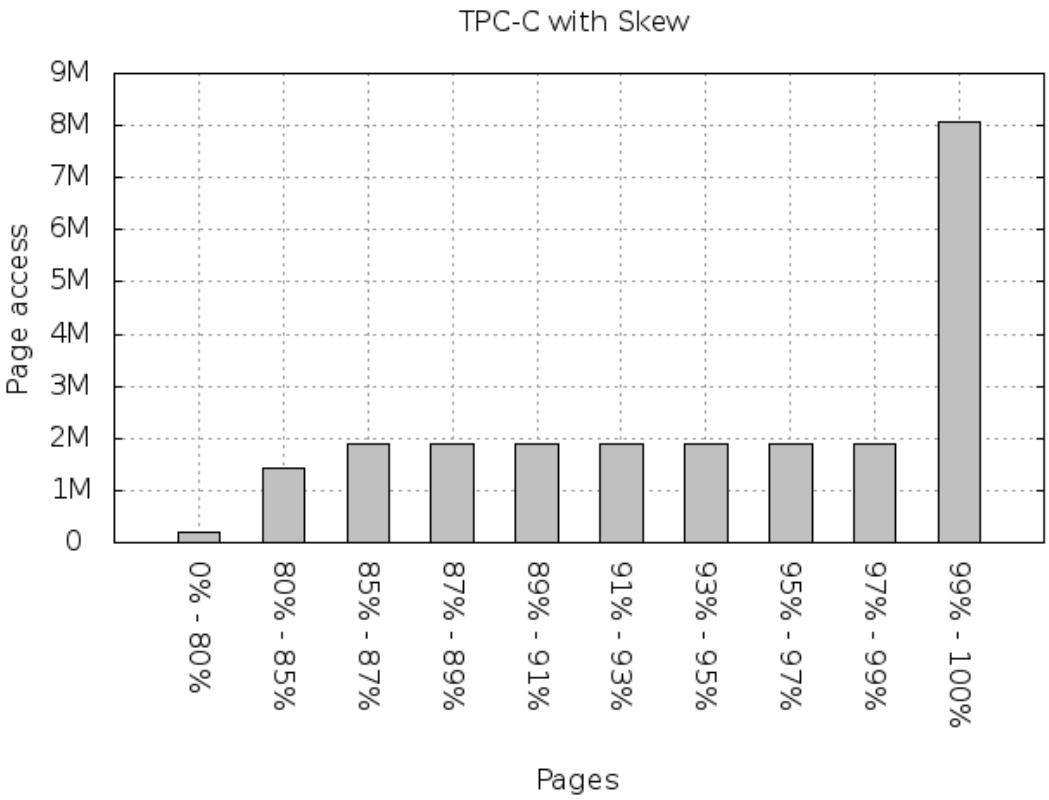


Figure 4.14 – TPC-C: Histogram of page access with skew



4.2.3 TATP

TATP exhibits intensive access to pages, as we can see in Figure 4.15. When we run the TATP benchmark with skew, we notice in the chart of Figure 4.16 that the pattern basically remains the same. It is possible to see a pattern of less dense access in the bottom of Figure 4.16, but it is a small range of pages, which does not have a high influence in page access skew, as we can see in Figures 4.17 and 4.18. Comparing both charts we see that the 10% of pages most accessed remains being much more accessed than the others, having a pattern of skew very similar.

By seeing the similarity of the patterns with and without skew, we wonder why there is no major difference between the charts. However, when we check the scaling rules for TATP, we see that the scale unit is the number of subscribers, which in our case is 10 thousand. Each subscriber is associated with around 2,5 records in the Access_Info table, 2,5 records in the Special_Facility table, and 3,75 records in the Call_Forwarding table. So, even with skew restricting 70% of the accesses to 30% of the subscriber table, the accesses to the

associated tables, which are greater in cardinality, likely remains fairly uniformly distributed, thus leading to no observable difference in our experiments.

