Oracle TimesTen: An In-Memory Database for Enterprise Applications

Tirthankar Lahiri
tirthankar.lahiri@oracle.com

Marie-Anne Neimat
neimat@outlook.com

Steve Folkman
steve.folkman@oracle.com

Abstract

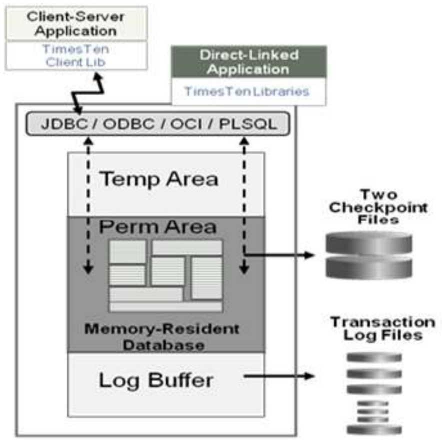
In-memory databases can provide a significant performance advantage over disk-oriented databases since they avoid disk IO, and since their storage managers are built and optimized for complete memory residency. In this paper we describe the functionality of the Oracle TimesTen In-Memory Database – a full-featured memory optimized relational database. With SQL-92 and ACID compliance, TimesTen is used by thousands of customers in a variety of applications – both for high-performance OLTP applications as well as for Business Analytics applications. While TimesTen can be used as a standalone database, it can also be used as a high-performance transactional cache that seamlessly caches data from an underlying Oracle RDBMS. This configuration is suitable for Oracle RDBMS applications that require real-time management of some of their data and scale-out on private or public clouds.

1 Introduction

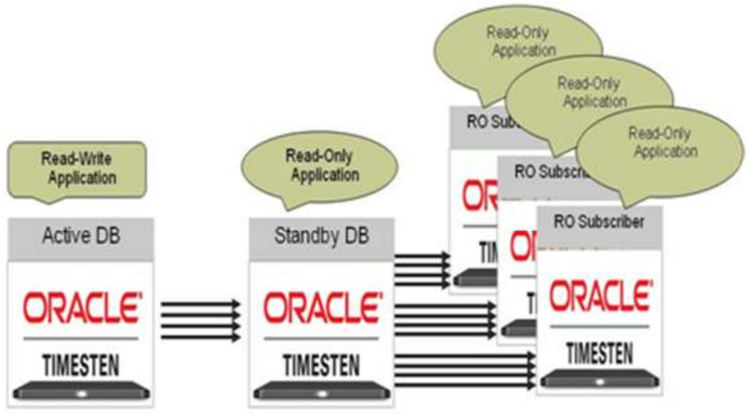
With increasing DRAM densities, systems with hundreds of GigaBytes, or even TeraBytes of memory are becoming common. A large fraction of user data, and in many cases, all user data, can be stored entirely in DRAM avoiding the need for costly disk IO for query processing. As a result of this trend, there are numerous in-memory databases that are commercially available now.

H-store [18] is an in-memory database optimized for highly concurrent OLTP workloads. However H-store requires applications designers to have complete understanding of their data access patterns in order to avoid conflicts, and all database accesses must be performed in terms of one or more pre-defined stored procedures. H-store is therefore not well-suited for ad-hoc applications, nor for applications with complex transactions. MonetDB [4] is a database that is highly optimized for complex query workloads, employing a shared-nothing architecture and columnar processing. MonetDB however is not well-suited for write-intensive OLTP workloads and as such cannot be used as a general-purpose in-memory solution. In [5] the author describes a vision for a general purpose in-memory database for SAP applications based on columnar storage and compression. However, columnar storage is notoriously expensive to maintain in an update-intensive environment, and in our experience does not lend itself well to OLTP workloads characterized by ad-hoc queries and concurrent DML – often just single-row updates. MySQL cluster [1] is a widely used scale-out in-memory database for high throughput OLTP environments with stringent HA requirements, however it is not designed for analytics workloads.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering



