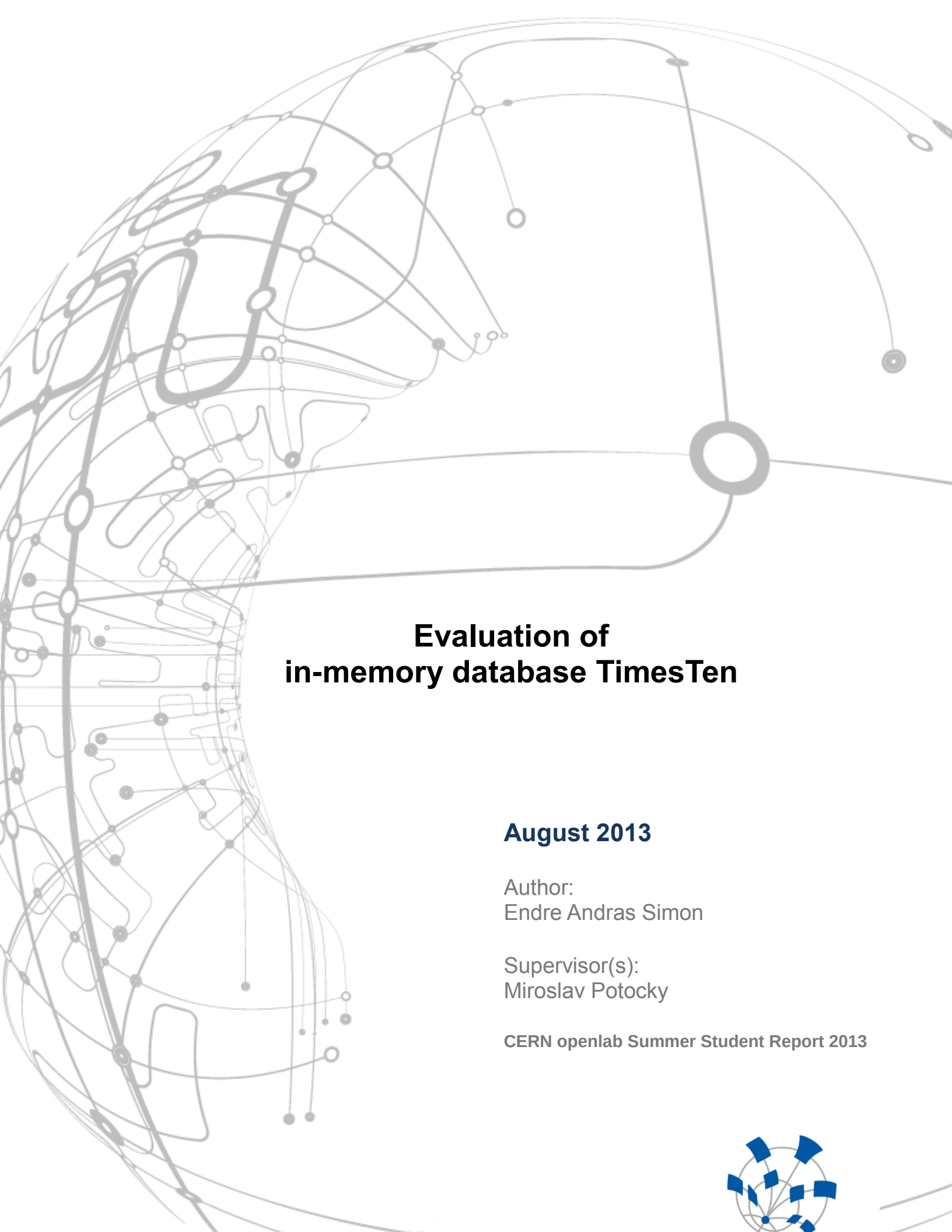

Evaluation of in-memory database TimesTen

August 2013

Author:
Endre Andras Simon

Supervisor(s):
Miroslav Potocky

CERN openlab Summer Student Report 2013

Project Specification

Oracle TimesTen In-Memory Database is a full-featured, memory-optimized, relational database with persistence and recoverability. For existing application data residing on the Oracle Database, TimesTen can serve as an in-memory cache database. This setup can provide great performance increase and almost instant responsiveness for database intensive applications. Cooperation between application and database support is needed to test integration, benefits and possibilities this product provides for database intensive applications in CERN.

Main goal is to test performance improvement in response time when using Oracle TimesTen in-memory database cache (IMDB) layer between high load CERN applications and their respective databases.

Abstract

In this paper I will introduce the key features of Oracle TimesTen In-memory database, focusing on the scenario when TimesTen is used as a cache between applications and their databases. Several industry standard benchmarks will be run against both Oracle database and Oracle TimesTen In-memory cache, to determine the performance gains when using Oracle TimesTen as a cache. Based on these results, we will examine the causes and consequences to make future assumptions. After reading this document, the reader will have a broad overview of uses-cases, when using TimesTen is advantageous.

Table of Contents

1 Introduction

CERN deals with a lot of data, thus the databases are indispensable part of the environment. All the experiments store their data in databases where most of them is between 1 and 12 TB in size. The LHC accelerator logging database (ACCLOG) is currently around 160 TB, and produced 110 GB new data in a day during the beam operations, so it has an expected growth up to 70 TB/year. Not only the experiments, but several other departments rely on database services. Several administrative, IT and Engineering databases are used for different purposes. In summary, more than 130 databases are used at CERN day to day, and because of the sensitive and valuable data, it is crucial to these databases to provide an efficient and reliable service. The DB group of IT department is responsible for administering these databases, and provide integrated support for users all around CERN.

Because of the large amount of data, the performances of the databases are not negligible. To analyse the data, and produce scientific results, there must be a quick way to query and process all the data, collected by the experiments. For this reason CERN started Openlab over a decade ago. Within Openlab, CERN is collaborating with leading IT companies, to accelerate the development of cutting-edge technologies. The DB group, besides the activities mentioned above, is also a part of the CERN Openlab, and experiments with several new technologies together with Oracle, to improve the provided services in the future. Oracle TimesTen In-Memory database is one of the promising opportunities, which can dramatically improve the performance in some cases. My task is to evaluate the performance of Oracle TimesTen In-Memory Database, and give a broad overview of the advantages and disadvantages of using TimesTen as a memory cache between high load CERN applications and their respective databases.

First we will have an overview about the used technologies in this document. We will get familiar with the basics of Oracle RDBMS and have an overview about the different capabilities of TimesTen. For benchmarking, HammerDB will be used, thus a short introduction will follow, about the several benchmark options and TPC standards. With this knowledge we can have an overview of the test system, and a detailed description of the used database schemas.

In the next section I will introduce the different test scenarios, and evaluation viewpoints. This section will cover the pre-testing scenarios, the testing schedules and the explanations of tests as well. I also try to predict the expected performance of TimesTen, providing tests to determine the upper and lower bounds of the IMDB.

In the fourth section, several results will be provided. This sections goal is to show the gathered results from different point of view, compared to each other. This section does not aim to interpret the outcome and solve the different disparities between the expectations and the real results.

In section Discussion and conclusion, the results will be explained and analysed, to have an overall picture about the impact of using TimesTen in the different test cases.

2 Testing methods

2.1 Technologies

In this section I'm summarizing the main properties of the benchmark environment. Below is a list of software used for the test.

- Oracle TimesTen In-Memory database [3][4][5]
- Oracle database [10]
- HammerDB [1][2][8]

The first two are products to compare, and the third is used to perform the benchmarks. In the next sections a more complete overview will be given.

2.1.1 Oracle database

Basic introduction

Oracle database is a widespread relational database management system (RDBMS), with several effective solutions for enterprises. The basic idea of RDBMS is to control the storage, organization and retrieval of data, hence it has the typically the following elements:

- Kernel code
- Metadata repository
- Query language

The data model which is used by an RDBMS is called relational model, and was invented by Codd in 1970. It is a well-defined mathematical structure, with operations on this structure. When an RDBMS moves data into a database, stores the data, or retrieves it, operations on the data model are done in the background. However the RDBMS distinguishes between two major types of operations:

- Physical operations
- Logical operations

In the first case, only content is selected through an operation, and this content will be retrieved as result. The second case is responsible for the low level data access, and optimization.