Figure 4.15 – TATP: Requests to pages without skew

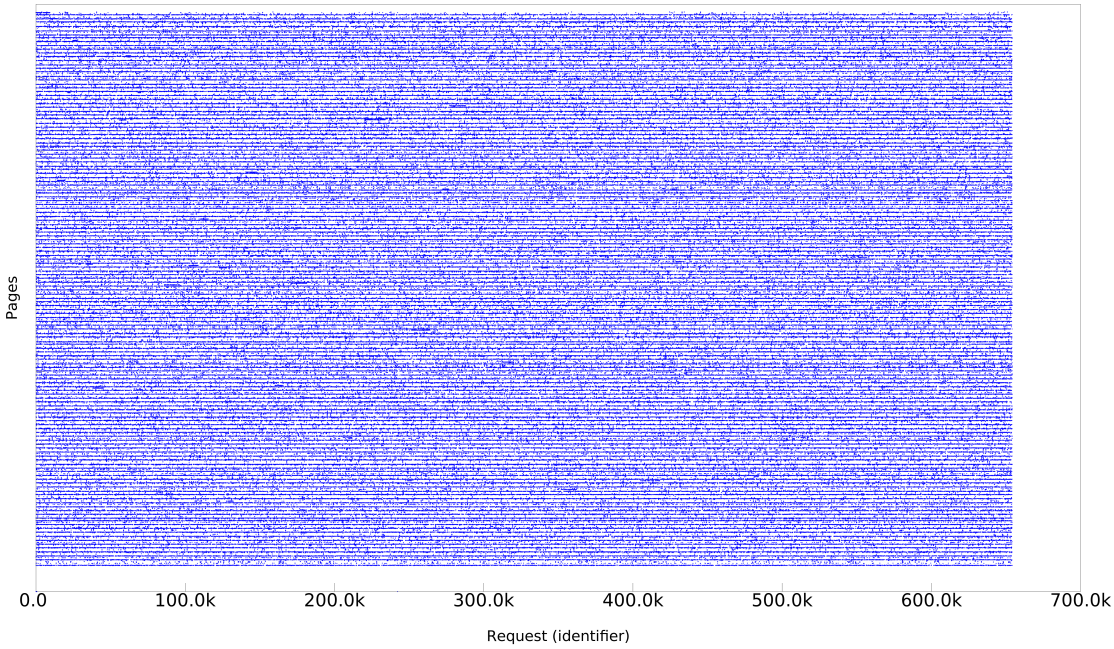


Figure 4.16 – TATP: Requests to pages with skew

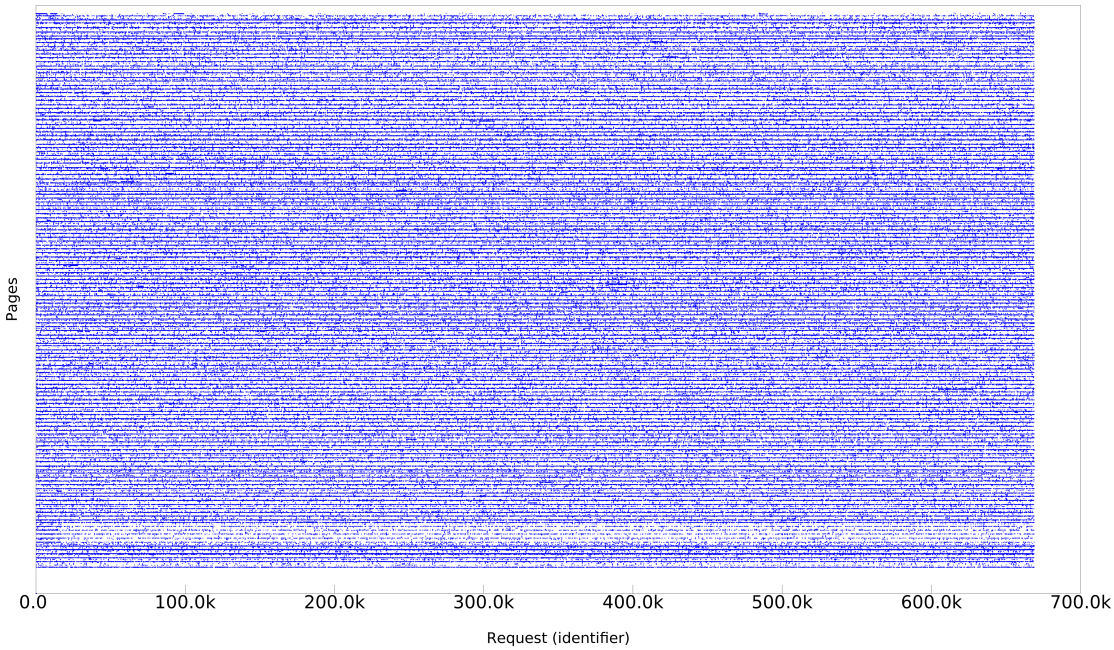