(a) Basic structure



(b) TimesTen replication

Figure 1: TimesTen architecture and replication

Oracle TimesTen is a an enterprise-class in-memory database with a rich feature set, with the intention of being applicable to a wide variety of workloads and data-sets. TimesTen has full-featured SQL support and supports standard APIs, full ACID properties, and high-availability mechanisms. TimesTen can also be used to complement an Oracle RDBMS by providing a high-performance transactional cache for performance-critical data. This cache can be scaled-out to a grid across a networked cluster of nodes. It is because of this gestalt of features, that TimesTen is the most widely deployed in-memory database in existence today. It is used in many different application areas such as Telecommunications, Financial Services, Ecommerce, Fraud Detection, etc., all of which have stringent response time and throughput requirements. In response to the growing need for high-performance real-time analytics, Oracle recently announced the Exalytics In-Memory machine in which TimesTen is the in-memory relational repository for interactive BI workloads.

In this paper, we first describe the basic architecture of the Oracle TimesTen In-Memory Database. Second, we describe how TimesTen replication can be used to provide a highly-available configuration. Third, we describe the *In-Memory Database Cache* configuration and how it can be used to scale-out TimesTen to a grid of caches. We conclude with performance results.

2 Basic Architecture

The TimesTen database is a memory-resident and completely persistent database. As shown in Figure 1(a), TimesTen has a shared memory segment storing the database. The memory segment is divided into three areas of configurable sizes: a permanent portion storing database objects, a temporary portion for run-time allocations, and a buffer for the transaction log. The on-disk structures comprise 2 checkpoint files for capturing the state of the in-memory permanent partition, and transaction log files for logging changes to the permanent partition of the database.

2.1 APIs and connectivity

TimesTen provides standard APIs for database operations. These include ODBC, JDBC, as well as the Oracle Call Interface (OCI). Applications connect to TimesTen either via a conventional client-server protocol, or an optimized *direct-mode* protocol in which the application directly links the TimesTen library. In direct mode, all database API invocations are treated simply as function calls into the TimesTen shared library allowing for in-process execution of database code.

2.2 Basic storage manager design

Some basic design principles guide the design of the storage manager. Concurrency control mechanisms are both light-weight (e.g. using latches instead of locks whenever possible) and fine-grained to allow for maximal scaling on multi-core architectures. Another over-arching design principle is the use of memory-based addressing rather than logical addressing. For instance, indexes in TimesTen contain pointers to the tuples in the base table. The metadata describing the layout of a table contains pointers to the pages comprising the table. Therefore both index scans and tablescans can operate via pointer traversal. This design approach is repeated over and over again in the storage manager, with the result that TimesTen is significantly faster than a disk-oriented database, even one that is completely cached – since there is no overhead from having to translate logical rowids to physical memory addresses of buffers in a buffer cache.

Multiple types of indexes are supported by TimesTen. TimesTen supports Hash Indexes for speeding up lookup queries, Bitmap Indexes for accelerating star joins with complex predicates, as well as Range Indexes for accelerating range scans. Since the data is always in memory, indexes do not need to store key values, with significant space savings. Indexes are used far more aggressively by the TimesTen optimizer than by a disk-based optimizer: They are used for scans, joins, group by computations, etc. For instance, the optimizer often prefers index scans over full table scans since the CPU cost of index scans is usually lower, unless the column being scanned is compressed. Extensive use of indexes is an advantage in-memory databases have over disk-based databases as the use of indexes in disk-based databases will result in many disk seeks if the index traversal does not match the clustering of records on disk.

The TimesTen storage manager supports columnar compression using dictionary-based encoding. With this feature, the column values are replaced with dictionary table identifiers. Each column has a separate dictionary of unique values, and the compression domain spans the whole column (or a group of columns). Columnar compression has been observed to offer a 5-10x compression benefit (See Section 5.3) and helps to accommodate large datasets as is typical of BI and OLAP workloads. For example, the present generation Exalytics Machine has a DRAM capacity of 1TB. This allows for a conservative limit of 500GB for the TimesTen permanent memory segment (taking into account the needs of the OS, and the TimesTen temporary memory area) but with compression allows an effective capacity of 2.5-5 TB.