Oracle Database Architecture

An Oracle database server consists of two entities:

- A database
- At least one database instance

The two entities are strongly related, but not the same. A database is a set of files, which stores all the data of a database, and is independent from an instance. An instance is a set of structures in the memory, to manage the database. As you can see on Figure 1 the instance consist the system global area (SGA), and other background processes. [10]

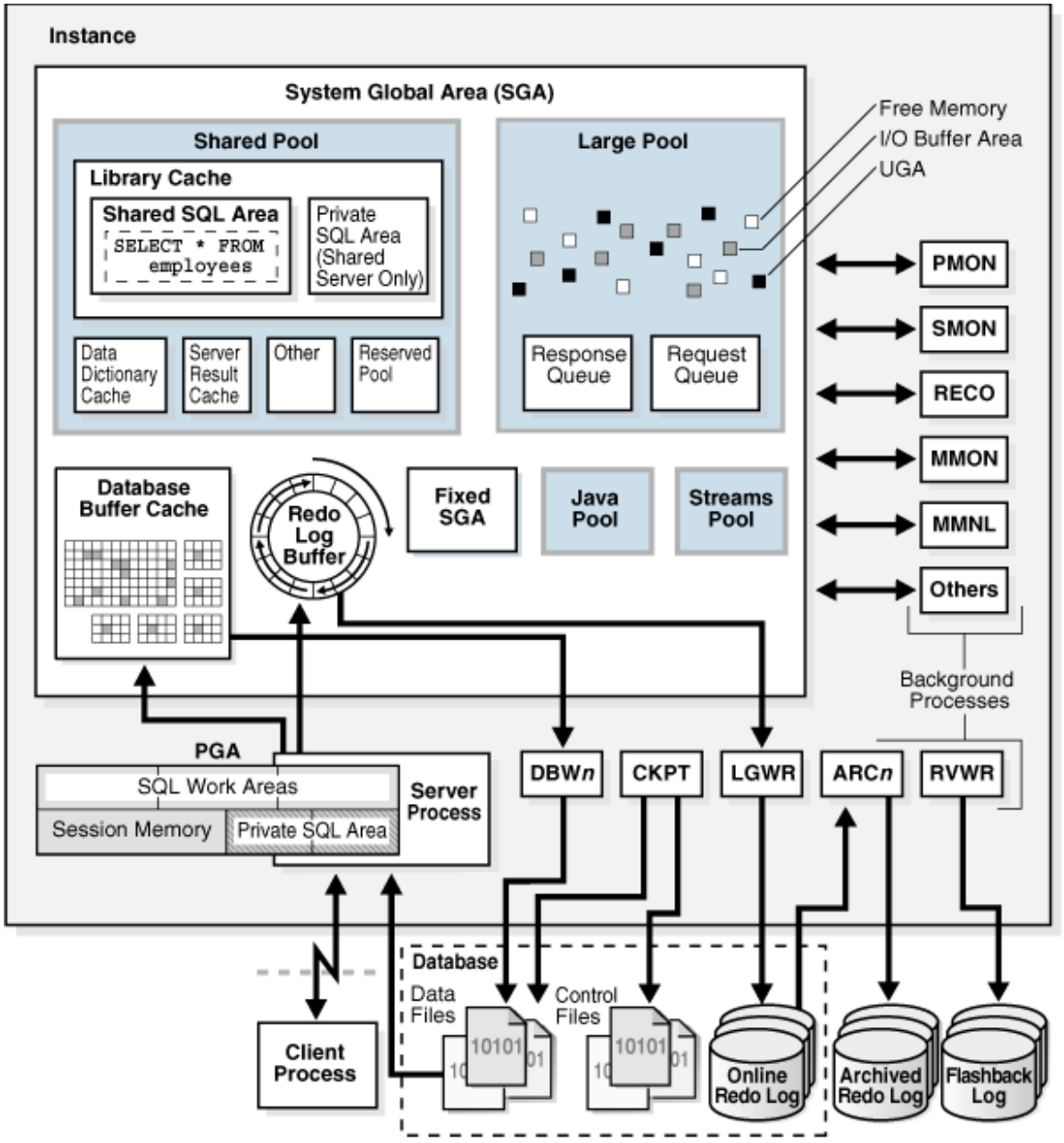

Figure 1. Oracle instance and database[1]

2.1.2 Oracle TimesTen In-Memory database

Basic introduction

Oracle TimesTen is an In-Memory database with cache capabilities. It is optimized to deal with data stored in the memory, therefore it provides extremely high throughput and fast response

[1] http://docs.oracle.com/cd/E11882_01/server.112/e25789/intro.htm#i68236

time. Because of these properties, TimesTen is ideal to use as a database cache for real-time applications or for applications where the high throughput is mandatory.

This remarkable improvement can be achieved by changing the behaviour of the data access. TimesTen already assumes that the data resides in memory, so memory optimized algorithms and data structures can be used. This means, that the complexity of database drops, and data can be queried faster. Compared to disk-based relational database management systems (RDBMS) TimesTen can gain impressive performance, because disk-based systems make the assumption that the data is written on the disk. Even when a disk based RDBMS holds the data in its own memory cache, the performance is restricted by the assumption that data resides on disk.[3] In contrast of that, when the assumption of disk-residency is removed, many things can be done much simpler:

- No buffer pool management is needed
- No extra copies needed
- Index pages shrink
- The number of machine instruction drops

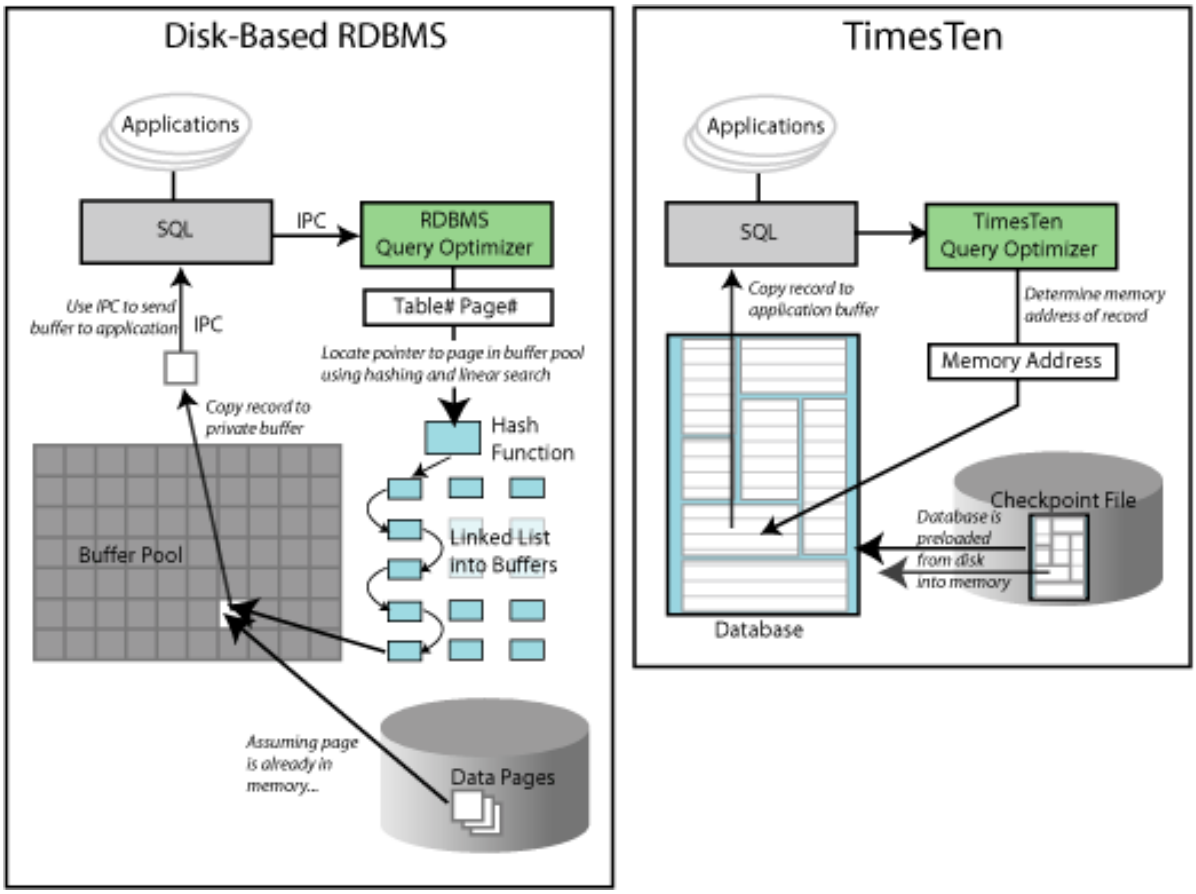

Figure 2. Comparing a disk-based RDBMS to TimesTen[2]

[2] http://docs.oracle.com/cd/E21901_01/doc/timesten.1122/e21631/overview.htm

The difference can be observed well on Figure 2. The image explains the differences between a disk-based RDBMS and TimesTen.