Figure 4.17 – TATP: Histogram of page access without skew

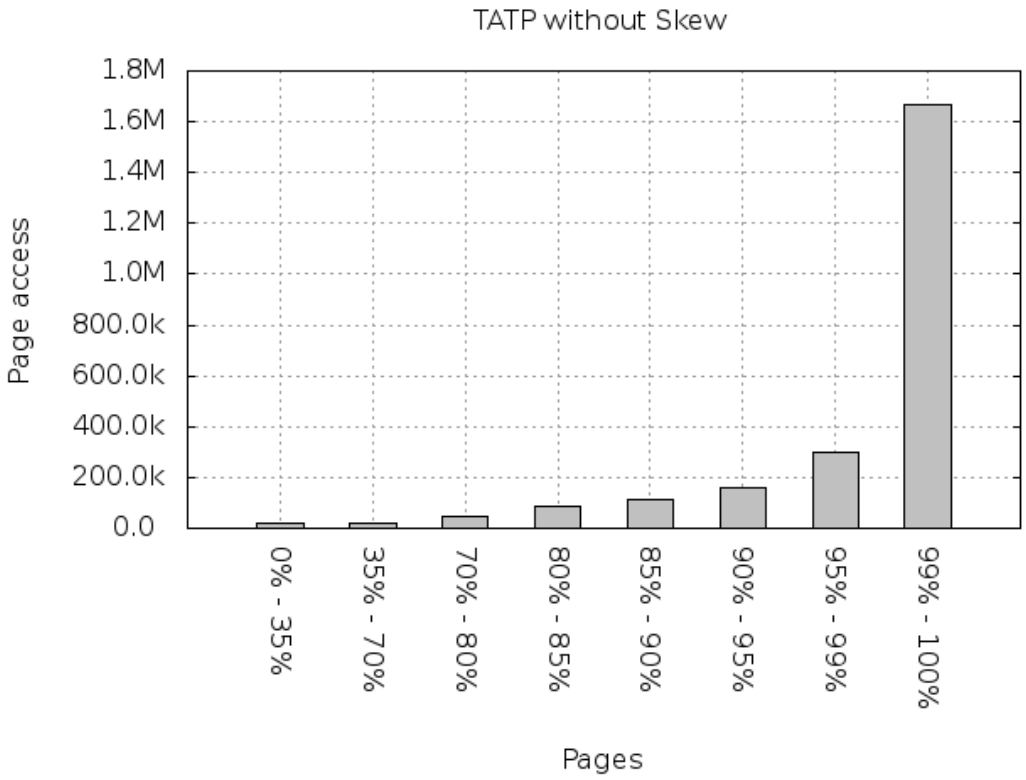
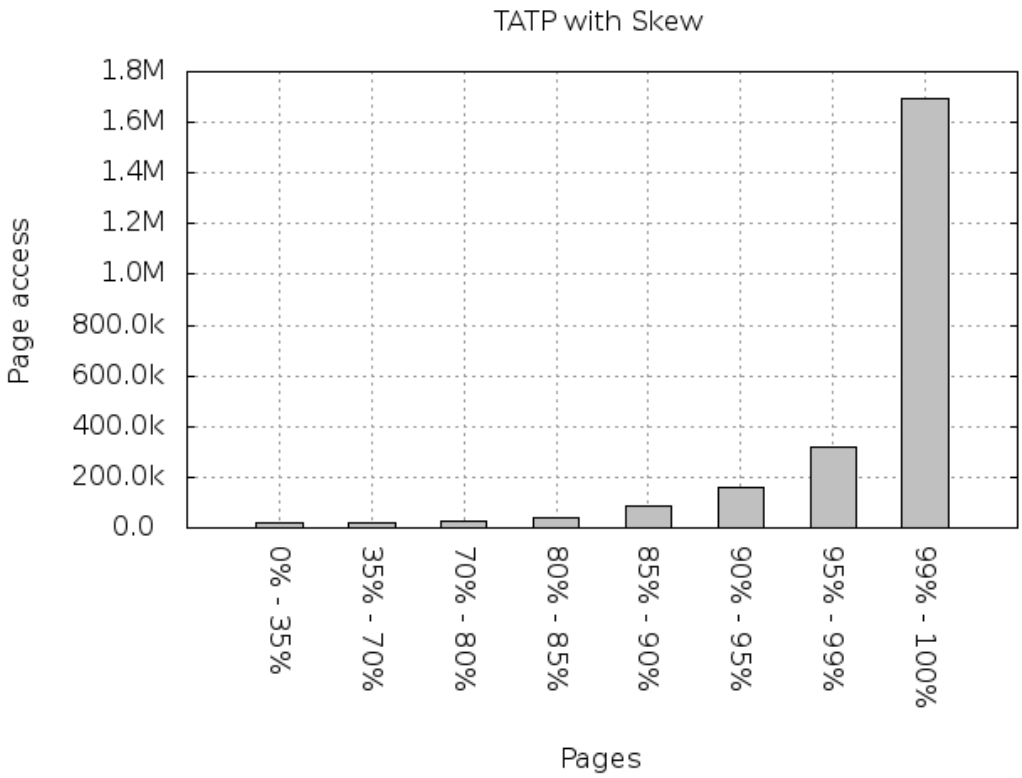


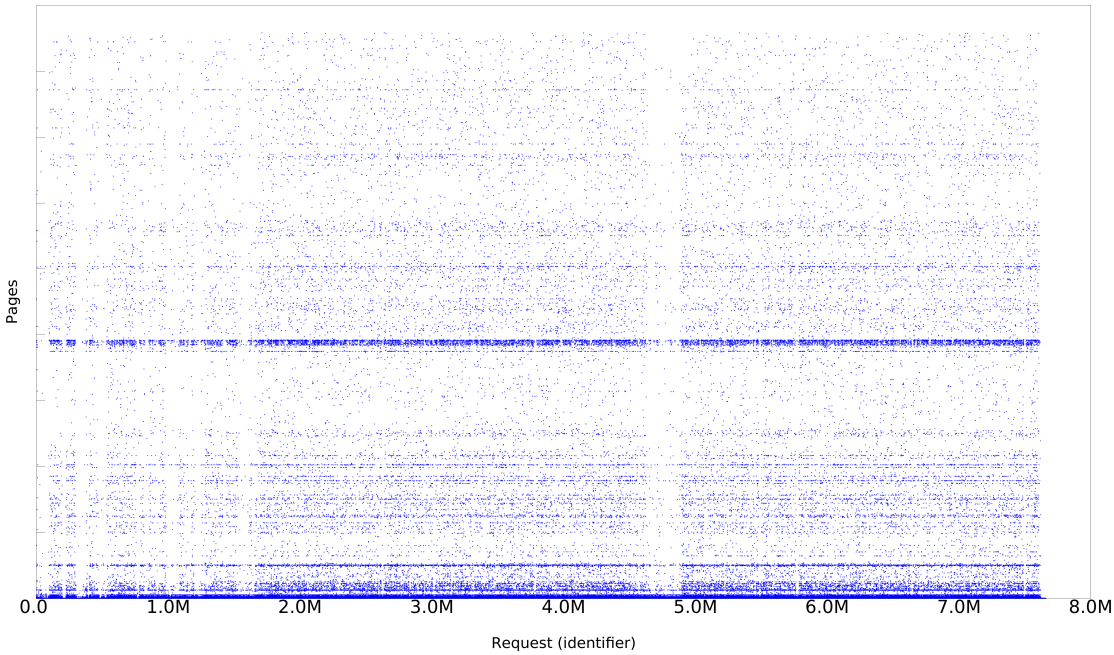
Figure 4.18 – TATP: Histogram of page access with skew