2.3 Transactional durability

TimesTen has been designed from the ground up to have full ACID properties. TimesTen employs checkpointing with write-ahead logging for durability. Logging is frequently a scaling bottleneck on large multi-core architectures, and for this purpose – again in keeping with the fine-grained concurrency control design motif, TimesTen has a multi-threaded logging mechanism so that generation of log is not serialized. In this mechanism, the TimesTen log buffer is divided into multiple partitions which can be populated in parallel by concurrent processes. Sequential ordering of the log is restored when the log is read from disk. For the highest performance, TimesTen offers an optional *delayed-durability* mode: an application commits by writing a commit record into the log buffer without waiting for the log to be forced to disk. The log is periodically flushed to disk every 100 mSec as a background activity. This mode of operation allows the highest possible throughput with the possibility of a small (bounded) amount of data loss, unless the system is configured with 2-safe replication as described in Section 3.

The database has two checkpoint files for the permanent data, and a variable number of log files. Once a database checkpoint is completed, the checkpointing operation switches to the other file. With this approach, one of the two checkpoint files is always a completed checkpoint of the in-memory contents and can be used for backup and roll-forward recovery. TimesTen supports fuzzy checkpointing and the user can configure the rate of checkpoint disk writes.

TimesTen offers read-committed transactional isolation by default (serializable isolation as an application choice). With read-committed isolation, application bottlenecks are minimized since TimesTen has a lockless multi-versioning mechanism for read-write concurrency and row-level locking for maximal write-write concurrency. Row-level multi-versioning allows readers to avoid getting any locks altogether, while row-level locking provides the highest possible scalability for update-intensive workloads.

3 High Availability

The vast majority of TimesTen deployments use replication for high-availability. TimesTen replication is log-based, relying on shipping log records for committed transactions from a transmitter database to a receiver database, on which the changes in the log are then applied. Replication can be configured with multiple levels of resiliency. For the highest resiliency, TimesTen provides *2-safe* replication. With 2-safe replication, a transaction is committed locally only after the commit has been successfully acknowledged by the receiver. 2-safe replication is often used with non-durable commit. This combination allows applications to achieve commit durability in two memories, without requiring any disk IO.

Replication can be at the level of individual tables or at the level of the whole database. Replication can be configured to be multi-master and bi-directional. The preferred configuration for replication is to have whole database replication from an *Active* Database to a *Standby* Database. Standby Databases can in turn propagate changes to a number of *Read-Only Subscriber* Databases, as depicted in Figure 1(b). In this type of replication scheme, the Active Database can host applications that both read and modify the database. The Standby and the Read-Only Subscriber Databases can host read-only applications. If a failure of the Active occurs, the Standby can be promoted to become an Active Database. If a failure of the Standby occurs, the Active will then replicate directly to the Read Only Subscribers.

In keeping with the design principle of end to end parallelism, replication can be parallelized across multiple *replication tracks*. Each track consists of a replication transmitter on the sending side, and a replication receiver on the receiving side. Parallel replication preserves commit ordering, but allows transactions without any data dependencies to be applied in parallel by the receiver. Together with log parallelism, replication parallelism provides end to end parallelism between the master and the receiver, and the highest possible throughput.

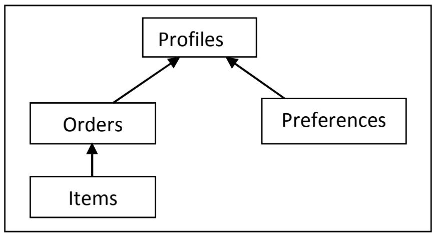
4 IMDB Cache Grid

TimesTen as we have shown is a full-featured database with durability, persistence and high availability and can be used as the only database of record for an application. However, in practice most enterprises still store their business critical data in full-featured disk-based database such as the Oracle RDBMS. The storage capacities and functionality far outweigh those of in-memory databases. For this purpose the IMDB cache configuration allows TimesTen to be deployed as a persistent transactional cache [2] for data in an Oracle database. This is a cache that the application must be aware of, since it is a full featured database embedded in the application tier.