In the first case, the traditional RDBMS queries the database with an SQL query through shared memory. The query optimizer tries to evaluate this statement, and through hash functions finds the page and table numbers. The corresponding page in the buffer pool contains the data, and will be copied to a private buffer for further use. The client will receive the content of this private buffer through shared memory.

In case of TimesTen, the complexity is dramatically reduced. Consider that the application makes the same query in TimesTen. A direct connection will be used to pass the query to the optimizer. The optimizer simply determines the memory address of the record. Because the database resides in memory, the data can be easily copied to the private buffer, and the application can use it because of the direct link.

TimesTen features

Without further discussion I will introduce the key features of TimesTen, for detailed information please see the first chapter of [3].

- TimesTen API support
- Access Control
- Database connectivity
- Durability
- Performance through query optimization
- Concurrency
- Database character sets and globalization support
- In-memory columnar compression
- Data replication between servers
- Cached data with the IMDB Cache
- Load data from an Oracle database into a TimesTen table
- Business intelligence and online analytical processing
- Large objects
- Automatic data aging
- System monitoring
- Administration and utilities

Using TimesTen as an IMDB Cache

In this document, the most important features of TimesTen are the caching capabilities. Basic unit of caching in TimesTen is called cache group. A cache group is a set of related tables in a database. It can be configured to cache all or just a part of the database tables. Every cache group has exactly one root table, and one or more child tables. Each child table must reference the root table or another child table in the same group using a foreign key constraint. When data is loaded from the database to TimesTen, each row from the root table, and the related child tables are moved together. This subset of database is called cache instance.

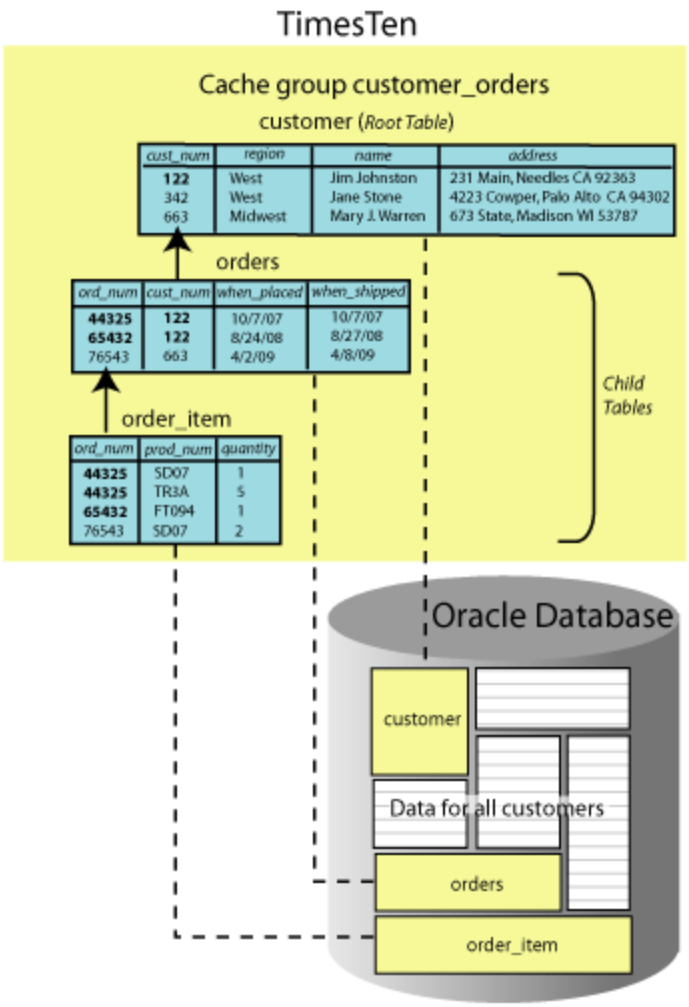

Figure 3. A cache group and three cache instances. [3]

The four main types of cache groups are:

- Read-only cache group

  Read only cache groups can cache and automatically refresh cache data from the underlying database. Read only cache group is suitable for caching heavily used data.

- Asynchronous writetrough cache group (AWT)

  AWT cache groups propagate committed data from the cache to the underlying database in asynchronous way. AWT is intended for high speed data capture and OLTP.

- Synchronous writetrough cache group (SWT)

[3] http://download.oracle.com/otn_hosted_doc/timesten/1122/doc/timesten.1122/e21634/concepts.htm#BABFBIEC

SWT is the same as AWT, but it propagates data in a synchronous way.

- User managed cache group

  User managed cache groups can implement customized caching behaviour.

TimesTen exchanges committed data with the Oracle Database. There are different types of data exchange operations, based on the direction of the transmitted data, and based on the way they are performed.

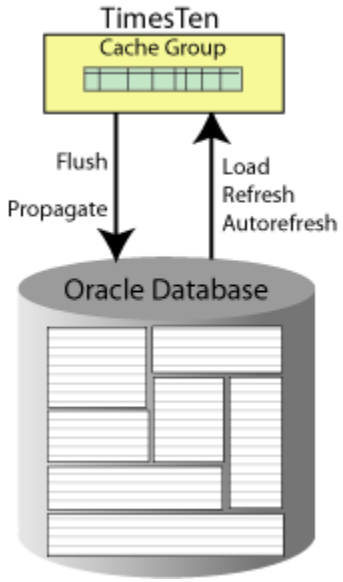

Figure 4. Data propagation between TimesTen and Oracle Database[4]

As you can see on Figure 4, *Flush* and *Propagate* operations transfer the committed data from TimesTen to Oracle Database. Flush is a manual operation and Propagate is executed automatically.

Load, Refresh and *Autorefresh* are used to transfer committed changes from the Oracle Database into cache tables. Load operation can load a selected set of data into the cache group. Refresh only looks for committed changes, and keeps the cached data up-to-date. Both operations are performed manually. Autorefresh does the same as refresh, but it is done automatically.

Another categorization can be based on how data is loaded into a cache group. There are explicitly loaded and dynamic cache groups. In an explicitly loaded cache group, cache instances are loaded manually, before any query occurs. The most common use-case of explicitly loaded cache groups is for data which is static, and can be predetermined before the application begins to perform queries. In dynamic cache group, data is loaded on demand from Oracle Database. Dynamic cache groups are used, when data cannot be predetermined before the application performs queries.

4 http://download.oracle.com/otn_hosted_doc/timesten/1122/doc/timesten.1122/e21634/concepts.htm#BABFBIEC

Cache groups can be classified based on how they share data across the grid as well. A cache group can be either local or global. The tables in a local cache groups are not shared across the grid members. Therefore, the cached tables in different grid members can overlap, and when data is committed, further coordination is needed between grid members to preserve consistency. Any types of cache group can be declared as a local cache group. Global cache groups share data between the other grid members. To keep the data consistent, committed data is propagated in a strict order in which they were executed within the grid. Only AWT cache groups can be defined as global.

	Explicitly loaded		**Dynamic**	
	Local	**Global**	**Local**	**Global**
Read only	**X**		**X**	
AWT	**X**	**X**	**X**	**X**
SWT	**X**		**X**	
User managed	**X**		**X**	

Table 1 The table shows the different types of cache groups

2.1.3 HammerDB

Basic introduction

HammerDB is a free, open source database benchmarking tool for Oracle, SQL Server, TimesTen, PostgreSQL, Greenplum, Postgres Plus Advanced Server, MySQL and Redis. It supports TPC-C and TPC-H industry standard benchmarks, and supports scripting, so the benchmarks can be easily extended. In our case it will be used to perform both TPC-H and TPC-C benchmarks against Oracle database and Oracle TimesTen In-Memory database.

An overview of TPC-C benchmarking

TPC-C is the benchmark of the Transaction Processing Performance Council (TPC), which measures the performance of online transaction processing systems (OLTP). The goal of the benchmark is to define operations that are supported by all transaction processing systems, and to define a schema which is used for benchmarking. The documentation of the benchmark is open to the public, so everybody can implement and use it for reliable benchmarking.

TPC-C simulates a real-life example of online transaction processing, where transactions are executed against the database. The benchmark implements a company that delivers orders, records payments and monitors several business parameters in the database.

The company has some warehouses, and each warehouse has ten sales districts. Each sales district maintains three thousand customers. The distribution of transactions are modelled after realistic scenarios, thus the most frequent transactions are entering a new order and receive payment from users. Less frequent transactions are transactions from operators, to maintain a particular sales district. As the company expands, the database grows with it, hence TPC-C benchmarks are easily scalable. Figure 5 shows the organization structure described above.