4.2.4 TPC-E

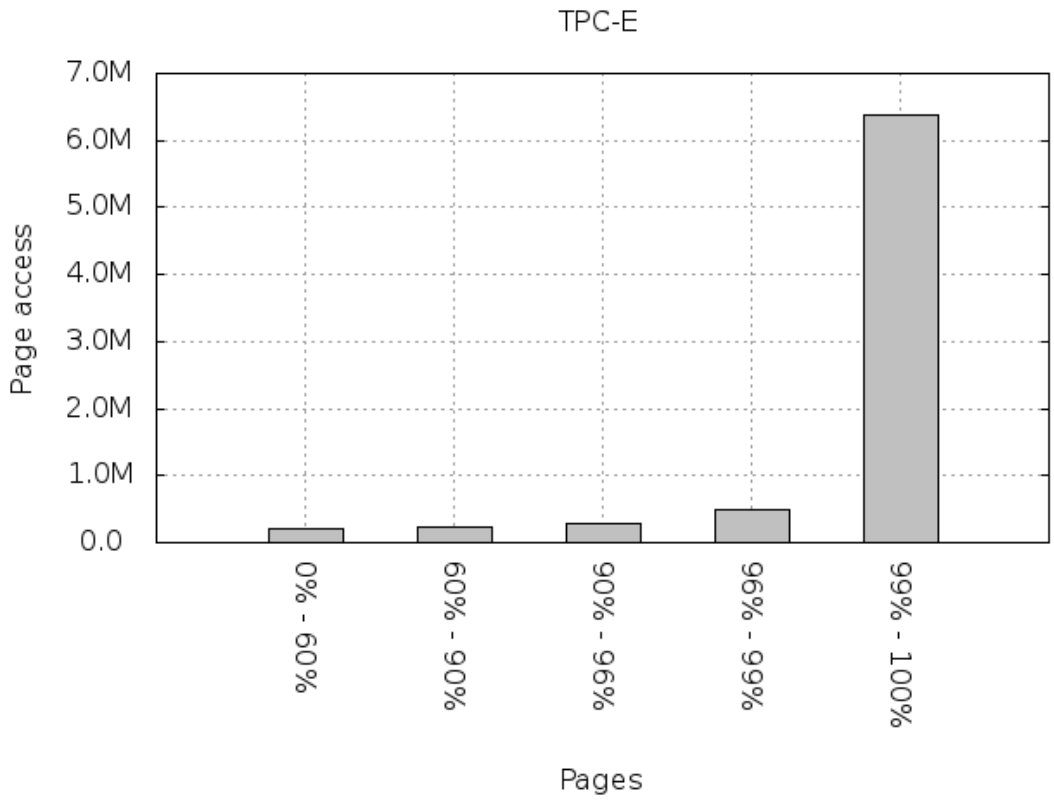
TPC-E exhibits a much more uniform access pattern in terms of page ID ranges being accessed, but with a less even distribution as seen in Figure 4.19. This is due to the factor that TPC-E database design is more complex than other benchmarks, and its transactions perform more full scans and joins between different tables than transactions from other benchmarks.

Figure 4.19 – TPC-E: Requests to pages



Although TPC-E regularly accesses most of the page ID range, we can see in Figure 4.19 that there are small ranges of pages with a high concentration of accesses, which is evidence of high skew. This skew is even more evident when we observe the histogram of page access in Figure 4.20. In [6] it was described that TPC-E presents a data access skew bigger than TPC-B and TPC-C and our experiments confirms this expectation.

Figure 4.20 – TPC-E: Histogram of page access



4.2.5 TPC-H

In Figure 4.21 we can see that TPC-H shows, as expected from an OLAP benchmark, intensive accesses to database pages and that all pages keep being intensively accessed during all benchmark execution. We can see that there is a range of pages in the bottle of the graph which are more accessed than others, however, as showed in the histogram of Figure 4.22, the difference between the less accessed pages and the most accessed pages is not so huge as in the other benchmarks. In this histogram, we also see that TPC-H shows a low degree skew comparing to the other benchmarks, which happens because TPC-H queries access a large amount of data each time they are executed. Also, the database data remains the same over all execution since there is no data being created or updated by this benchmark.

Analyzing the page access pattern, in Figure 4.1, we see that there is a big portion of pages which seems to refer the same table, since patterns similar to transversal lines are showed in the graph as if the queries are reading information from these pages in sequence, forming a “V” pattern. When we try to associate these pages to a database table, the most frequent table is LineItem, since it is the biggest table from TPC-H and it is used in 17 out of the 22 queries from TPC-H. Some of these queries read all data from LineItem table in

sequence in a process called scan. This sequential reading could explain such patterns, which shows these accesses to the pages being executed following an order. Also, the explanation for no other queries access these pages while they are being accessed sequentially can be attributed to the fact that the scan performed by the queries is protected (i.e. Shore-Kits uses guards in the pointers that iterates over the table) and other scans can not perform the same operation at the same time. In this assumption the queries are divided in two subsets: queries that perform a scan over LiteItem table and cannot execute at the same time and the other queries that accesses the other tables. By checking the begin and end of each transaction we identified that there are 5 queries which cannot execute together, and them all execute this scan in the LineItem table, which strengthen this assumption. If it is correct, i. e. these pages refer to the LineItem table, the pages located in the inferior part of the graph refer to the other 7 TPC-H tables, which explains why this portion of data is so heavily accessed.

In order to achieve an analysis of this pattern with fewer assumptions, we could execute isolated queries, identify each query behavior, and which pages refers to each table. However, due to lack of time, we could not perform such analysis.