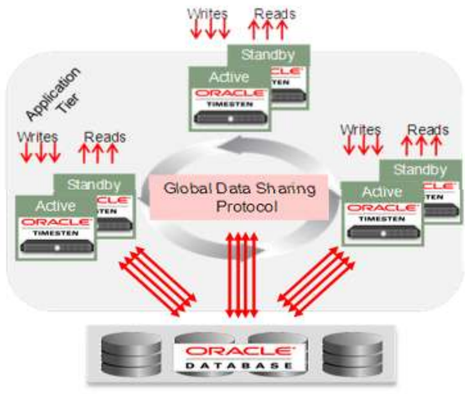
Deploying TimesTen as an application-tier cache against a disk-based database can greatly accelerate an application even when the database working set is fully cached in memory within the buffer cache. This is for two reasons:

1. Application Proximity: The TimesTen database can be collocated in the application-tier, resulting in lower communication costs to the database for access to cached data. For the best response time, TimesTen can be directly linked to the application providing in-process execution of operations on cached data.

2. In-Memory Optimizations: As stated in Section 2, the TimesTen storage manager is built assuming full memory-residency; it requires far fewer instructions for database operations than disk-based databases.



(a) Example cache group



(b) IMDB cache grid

Figure 2: TimesTen cache group and cache grid

4.1 Cache groups

TimesTen allows a collection of tables to be declared as a *Cache Group* against corresponding tables on an Oracle database. A cache group is a syntactic declaration and comprises one *Root table*, and optional *Child tables* related by foreign key constraints. For instance, for an ecommerce application, one approach to creating a cache group would be to have a root table for all user profiles, and child tables for open orders placed by each user, and the list of preferences for each user. This cache group is depicted in Figure 2(a).

Cache groups can be loaded from an Oracle database in one of two ways:

Pre-Loaded: In this mode, the cache group is explicitly loaded before the workload is run. For instance in the above example, all users could be preloaded from the Oracle RDBMS into the cache group.

Dynamically Loaded: In this mode, data is brought into the cache when referenced. For instance, when a user logs in, it could be at that point that the user profile record for the user and all the associated Orders and Preference records are brought into the cache group so that all subsequent references by that user will benefit from in-memory processing (with a penalty on the first reference only). The data to be referenced must be identified by an equality predicate on the primary key of the root table (in this case, by the user profile Id). The root table row and related child table rows are referred to as a *cache instance*.

For dynamically loaded cache groups the user can set one of two different aging policies to remove data to ensure that the database does not run out of space. TimesTen supports time-based aging (age on the value of a timestamp column) or LRU aging, based on recency of usage. When the aging mechanism removes rows from a cache group, it removes them in units of cache instances. Cache instances form the unit of caching since they are atomic units for cache replacement.

Cache groups can be of two basic types in order to handle a variety of caching scenarios. There are two corresponding data synchronization mechanisms for these cache group types.

Read-Only Cache Groups: For data that is infrequently updated, but widely read, a read-only cache group can be created on TimesTen to offload the backend Oracle database. Very hot reference data, as online catalogs, airline gate arrival/departure information, etc. is a candidate for this type of caching. The Oracle side tables corresponding to Read-Only cache groups are updated on Oracle. The updates are periodically refreshed into TimesTen using an automatic refresh mechanism.

Updatable Cache Groups: For frequently updated data, an updatable cache group with write-through synchronization is appropriate. Account balance information for an online ecommerce application, the location of subscribers in a cellular network, streaming sensor data, etc. are all candidates for write-through caching. TimesTen provides a number of alternative mechanisms for propagating writes to Oracle, but the most commonly used and highest performing mechanism is referred to as *Asynchronous Write-through*, where the changes are replicated to Oracle using a log-based transport mechanism. This mechanism is also capable of applying changes to Oracle in parallel, in keeping with the parallel-everywhere design theme of the system.