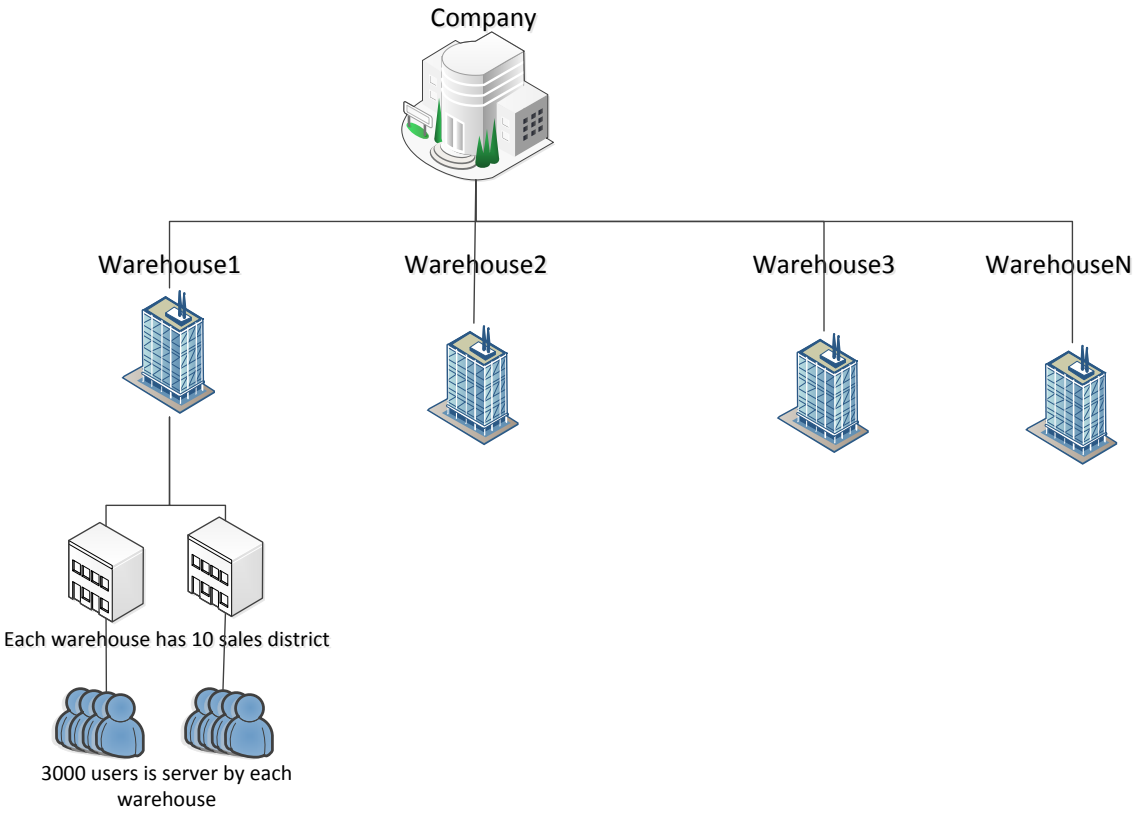

Figure 5. The organization structure of TPC-C

An overview of TPC-H benchmarking

TPC-H benchmark is a data warehousing benchmark, thus it executes a couple of business oriented ad-hoc queries and concurrent data modifications. This benchmark illustrates high-load systems that examine large volume of data, and executes complex queries to answer important questions.

The TPC-H schema size is not fixed, it can be manipulated trough a Scale Factor, so schemas can be either small or large depending on the system that you want to test. While performing TPC-H you will create a performance profile, based on the execution time of all 22 TPC-H queries. To

calculate the Composite-Query-per-Hour Performance metric (Qph@Size), you have to know the following things:

- Database size
- Processing time of queries in a single user case (Power test)
- Processing time of queries in a multi user case (Throughput test)

In the last case, the number of concurrent user is based on the size of the database, for more details please see [2] or [7].

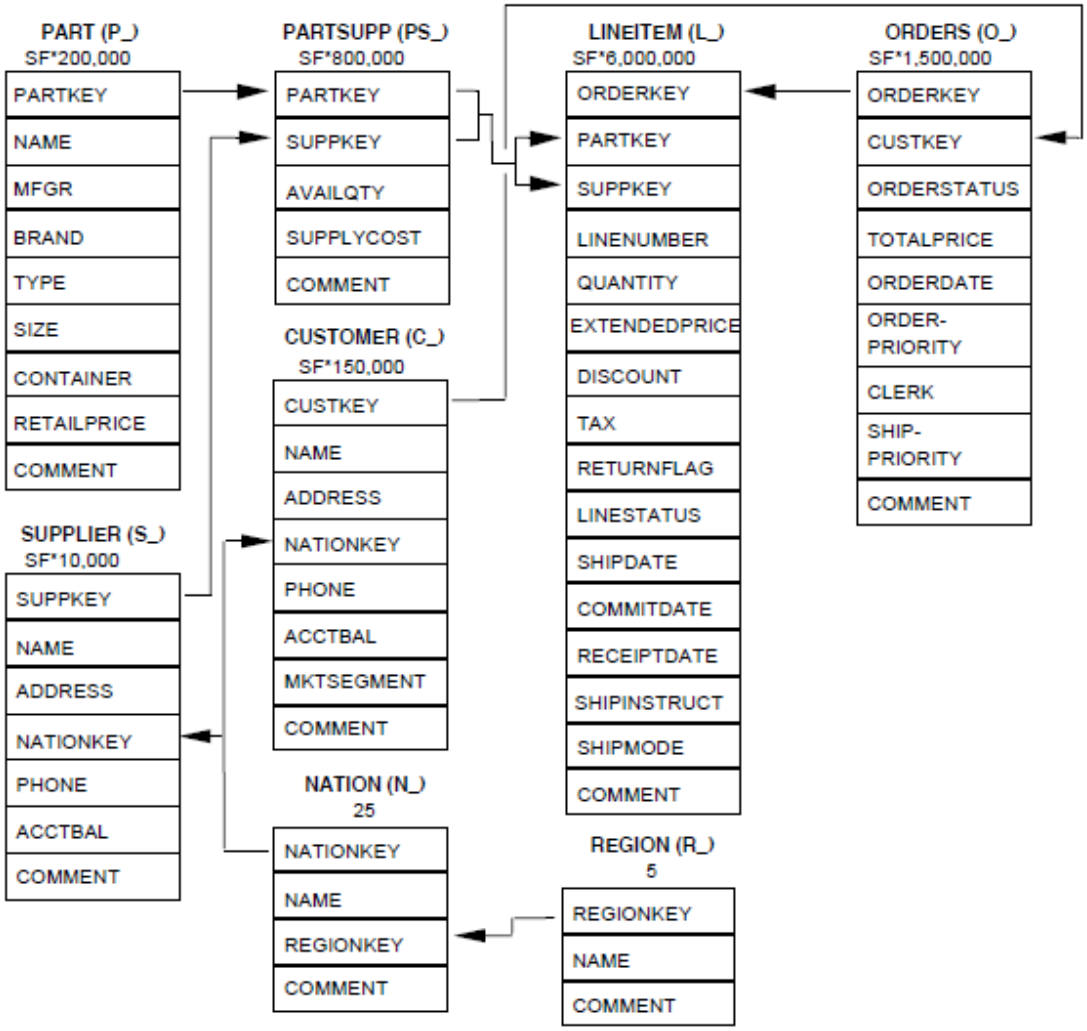

Figure 6. The TPC-H schema[5]

[5] http://www.tpc.org/tpch/spec/tpch2.16.0.pdf

2.2 Test network installation and configuration

2.2.1 Overview of the system

The test system, shown on Figure 7, is very simple, but straightforward. There are three main components which are necessary for successful testing:

- System Under Test
- Load Generation Server
- Administrator PC

The System Under Test (SUT) is the server running the Oracle Database and Oracle TimesTen In-Memory Cache. It is very important during the benchmarks to leave SUT configuration the same, to gain consistent results.

The Load Generation Server runs the HammerDB with TPC-C and TPC-H benchmarks. To be able to connect to the Oracle database and to the TimesTen In-Memory Cache, Oracle Instant Client and TimesTen libraries are required.

The Administrator PC is the PC which runs the monitoring tools, to gather information while benchmarks are running. Because this activity does not need high computing capacity, I choose to run this tool on the Load Generation Server.

The parts of the test system are connected through network connection, considering that network remains the same during the test, network latency will not cause notable problems in the test results.