Figure 4.21 – TPC-H: Requests to pages

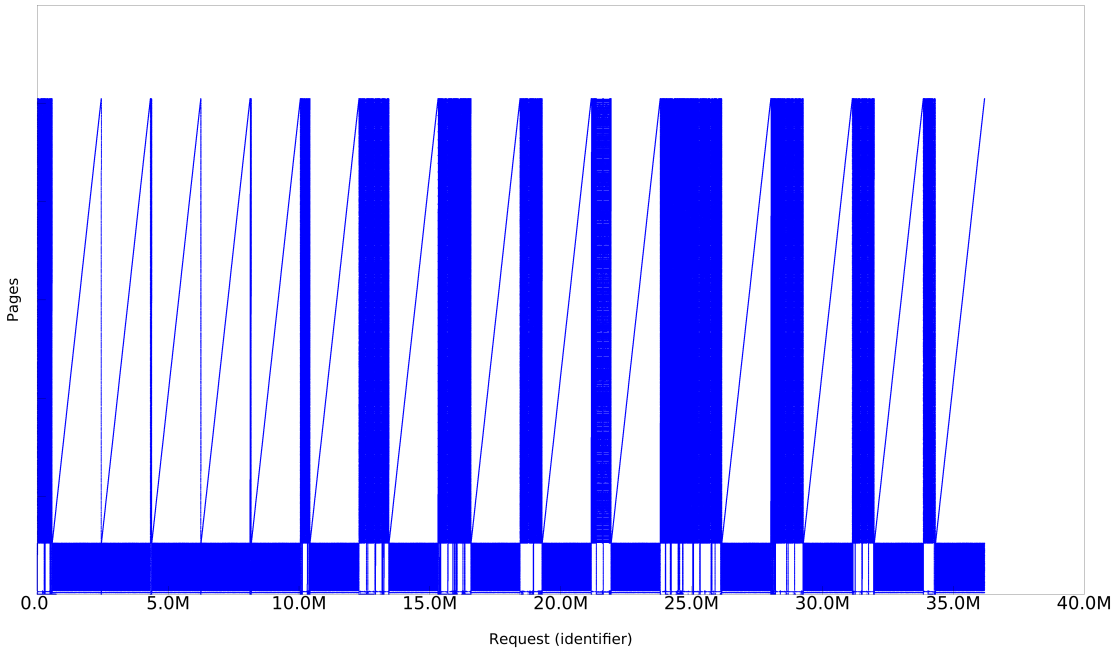
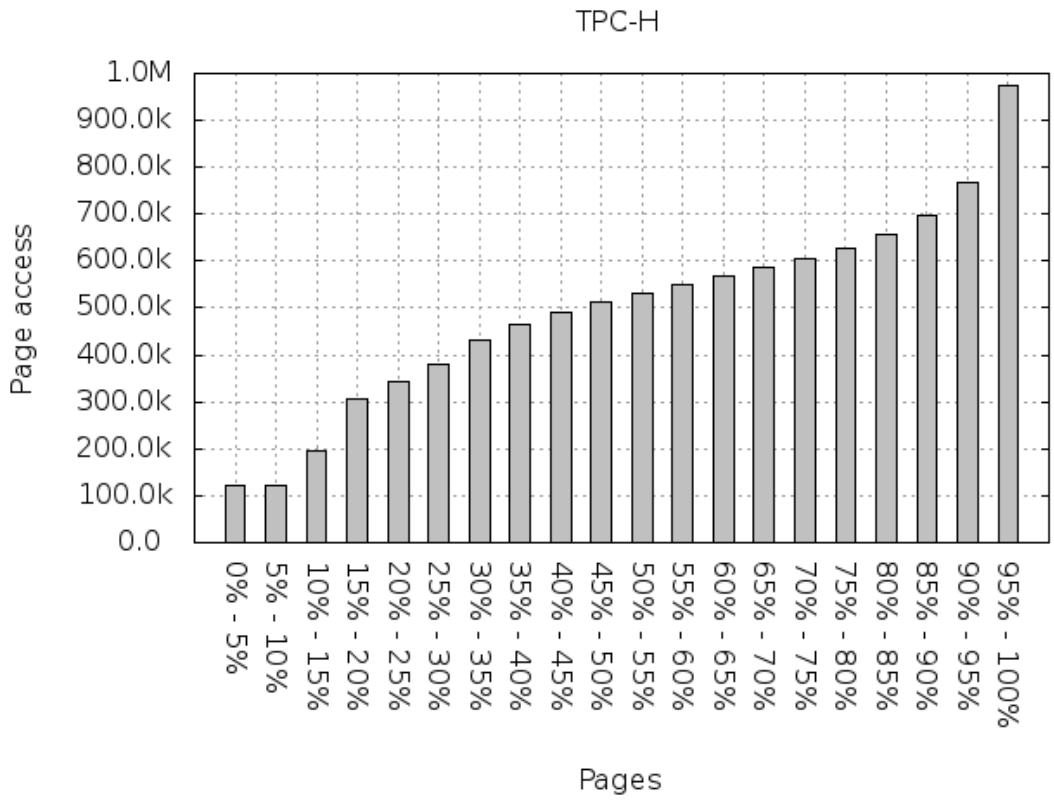


Figure 4.22 – TPC-H: Histogram of page access



4.2.6 Comparing skew among benchmarks

Figure 4.23 shows the distribution of page access per page represented in a boxplot graph created for each of the executed benchmarks. The y-axis represents the domain of pages and the boxplot shows how the page access is distributed in each benchmark. The graphs were created in scalar perspective to facilitate the comparison. In order to generate those graphs, pages were sorted by the amount of accesses to them, then we verified the points in the range of tables where summed up 25%, 50% and 75% of the accesses. These 3 points form the box. We call the line in the bottom of the box as the 25th percentile and the line in the top of the box as the 75th percentile. The blue line inside the box represents 50% of all accesses and is called as median. An evenly access distribution over the pages would have the median exactly in the center of the graph, i.e., 50% of the accesses in 50% of the pages. The closer the median is from the top of the graph, the highest is the skew degree.