It is possible to deploy multiple TimesTen cache databases against a single Oracle database. This architecture is referred to as the *In-Memory Database Cache Grid* (depicted in Figure 4). Cache groups can once again be classified into two categories of data visibility on a grid.

Local Cache Groups: The contents of a local cache group are not shared and are visible only to the grid member they are defined on. This type of cache group is useful when the data can be statically partitioned across grid members; for instance, different ranges of user profile Ids may be cached on different grid members.

Global Cache Groups: In many cases, an application cannot be statically partitioned and Global Cache Groups allow applications to transparently share cached contents across a grid of independent TimesTen databases. With this type of cache group, cache instances are migrated across the grid on reference. Only consistent (committed) changes are propagated across the grid. In our example, a user profile and the related records can be loaded into one grid member on initial login by the user. When the user disconnects and later logs in onto a different grid member, the records for the user profile and related child table entries are migrated from the first grid member to the second grid member. Thus, the contents of the global cache group are accessible from any location, via data shipping.

The cache grid also provides data sharing via function shipping or global query. A global query is a query executed in parallel across multiple grid members. For example, if a global query needs to compute a COUNT(*) on the user profiles table, it would ship the count operation to all the grid members, and then collate the results by summing the received counts. This mechanism can be generalized to much more complex queries involving joins, aggregations, and groupings. Global queries are useful for running reporting operations on a grid, for example, what is the average value of the current outstanding orders for all users on the grid.

5 Experimental Results

5.1 Response time and throughput

The standard TimesTen throughput performance benchmark (TPTBM) models a telecommunication workload. The combination of in-memory execution and direct-linked access allow TimesTen to provide extremely low response times, see Table 1. This experiment was run on a standard commodity Intel Xeon 5670- system with 2 processors and 12 cores, running Oracle Enterprise Linux 5.6.

Scale up on multi-core systems is also evidenced by the throughput data, in which we measure peak throughput for a number of different workload mixes. These measurements were taken on a SPARC T5-8 system with 128 cores. On this system TimesTen achieves nearly 60 million TPTBM reads per second, and over 1 million TPTBM writes per second (this number is limited by the available IO bandwidth for logging and checkpointing). On a mobile call processing benchmark derived from a customer workload, TimesTen achieves 367K TPS on a SPARC T5-2 system with 32 cores.