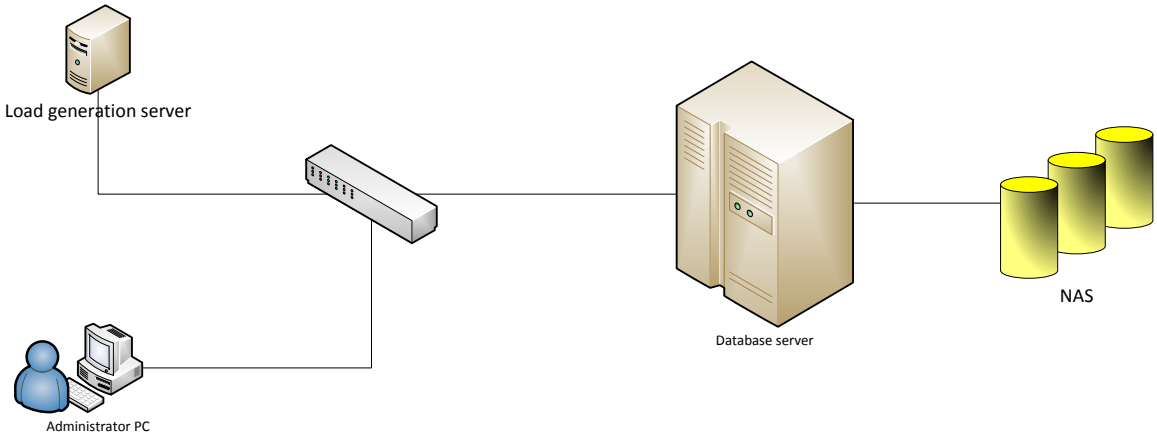

Figure 7. The infrastructure of the test-system.

2.2.2 Load Generation Server configuration

The configuration process of the load generation server is pretty simple. In this section, I will describe the main steps to set up the environment. You can download the software mentioned in this section by following the links in Appendix A. Because the Load Generation Server runs on Windows, I will describe the configuration process for Windows machines, but the main steps are the same for other operating systems.

Installing Oracle Instant Client

1. Download Oracle Instant Client, and unpack it somewhere (for example: *C:/instantclient*)
2. Create *C:/instantclient/tnsnames.ora*, and configure the appropriate settings to reach your SUT [11]. The following is an example for proper setting:

```
TTDB1 =
 (DESCRIPTION =
   (ADDRESS_LIST =
     (ADDRESS = (PROTOCOL = TCP)(Host = 10.16.6.151)(Port = 1521))
   )
 (CONNECT_DATA =
   (SERVICE_NAME = ORCL)
 )
)

TTCACHE1 =
 (DESCRIPTION =
 (CONNECT_DATA =
   (SERVICE_NAME = TTCACHE1)
   (SERVER = timesten_client)
 ))
```

3. Create *C:/instantclient/sqlnet.ora* and write the following code inside. This is required to the proper use of HammerDB [1].

```
TCP.CONNECT_TIMEOUT=15
SQLNET.AUTHENTICATION_SERVICES = (NTS)
DIAG_ADR_ENABLED=OFF
DIAG_SIGHANDLER_ENABLED=FALSE
DIAG_DDE_ENABLED=FALSE
```

4. Go to the *Control Panel/System/Edit environment variables* and add new system variable *SQLPATH* with the appropriate path (*C:/instantclient/* in this example). Add this path to the system's PATH variable too.

Installing Oracle TimesTen libraries

1. Download the appropriate version of Oracle TimesTen.
2. Run the setup, if you only need the client libraries select client only option [12].

Installing HammerDB

1. Download HammerDB.
2. Use the setup to install the software.

2.2.3 SUT configuration

This section will provide a guide to set up TimesTen on your server. Setting up TimesTen as an In-Memory Cache will be also covered.

Preparing the system to install TimesTen

1. Large pages

In order to use TimesTen you have to enable large page support. If enabled, you have to set the appropriate values. Consider an example, where TimesTen database is around 100GB. First you have to determine the huge page size.

```
[root@itrac1301 TTDB1]# cat /proc/meminfo | grep Hugepagesize
Hugepagesize:       2048 kB
```

Now you can calculate the number of huge pages easily [4]:

```
100GB     =      102400MB
2048kB    =      2MB
nr_hugepages = 102400MB / 2MB = 51200
```

Now, you have to edit `/etc/sysctl.conf`, and set the value, calculated above. After you are done editing, execute the command `sysctl –p` to let make changes immediately.

2. Semaphores

TimesTen uses 155 semaphores, plus one per each connection [4]. To set this number you have to edit the `/etc/sysctl.conf` files `kernel.sem` parameter. This parameter looks like this:

```
[root@itrac1301 TTDB1]# sysctl -a | grep kernel.sem
kernel.sem = 250        32000   100     128
```

The first parameter is `SEMMSL`, the maximum number of semaphore per array. The second parameter `SEMMNS` is the maximum number of semaphores. The third parameter is the maximum number per semop call (`SEMOPM`), and the last parameter is the number of arrays. As you can see the `SEMMNS = SEMMSL * SEMMNI` equation is satisfied, although is not necessary.

To set this parameter properly, you have to decide the maximum connections per TimesTen instance, and calculate the relevant number. For example if you want to allow 100 connections, you have to set `SEMMSL` to 255. After editing, use `sysctl –p.`

3. Shared memory

You have to set shared memory settings as well. The recommended value for the maximum shared memory size (`SHMMAX`) equals the size of the database. After this you have to set the

total number of memory pages (`SHMALL`). This value should be equal to `ceil(SHMMAX/PAGE_SIZE) [4]`. Page size is generally 4KB on x86 systems and 16KB on Itanium. For example, if you have a 64GB database on Itanium, set the following values:

```
kernel.shmmax=68719476736
kernel.shmall=4194304
```

Creating TimesTen admin user

To use TimesTen you have to specify the users group, who are allowed to use the IMDB. By default it is the primary group of the user, who installs the instance. You can modify the default setting, by giving the name of the group explicitly or by making the instance world accessible.

To install TimesTen, you also have to create an instance registry [4]. To do this, do the following:

```
[root@itrac1301 TTDB1]# groupadd ttadmin
[root@itrac1301 TTDB1]# mkdir /etc/TimesTen
[root@itrac1301 TTDB1]# chgrp –R ttadmin /etc/TimesTen
[root@itrac1301 TTDB1]# chmod 770 /etc/TimesTen
```

Now you can install TimesTen.

Installing Oracle TimesTen

Now, if you downloaded the TimesTen package, simply use the `setup.sh` command, located in the main directory and follow the instructions. After the installation is complete, the instance will be started. The last step is to set the environment for TimesTen, to make administrative things easier. To do this, login as the TimesTen user, and edit your `.bashrc` file like this, of course you should change the paths to your install directory:

```
export ORACLE_BASE=/ORA/dbs01/oracle/product/ttcache/
export ORACLE_HOME=/ORA/dbs01/oracle/product/ttcache/TimesTen/TTCACHE1
export
TIMESTEN_INSTALL_ROOT=/ORA/dbs01/oracle/product/ttcache/TimesTen/TTCACHE1
source $ TIMESTEN_INSTALL_ROOT/bin/ttenv.sh
export ORACLE_SID=TTCACHE1
export PATH=$ ORACLE_BASE
/TimesTen/TTCACHE1/bin:/usr/sue/sbin:/usr/sue/bin:/sbin:/bin:/usr/sbin:/usr
/bin
export TNS_ADMIN $ORACLE_BASE/TimesTen/TTCACHE1/network/admin
```

Setting up TimesTen as an In-Memory Cache

2.3 Creating the test schemas

HammerDB makes it easy, to create the TPC-C and TPC-H schemas, because of the built-in schema creator. Although, the TPC-C schema created by HammerDB cannot be cached in TimesTen, due to some missing primary and foreign keys. The `create_tpcc_cacheable.tcl`

script will create a totally equivalent scheme, with fixed properties. To create the TPC-H scheme you can easily use the built in schema generator. For more details using HammerDB schema generator see [1], [2] and [9].

3 Test cases and expectations

3.1 Pre-testing

Pre-testing aims to verify the test cases, database configurations and previously created TPC-C and TPC-H schemas. This ensures you that everything is fine before beginning the real benchmarking. After pre-testing, it is advisable, to back up the database, and to use this backup later as a starting point for the benchmarks. This will provide consistent and clean results.