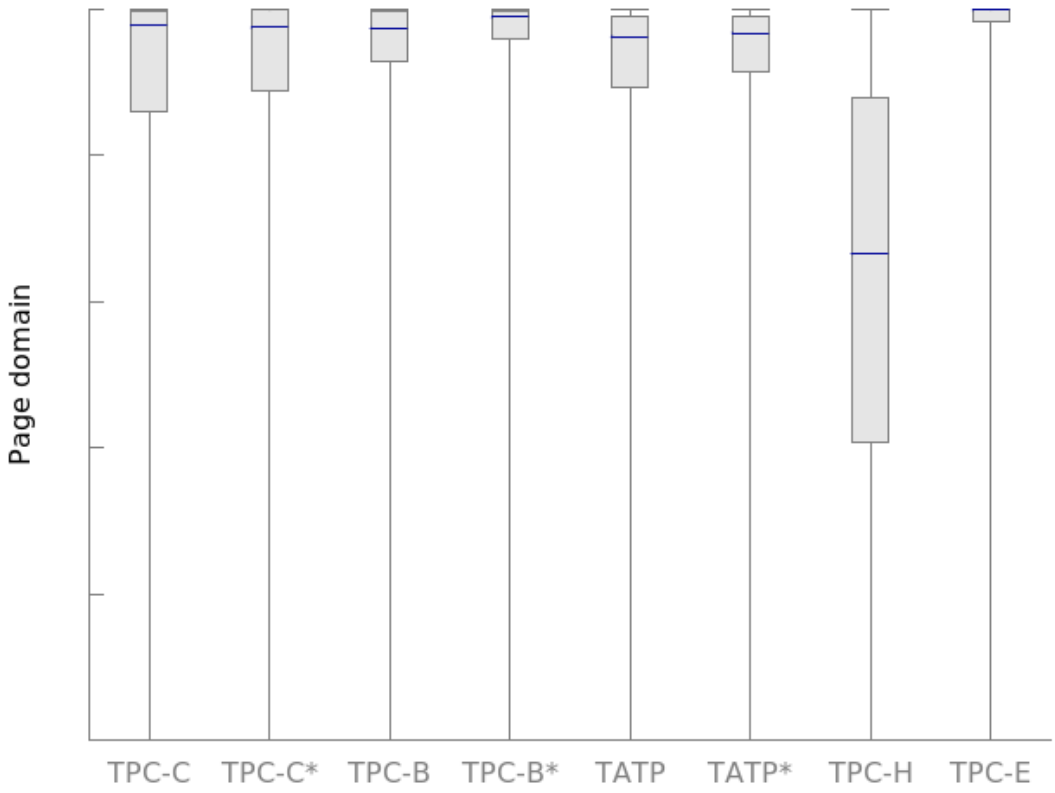
By analyzing the boxplots, it is possible to see that the most uniform distribution is showed by TPC-H. In TPC-H boxplot the median is positioned lower than the others, and it has the biggest box among them all, which means that it has the most uniform distribution

from the 25th percentile to the 75th percentile. This pattern is expected since TPC-H transactions analyzes big amount of data and perform scan on tables, which implies in many pages being highly accessed.

The highest degree of skew is showed in TPC-E boxplots. The median is positioned almost in the top of the graph, which means that 50% of all accesses is located in the few percentage of pages located between the median and the top of the graph. The 75th percentile is so close to the top of the graph that it is not possible to see it.

TPC-B, TPC-C, and TATP without skew have their medians located in similar position, which shows a similar distribution of accesses. When these benchmarks are executed with skew, the box from these 3 benchmarks reduces, which means a higher concentration of accesses in the superior pages as expected, however the median in TPC-C and TATP remains almost in the same position, meanwhile the TPC-B median goes closer to the top of the graph. This behavior was already showed when these benchmarks were individually analyzed. Now, comparing them together, it is possible to see that TPC-B is the benchmark more affected by the change of skew. As already explained it happens due TPC-B database design, which have few big subsets of data and, when skew is applied, most of the accesses can be done in only few of these subsets. The same thing does not happen with TPC-C and TATP. In the first because the existent subsets have a relation among themselves, and in the second because there are many small subsets, which makes the accesses be distributed even with skew.

Figure 4.23 – Boxplots of all benchmarks



* is used to show when the benchmark is executed with skew.

4.3 Counters of transaction events

Table 4.1 shows the amount of transactions created, committed and aborted per benchmark. When we analyze this data, we see that TPC-H creates fewer transactions than any other benchmark. This happens because TPC-H, as already mentioned, contains long-running queries, for this reason, while other benchmarks create thousands or millions of transactions, TPC-H creates only a few hundreds.

In the opposite side is TATP with simple and small transactions. TATP is the benchmark that created the most transactions, summing up more than 3,3 million transactions in 90 seconds. It also has the biggest portion of its transactions aborting. It occurs because most of the data used as input in TATP transactions are randomly created and not checked if correspondent values for these inputs exist in the database.

As occurred in the page access analysis, once more we see TPC-B being more affected by changes of skew than other benchmarks while we see TPC-C and TATP suffering small effects of this variation. It occurs because transactions in TPC-B with skew need to wait more

to acquire locks, as described in section 4.5, and, for this reason, they become longer than without skew, which results in less transactions in the same amount of time.

Among the OLTP benchmarks, we see that TPC-C is the benchmark that creates fewer transactions, even fewer than TPC-E, which is a benchmark that contains OLAP and OLTP transactions. It occurs because among the 10 TPC-E transactions there are transactions simpler and smaller than any TPC-C transactions, which enables TPC-E to create more transactions than TPC-C.