3.1.1 Testing the TPC-C schema

Testing the TPC-C schema includes two things:

- Verifying that the schema is correct
- Verifying that the AWR snapshots are taken

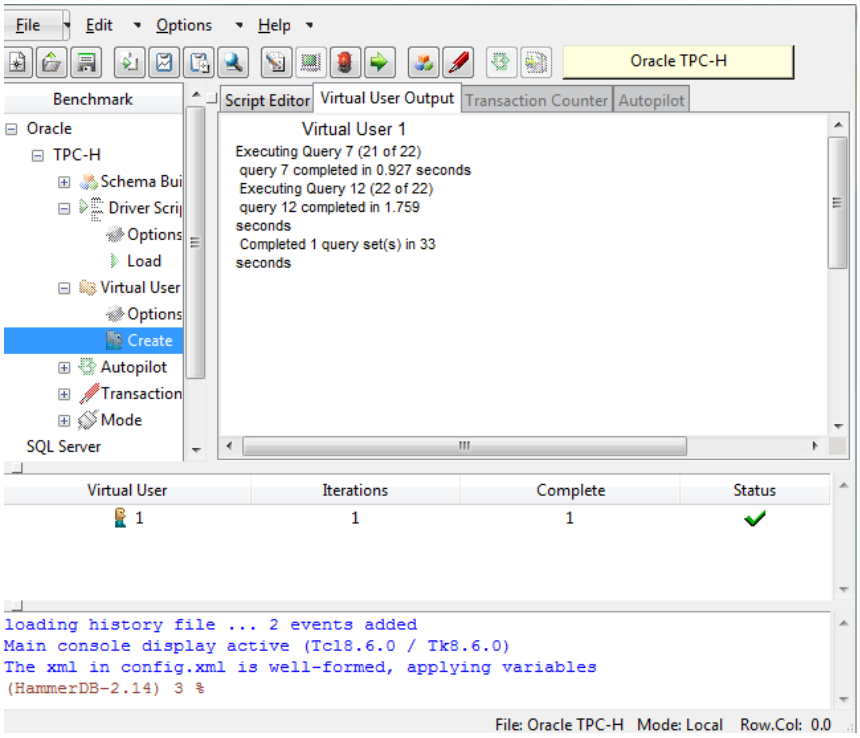

Figure 8. HammerDB after successfully finished benchmark

For this, you have to create a driver script which connects to the database, run a short test, and saves the changes in an AWR Snapshot. You can create it by the Driver Script creator in HammerDB, or you can use the scripts provided in Resource download links

Appendix B. If the script finishes properly, you can ensure that the created schema works, and you can also check the created AWR reports. For more information see [1] and [13].

3.1.2 Testing the TPC-H schema

According to [2] and [7], to do proper test we have to find the optimal Degree of Parallelism (DOP), when using TPC-H. To do this, you have to run the TPC-H benchmark several times, with different DOP values. At the end, you will have a similar graph to Figure 9, and you can easily determine the proper value for DOP. Note that you only need to calculate the value, when you use Oracle Database. For more details, see [2].

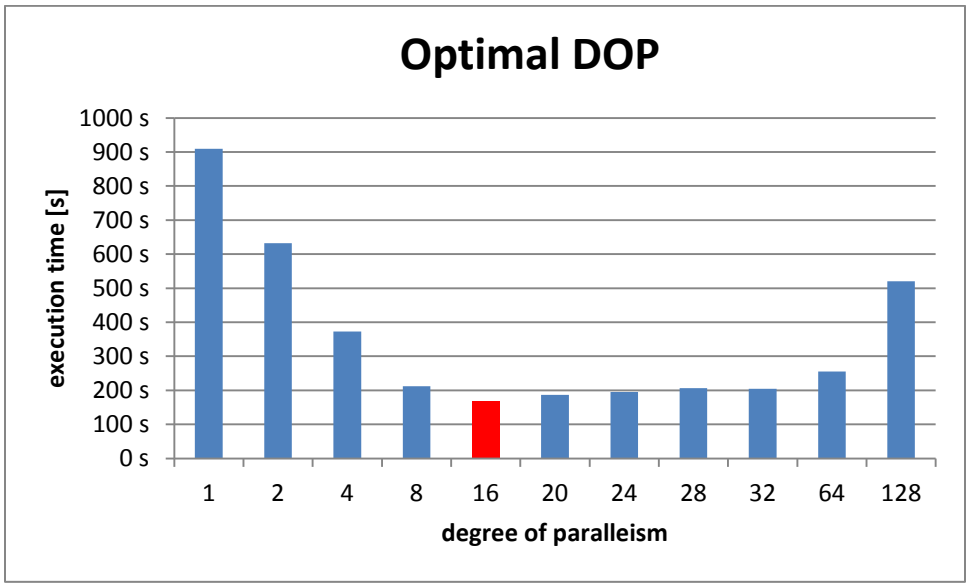

Figure 9. Finding optimal DOP.

3.2 Planning and preparation

3.2.1 Planning the TPC-C tests

As I need to compare the performance of the Oracle Database with and without TimesTen In-Memory cache, I have to run several tests both on TimesTen and on Oracle Database. Additionally, I had an opportunity to test the performance of the database when running on local disk and on NAS.

Because disk reads and writes are always bottlenecks, I tried to experiment with solutions which try to avoid these factors. The `commit_read` parameter of Oracle Database can modify the way of handling transactions [14]. When this parameter is set to `IMMEDIATE,WAIT` transactions are committed immediately and the client waits until the redo is flushed to disk. This is the default parameter, and this is the recommended setting to use. Although, when a lot of small transactions are executed in a row, this approach is not very effective. When `BATCH, NOWAIT` is used, the redo writes are buffered, and the real disk operation is executed after the buffer reach a limit. The `NOWAIT` option indicates that the client can continue without waiting for the transaction commit. This setting can result better performance, but there are data consistency risks associated. However for the tests performed the risk is negligible.

Because I also wanted to find out the upper limit of throughput on local disk and NAS, I executed benchmarks both in `IMMEDIATE,WAIT` and `BATCH, NOWAIT` mode.

For summary, the benchmarks on Oracle Database are:

- Local disk, `BATCH, NOWAIT` mode
- Local disk, `IMMEDIATE,WAIT` mode
- NAS, `BATCH, NOWAIT` mode
- NAS, `IMMEDIATE,WAIT` mode

In case of TimesTen, data resides in main memory, thus I/O is not a bottleneck anymore. One aspect of the benchmarks is to find the upper limit of performance. This consideration can be easily achieved, when we cache the whole underlying database in TimesTen, so we avoid data propagation between the database and TimesTen. In the other hand, I wanted to find out the performance when the database is not fully cached. This is important, because in real life, the databases cannot fit in the main memory.

The other aspect of testing TimesTen was to experiment with different cache groups. Running these tests, requires readable and writable cache instances, so I could only test Asynchronous Write through and Synchronous Write trough cache groups.

The benchmarks executed on Oracle TimesTen are the following:

- FULLY CACHED, AWT
- FULLY CACHED, SWT
- PARTIALLY CACHED, AWT
- PARTIALLY CACHED, SWT

For details please see

Appendix C.

3.2.2 Planning the TPC-H tests

Because TPC-H tests are rather CPU than I/O intensive, the considerations above are not true for these benchmarks. Because of this, I didn't considered any special cases for TPC-H, I ran the TPC-H power and throughput [2][7] tests against Oracle Database and TimesTen.

For details please see

Appendix C.

4 Results

4.1 TPC-C results

Results are available in Table 2 and Table 3. Following columns are presented:

- **Virtual users:** number of parallel virtual users
- **tpm:** raw transactions per minute
- **noPM:** TPC-C normalized transactions per minute
- **Average response time:** average response time for queries
- **CPU load:** average CPU load during the benchmark
- **Disk reads/write**: average disk read/write speed during the benchmark

4.1.1 Tests on Oracle

Oracle						
(E01) Local Disk, IMMEDIATE						
virtual users	**tpm**	**noPM**	**Average response time [ms]**	**CPU load**	**Disk reads [kB/s]**	**Disk writes [kB/s]**
1	1576	511	38.071	0.754688832	916.668	916.668
2	1746	572	34.364			
4	2206	719	27.199			
8	3408	1128	17.606			
16	3034	988	19.776			
32	2603	892	23.050			
64	1557	557	38.536			
(E02) Local Disk, BATCH						
virtual users	**tpm**	**noPM**	**Average response time [ms]**	**CPU load**	**Disk reads [kB/s]**	**Disk writes [kB/s]**
2	13699	4555	4.380	0.718584884	988.459	988.459
4	22568	7526	2.659			
8	22839	7626	2.627			
16	9377	3178	6.399			
32	8390	2798	7.151			
64	8124	2456	7.386			
(E04) NAS, IMMEDIATE						
virtual users	**tpm**	**noPM**	**Average response time [ms]**	**CPU load**	**Disk reads [kB/s]**	**Disk writes [kB/s]**
1	5769	1948	10.400	7.535618878	0.096	0.096
2	21500	7194	2.791			
4	103614	34525	0.579			
8	139729	46611	0.429			
16	262690	87760	0.228			
32	368560	123393	0.163			
64	364169	123175	0.165			
(E05) NAS, BATCH						
virtual users	**tpm**	**noPM**	**Average response time [ms]**	**CPU load**	**Disk reads [kB/s]**	**Disk writes [kB/s]**
1	32611	12127	1.840	13.36701291	0.117	0.117
2	54620	18124	1.098			
4	105142	35144	0.571			
8	181038	60470	0.331			
16	356755	119403	0.168			
32	490354	164443	0.122			
64	454420	153174	0.132			

Table 2 Oracle TPC-C benchmark results

4.1.2 Tests on TimesTen

TimesTen						
(E08) AWT, Full cache						
virtual users	**tpm**	**noPM**	**Average response time [ms]**	**CPU load**	**Disk reads [kB/s]**	**Disk writes [kB/s]**
1	60903	18998	0.985	13.15290231	4.410	4.410
2	25313	7923	2.370			
4	193985	61044	0.309			
8	326518	102078	0.184			
16	428839	133810	0.140			
32	311485	98485	0.193			
64	310349	97634	0.193			
(E09) SWT, Full cache						
virtual users	**tpm**	**noPM**	**Average response time [ms]**	**CPU load**	**Disk reads [kB/s]**	**Disk writes [kB/s]**
2	27512	8608	2.181	8.307428571	2.315	2.315
4	32625	10116	1.839			
8	50456	15652	1.189			
16	113333	35332	0.529			
32	88161	27383	0.681			
64	153954	47907	0.390			
(E10) AWT, Partial cache						
virtual users	**tpm**	**noPM**	**Average response time [ms]**	**CPU load**	**Disk reads [kB/s]**	**Disk writes [kB/s]**
1	87956	29845	0.682	3.598264642	4.102	4.102
2	110429	34245	0.543			
4	98267	30544	0.611			
8	231255	71917	0.259			
16	224658	69819	0.267			
32	209213	64958	0.287			
	204134	63308	0.294			
(E11) SWT, Partial cache						
virtual users	**tpm**	**noPM**	**Average response time [ms]**	**CPU load**	**Disk reads [kB/s]**	**Disk writes [kB/s]**
1	17049	5262	3.519	3.678125	9.158	9.158
2	30582	9514	1.962			
4	47008	14477	1.276			
8	52108	16186	1.151			
16	13659	4237	4.393			
32	50058	15516	1.199			
64	56101	17372	1.069			

Table 3 TimesTen TPC-C benchmark results

4.2 TPC-H results

Table 4 contains the execution times for each query, on the different configurations. Best performance is highlighted with red.

	Response time		
	Oracle Local	Oracle NAS	TimesTen
Query 1	8.749 s	24.807 s	41.317 s
Query 2	0.857 s	1.487 s	2.472 s
Query 3	13.925 s	33.413 s	24.787 s
Query 4	2.945 s	23.958 s	55.604 s
Query 5	22.817 s	24.214 s	35.073 s
Query 6	1.717 s	26.039 s	2.822 s
Query 7	3.571 s	24.216 s	87.917 s
Query 8	3.441 s	26.487 s	43.625 s
Query 9	8.196 s	27.717 s	48.977 s
Query 10	7.753 s	29.902 s	35.599 s
Query 11	9.461 s	13.656 s	28.651 s
Query 12	3.004 s	24.115 s	11.921 s
Query 13	3.987 s	4.024 s	49.601 s
Query 14	2.880 s	24.431 s	15.040 s
Query 15	1.875 s	23.604 s	10.704 s
Query 16	1.580 s	3.474 s	24.968 s
Query 17	2.280 s	24.360 s	201.330 s
Query 18	8.230 s	49.218 s	100.170 s
Query 19	3.231 s	24.639 s	17.092 s
Query 20	2.657 s	24.952 s	63.516 s
Query 21	7.830 s	52.097 s	0.013 s
Query 22	1.470 s	1.520 s	6.389 s

Table 4 TPC-H performance profiles

5 Discussion and conclusion

5.1 TPC-C conclusions

As you could see the different Oracle performances, it turned out that disk I/O speed is the key aspect of high throughput. The test system with performs way better than with local disk. On Figure 10 you can see the results on a plot. As expected, BATCH mode performs better on both local disk and NAS. Although higher performance comes with more CPU load and more intensive disk usage.

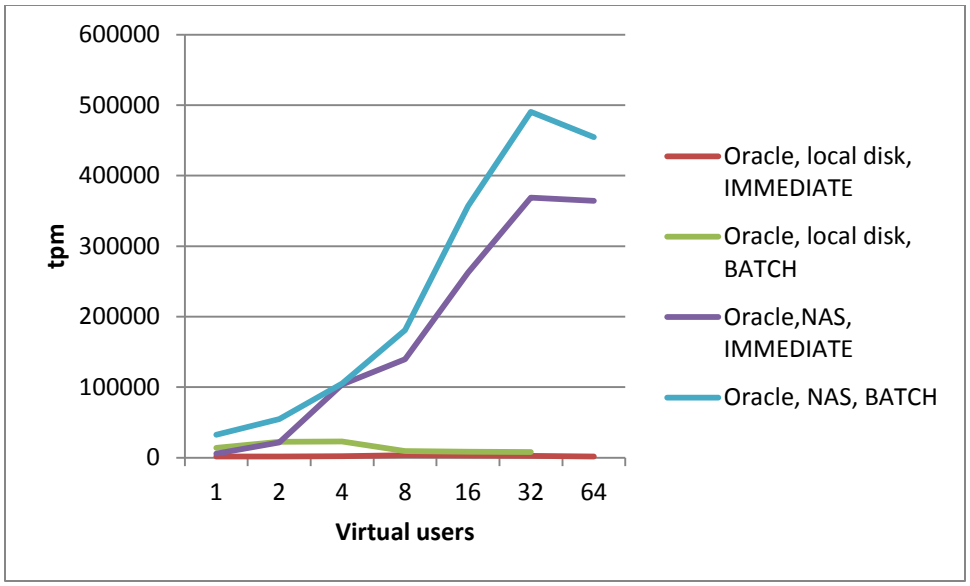

Figure 10. Benchmark results of Oracle Database.

Using TimesTen with fully cached database tables, with AWT cache groups gives remarkable results, but in real-life scenarios caching a whole database is not always possible. Partial cached database tables come with lower performance, but the usage of system resources is less, and scalability seems to be better.

On the other hand, SWT cache groups have moderate performance, because of the immediate synchronisation of data. They scale better than AWT groups, and CPU and disk usage is also moderate.

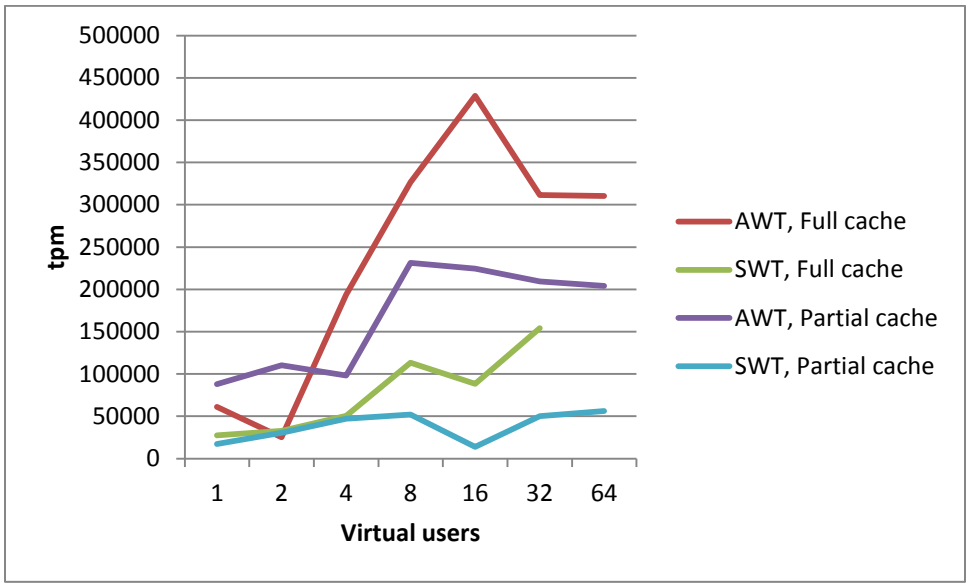

Figure 11. Benchmark results of TimesTen

In conclusion, for heavy OLTP workload TimesTen can be quite good, considering that NAS storage based traditional database can only reach similar throughput by sacrificing data consistency in BATCH commit mode. If TimesTen is used in grid, the performance can be even better, and data consistency will be also preserved.