The relation between the number of transactions created and the transaction complexity is evident in Table 4.1. The simpler and shorter the transactions from a benchmark are, the more transactions this benchmark creates, and vice-versa.

Table 4.1 – Transaction counters

	TPC-B	TPC-B*	TPC-C	TPC-C*	TATP	TATP*	TPC-E	TPC-H
Begin	740,435	537,485	15,070	16,060	3,343,445	3,367,025	180,725	332
Commit	669,469	441,703	14,913	15,649	2,505,575	2,519,708	179,456	332
Abort	70,966	95,782	157	411	837,870	847,317	1269	0

* is used to show when the benchmark is executed with skew.

4.4 Counters of buffer manager events

When the benchmark is running in the DBMS the component Buffer Manager is responsible to perform a fix operation, which fetches pages from the disk and, when necessary, latch them in exclusive or shared mode. These pages, which are in memory, can be read or modified by transactions. When a transaction modifies a page in memory, it makes this page to have a different version in memory from the one existent in disk, and this page is then considered as a dirty page. So how these pages can have this modified version in the disk and no longer be considered as dirty? There is a process called cleaner, which reads all pages from memory and write the dirty pages to the disk. If two or more pages in the memory can be written in sequence to the disk, the cleaner process can write these pages in one access to the disk, otherwise it needs to perform two or more accesses, what increase the cost to write these pages. Table 4.2 presents the counters for each workload regarding the process to write dirty pages in disk, if it is done in one or more accesses. It also presents the Buffer Manager fix and latch operations, which show how each benchmark stresses this components.

By verifying Table 4.2 we see that TPC-E is the benchmark that stress the buffer manager the most. It occurs due its transactions complexity, which turns out that TPC-E's transactions work with data from many different tables when executing. Besides, the operations performed by TPC-E transactions involve a large amount of data, which leads to the largest number of page fixes among all studied benchmarks.

TPC-B, despite being an intensive writer benchmark, it latches pages as exclusive in 22% of the time, which is not much more than TPC-C, which latches pages 19% of the time in exclusive mode.

Once more, it is possible to see that TPC-B is the benchmark more sensible to the change of skew. The number of page fix operations performed by TPC-B reduces in 28% with skew and, because of that, we can see how the number of latches acquired decreases as well. Meanwhile, TPC-C shows a difference smaller than 5% after change the skew, and TATP around 1%. When we project these numbers for hours of execution the accumulated effect is significant in terms of transaction throughput. However, it is clear that TPC-B is the benchmark more affected.

We could not explain the exclusive latches observed in TPC-H, since there is no operation to update a page and no log record of an update operation is created. We saw that, in the process of scan a table used by TPC-H, some locks of objects are acquired in exclusive mode, and for this reason the page where these objects are located must be latched as exclusive. However, further code analysis is necessary to understand the reason for the implementation that results in this behavior.

When we see the amount of page fix performed by the buffer manager while TPC-H was running we observe that the buffer manager performed less than half of the amount of page fixes than in TPC-E. This difference can be attributed to the fact that TPC-H, even having only OLAP queries accessing big amounts of data, in most of its queries the read data is reused in further operations without fetching pages from disk. Meanwhile, TPC-E, due its complexity and random access pattern, causes more IO.

When we analyze the write behavior from buffer to disk, we see that most of the page writes performed by TPC-B are operations to write pages in sequence. We can attribute this fact to TPC-B database design, which has a table continuously growing. Since as this table grows it needs to allocate more pages and these new pages can be sequentially written to disk.

While TPC-B uses more sequential write operations, the benchmarks TATP and TPC-C make more use of one-page write operations, which shows that the write behavior is random.

TPC-H and TPC-E do not show any page write operation. The first because it does not have any write operation to perform and the second because it was implemented in Shore-Kits by porting an existent implementation from other database, which makes its structure different from the others benchmarks and the counters not properly incremented in TPC-E benchmark.

Table 4.2 – Counters of buffer manager

	TPC-B	TPC-B*	TPC-C	TPC-C*	TATP	TATP*	TPC-E	TPC-H
Latch EX	3,925,332	2,894,104	565,760	591,662	665,467	670,354	2,812,609	3,342,302
Latch SH	13,506,145	9,707,092	2,372,768	2,410,308	22,150,276	22,313,985	116,919,258	47,466,353
Page fix	17,431,477	12,601,196	2,937,581	3,001,139	22,815,743	22,984,339	119,731,867	50,808,655
One page write	23	97	18,698	18,076	34,844	33,982	0	0
More than one page write	3,872	2,137	12,721	12,820	20,333	19,089	0	0

* is used to show when the benchmark is executed with skew.

4.5 Counters of locking events

Analyzing Table 4.3 we verify how benchmarks stress the locking manager and which the most locked objects are in each benchmark.

The first effect we can notice is the difference between TPC-H and TATP regarding volume locking. TATP needs to lock the volume more often than TPC-H because TATP creates millions of transactions while TPC-H creates only a few hundreds and, as described in Section 2.7.3, when a transaction locks an object it must lock the other objects in the hierarchy in intention mode, and volume is the highest hierarchy of the objects.

We also notice that TPC-C, TPC-E, and TPC-H have records as the main object locked. The use of objects smaller than a page is how most of the databases for current systems are designed. TPC-B and TATP do not follow this line and have Store as the most locked object, since they need to access different objects in different stores and, because of the hierarchical lock concept, the store lock must be acquired.

For the lock manager, as occurred for the buffer manager, we see that TPC-E is the benchmark that mostly stresses this component. It acquires more than 30 million records locks and 8.8 million page locks.