5.2 TPC-H conclusions

Based on the performance profile, TPC-H results are surprising. As shown on Figure 12, Oracle on local disk performs better than Oracle on NAS and even better as TimesTen. The main reason of disparity is not obvious, but with the AWR and TTSTATS reports I tried to find the main reason.

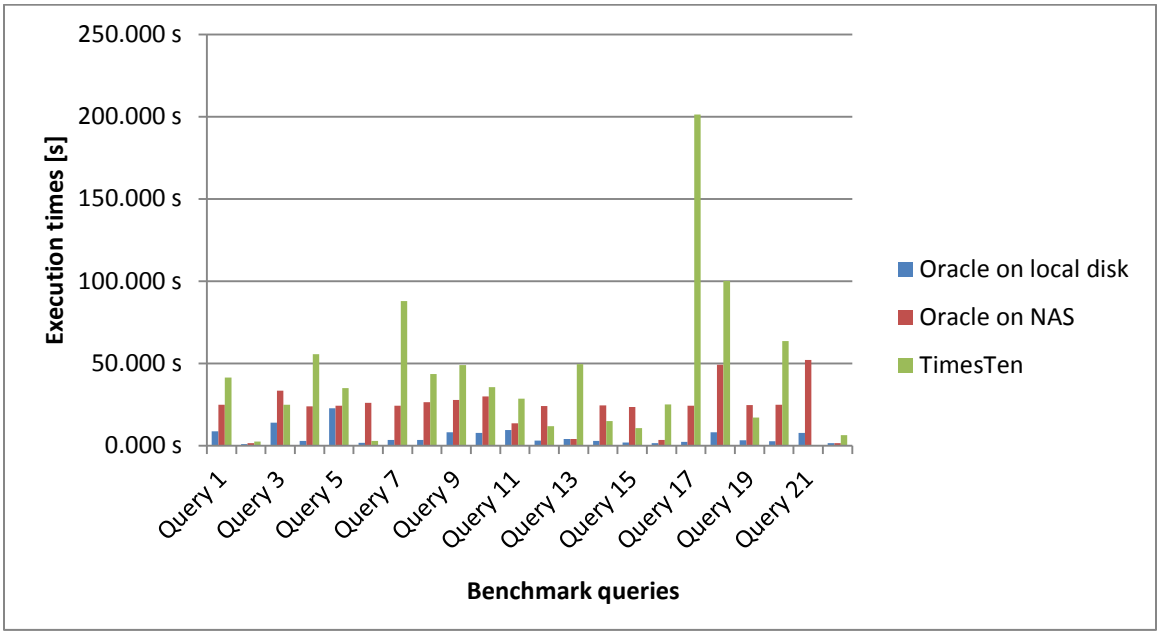

Figure 12. Performance profile

On Figure 13 and Figure 14 we can see the TOP 5 wait events. It is clear, that in the second case, direct path read is responsible for the performance drop. This means, that in this particular case, NAS had a higher latency than local disk. Fixing this problem avoids this kind of bottleneck.

Event	Waits	Time(s)	Avg wait (ms)	% DB time	Wait Class
DB CPU		1,475		99.49	
direct path read	465,177	126	0	8.49	User I/O
PX Deq: Table Q Get Keys	141	8	54	0.51	Other
library cache: mutex X	1,517	5	3	0.32	Concurrency
db file scattered read	17,328	3	0	0.22	User I/O

Figure 13. Oracle on local disk AWR report

Event	Waits	Time(s)	Avg wait (ms)	% DB time	Wait Class
direct path read	165,357	7,531	46	65.00	User I/O
DB CPU		2,397		20.69	
db file scattered read	25,440	190	7	1.64	User I/O
PX Deq: Table Q Get Keys	182	43	238	0.37	Other
db file sequential read	9,333	41	4	0.36	User I/O

Figure 14. Oracle on NAS AWR report

In the second case, the average performance of TimesTen is similar to the performance of Oracle Database on NAS, although the distribution is very different. The TTSTATS report shows nothing suspicious. The explanation of that is when the query hits the cached data, queries are executed extremely fast, but on the other hand, if the data is not cached, it takes some time to load it from the underlying database.

The conclusion is that performance of TimesTen really depends on actually cached data and on the size of memory cache. If somehow higher hit rate can be achieved, TimesTen would outperform Oracle Database. To examine these aspects exactly, more difficult test configuration (i.e. Real application testing – workload capture), and more time is needed to perform related tests.

References

[1] Oracle OLTP (Transactional) Load Testing, http://hammerora.sourceforge.net/hammerora_oracle_oltp_v2.8.pdf,

[2] Oracle DSS/Data Warehousing Testing Guide, http://hammerora.sourceforge.net/hammerora_oracle_dss_v2.7.pdf

[3] Oracle TimesTen Introduction, http://download.oracle.com/otn_hosted_doc/timesten/1122/doc/timesten.1122/e21631/toc.htm

[4] Oracle TimesTen Installation guide, http://download.oracle.com/otn_hosted_doc/timesten/1122/doc/timesten.1122/e21632/toc.htm

[5] Oracle TimesTen Cache Users Guide, http://download.oracle.com/otn_hosted_doc/timesten/1122/doc/timesten.1122/e21634/toc.htm

[6] TPC-C Benchmark Standard Specification, Revision 5.11, February 2010, http://download.oracle.com/otn_hosted_doc/timesten/1122/doc/timesten.1122/e21634/toc.htm

[7] TPC-H Benchmark Standard Specification, Revision 2.1.6.0, June 2013, http://download.oracle.com/otn_hosted_doc/timesten/1122/doc/timesten.1122/e21634/toc.htm

[8] Oracle TimesTen OLTP Load Testing Supplement, http://hammerora.sourceforge.net/hammerdb_timesten_oltp.pdf

[9] Todd M. Helfter, Oratcl Users's Guide and Reference, Revision5, http://oratcl.sourceforge.net/OraTcl_Users_Guide_and_Reference.pdf

[10] Oracle Database Documentation Library, 11g Release 3, http://www.oracle.com/pls/db112/homepage

[11] Oracle whitepaper, Instant client: An overview, May 2004, http://www.oracle.com/technetwork/database/features/oci/instant-client-wp-131479.pdf

[12] Working with the TimesTen client and Server, 11.2.2. E21633-05., http://docs.oracle.com/cd/E11882_01/timesten.112/e21633/client_server.htm

[13] Automatic Workload Repository (AWR) in Oracle Database 10g, http://www.oracle-base.com/articles/10g/automatic-workload-repository-10g.php

[14] Oracle Database Reference, 11.1, B28320-03, http://docs.oracle.com/cd/B28359_01/server.111/b28320/initparams032.htm

Appendix A

Resource name	Url
HammerDB installer	http://hammerora.sourceforge.net/download.html
Oracle Instant Client	http://www.oracle.com/technetwork/database/features/instant-client/index-097480.html
Oracle TimesTen	http://www.oracle.com/technetwork/products/timesten/downloads/index.html

Table 5 Resource download links

Appendix B

List of tcl script I used.

Script name	Description
Oraclestats.tcl	Creates TPC-C benchmark for Oracle, and stores some custom statistics in *stats.stats* schema.
Timestenstats.tcl	Creates TPC-C benchmark for TimesTen, and stores some custom statistics in *stats.stats* schema.
tpcc_scheme_cacheable.tcl	Creates the TPC-C schema, which is easily cacheable in TimesTen.

Table 6 List of scripts I used.

Appendix C

Description	Code	DB User	Folder
Oracle database, IMMEDIATE,WAIT	1	TPCC	E01
Oracle database, BATCH,NOWAIT	2	TPCC	E02
Oracle database TPCH	3	TPCH	E03
Oracle database, IMMEDIATE,WAIT - NAS	4	TPCC	E04
Oracle database, BATCH,NOWAIT - NAS	5	TPCC	E05
Oracle database TPCH - NAS	6	TPCH	E06
TimesTen TPCH	7	TPCH	E07
TimesTen, full cache, AWT	8	TPCC	E08
TimesTen, full cached, SWT	9	TPCC	E09
TimesTen, partially cached, AWT	10	TPCC	E10
TimesTen partially cached, SWT	11	TPCC	E11

Table 7 Benchmark scenarios