Regarding the lock wait counter, we see that TPC-B is the one that suffers more. It occurs because, as TPC-B is an update-intensive benchmark, it create more requests to lock pages as exclusive which are already locked. When it occurs it must wait until the lock is

released, what causes this big number of locks waiting. Also, we notice that in the three benchmarks which we applied skew, the lock wait number increased. It happens because as the benchmark is restricted to a smaller portion of the data, the data to be locked as exclusive has a bigger chance of already be locked by another transaction i.e., high contention. Finally, TPC-H is the only benchmark which shows 0 in this statistic. It is expected since operations in TPC-H do not need to lock any object as exclusive and, therefore, they do not need to wait to acquire any lock.

Table 4.3 – Counters of lock manager

	TPC-B	TPC-B*	TPC-C	TPC-C*	TATP	TATP*	TPC-E	TPC-H
Volume	1,480,865	1,074,965	29,579	31,120	3,893,245	3,920,463	248,257	406
Store	7,361,750	5,251,249	223,340	237,278	8,093,638	8,154,734	3027124	1,935
Page	5,882,038	4,203,293	558,435	583,895	4,492,655	4,516,901	8,844,358	21,904
Key value	7	7	19	19	9	9	25	25
Record	7,219,808	5,059,675	1,031,876	1,060,771	7,738,746	7,794,024	30,114,761	9,814,848
Lock wait	347,666	384,919	3,418	4,774	27	67	11,668	0

* is used to show when the benchmark is executed with skew.

5 CONCLUSION

In this work we investigated low-level characteristics of database systems when the benchmarks TPC-B, TPC-C, TPC-E, TPC-H, and TATP were executed, in order to understand how different applications properties impact the internal behavior of the system. The goal of such investigation was to understand typical application patterns and how they affect individual system components. This helps developers and administrators of database systems to improve performance and find optimal configuration parameters. In order to perform such investigation, we used the storage manager Shore-MT to collect statistics of some internal components and detailed patterns of access to pages of the database.

When we analyzed the page access patterns produced by the benchmarks, we verified that TPC-E is the benchmark with the highest degree of page access skew, and that TPC-H is the benchmark with most uniform access. Both behaviors are expected. The first because TPC-E already presents high degree of skew in the logical layer, and the second because the OLAP transactions from TPC-H contain mostly full scans, which implies in the creation of few transactions that read big portions of the dataset.

Also, by changing the skew in input parameters for TPC-B, TPC-C, and TATP, we noticed that the behavior of system internal components suffered different impact for each benchmark executed. In our qualitative analysis, we tried to identify which workload properties are responsible for such variations. For instance, as TPC-B creates only read-write transactions, when we increase the skew in the benchmark, the database exhibits more contention on locks and latches, which changes the system behavior by hurting the performance. In this instance, the DBMS maintainer may prioritize number of CPU cores over amount of main memory in a hardware upgrade. This is an example of how workload analysis supports human decisions.

Although our study presented important characteristics for the benchmarks analyzed, there are still gaps that further studies could fulfill, such as an analysis of transactions properties and how internal components of the database system behave when the TPC-E and TPC-H benchmarks are executed with skew. Also, it is necessary a deeper analysis of the small differences in the results with and without skew of TPC-C and TATP, since this difference can be caused by the benchmark implementation in Shore-KITS or the benchmark properties. Such future analysis could be performed by studying individual transactions behavior and by mapping pages to tables.

REFERENCES

TPC. TPC benchmark B standard specification, revision 2.0, 1994. Available at http://www.tpc.org/tpcb, accessed on July/2015.

TPC. TPC benchmark C standard specification, revision 5.11, 2010. Available at http://www.tpc.org/tpcc, accessed on July/2015.

TPC. TPC benchmark E standard specification, revision 1.13.0, 2014. Available at http://www.tpc.org/tpce, accessed on July/2015.

TPC. TPC benchmark H standard specification, revision 2.17.1, 2014. Available at http://www.tpc.org/tpch, accessed on July/2015.

Telecommunication Application Transaction Processing (TATP) Benchmark Description, version 1.0, 2009. Available at http://tatpbenchmark.sourceforge.net/index.html.

Tözün P, Pandis I, Kaynak C, Jevdjic D, Ailamaki A, From A to E: Analyzing TPC's OLTP Benchmarks: The obsolete, the ubiquitous, the unexplored, EDBT, 17 28, 2013.

Johnson R, I. Pandis, N. Hardavellas, A. Ailamaki, and B. Falsafi. Shore-MT: a scalable storage manager for the multicore era. EDBT, 24–35, 2009.

Shore-Kits Project Web Page: https://sites.google.com/site/shoremt/shore-kits, accessed on July/2015

Mohan C, Haderle D, Lindsay B, Pirahesh H, Schwarz P, ARIES: a transaction recovery method supporting fine-granularity locking and partial rollback using write-ahead logging, ACM TODS 17(1), 94 162, 1992

Härder, T. and Reuter, A. Principles of transaction oriented database recovery. ACM Computing Surveys 15 (4): 287-317, 1983

Carey M, D. J. DeWitt, M. Franklin, N. Hall, M. McAuliffe, J. Naughton, D. Schuh, M. Solomon, C. K. Tan, O. Tsatalos, S. White, M. Zwilling., "Shoring up persistent applications." Proc. SIGMOD, 1994.

Härder, T., DBMS Architecture – Still an Open Problem, Datenbanksysteme in Business, Technologie und Web, 11. Fachtagung des GI-Fachbereichs "Datenbanken und Informationssysteme" (DBIS), 2 – 28, 2005.

Gray J, Reuter A., Transaction Processing: Concepts and techniques. San Mateo, CA: Morgan Kaufmann Publishers, 556 – 559, 1993.