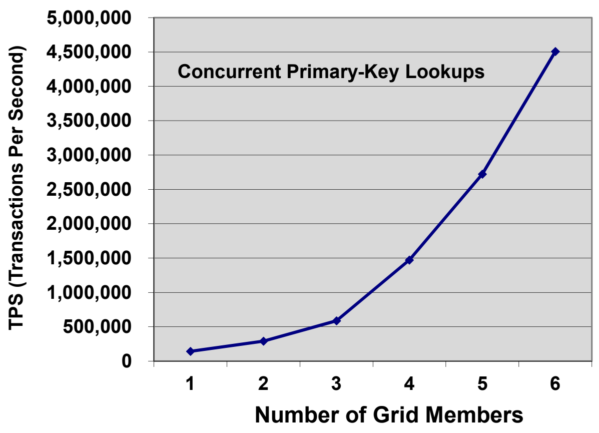
(a) Measured response time

Operation	**Response Time (microseconds)**
TPTBM Select	1.78
TPTBM Update	7.0

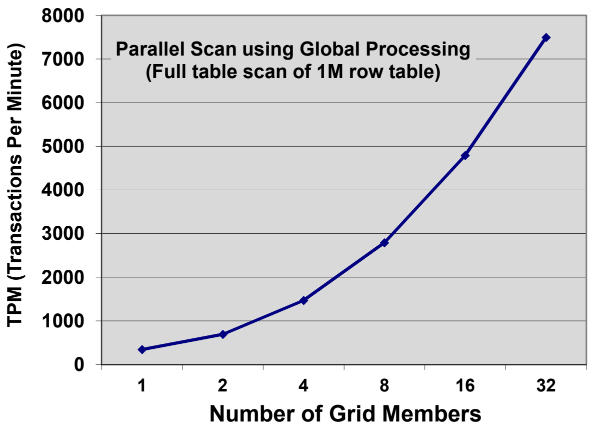
(b) Measured throughput

Benchmark	**Throughput (ops per second)**	**System (SPARC)**
TPTBM 100% read	59.9 million	T5-8, 128 cores
TPTBM 100% update	1.015 million	T5-8, 128 cores
Mobile call processing	367,000 txns	T5-2, 16 cores

Table 1: TimesTen response time and throughput



(a) Lookup on primary key



(b) Table scan

Figure 3: Concurrent lookup scaling on Cache Grid

5.2 Scaling across a grid

In this section, we show how TimesTen scales out when extended to the IMDB cache configuration. In Figure 3 (left graph), we show performance results from running concurrent primary key-based lookup queries across an IMDB grid simulated on a on older-generation Ultra SPARC T2 system using 2 CPU threads per grid member. As we can see that the grid scales very effectively as we increase the number of grid members. Here the key values are uniformly randomly distributed so that an increased percentage of the queries reference remote data as the number of nodes is increased.

Second, also in in Figure 3 (right graph), we show the performance of a global query that issues a full table scan on a million row table. In this test there are 6 application connections to one TimesTen grid member. The table being scanned is equi-partitioned across the grid databases. The scan is processed via global query execution in which the query is shipped to all grid members and the results compiled on the originating node. As shown below, the performance of global scans scales fairly linearly with the number of participating grid members.

5.3 Columnar compression

Next, we show the benefits of columnar compression on a database comprising aggregates computed from a CRM Data Warehouse star schema. This aggregated database has 29 tables. In Table 2, we show the distribution of compression factors achieved on individual tables in the schema. As we can see, 14 out of the 29 tables compress by more than a factor of 4x. The space savings from compression on this database is substantial: The total size of the tables in the database reduces from 19.5GB to around 3.16GB, or roughly a 6x reduction in size.

Compression factor	**Number of tables**
<2x	8
2x-4x	7
4x-6x	10
6x-8x	4

Table 2: Distribution of compression factors by table for CRM data warehouse aggregated star schema

6 Conclusions and Future Work

As we stated earlier, it is necessary for an in-memory database to provide a wide range of capabilities to be applicable to Enterprise workloads. We have shown that Oracle TimesTen is a full-featured relational in-memory database with ACID properties and transactional semantics. TimesTen has a storage manager optimized for in-memory access and for high concurrency. TimesTen has full-featured SQL support with extensions for analytics applications, stored procedures via the PL/SQL language, and access via standard database APIs. TimesTen provides full database as well as table-level multi-master replication for high availability, and can be deployed as a grid of cache databases to cache selective data from an underlying Oracle database. We have also shown performance results showing that TimesTen provides very low response times as well as very good scaling characteristics, both on a single system as well as across multiple systems on a cache grid. It is because of this comprehensive list of capabilities that Oracle TimesTen is the most widely deployed commercially available in-memory database product today, with a wide range of usages ranging from high performance in-memory OLTP workloads to high-speed in-memory Analytics.

Future work for TimesTen includes extending the capabilities of the database to large scale-out clusters, instantaneous restart using persistent RAM technologies such as Phase Change Memory, continued optimization for high performance network technologies such as Infiniband, and optimistic, lockless algorithms for maximal scaling on very large SMP systems.

7 Acknowledgments

Oracle TimesTen is a complex and sophisticated team-product, and the authors are merely scribes for the functionality in the product. Many thanks are due to the Oracle TimesTen Engineering team: Development, QA, Documentation, and Product Management, for their tireless contributions to the product.

We owe our thanks to the Oracle RDBMS development teams particularly in the areas of NLS support, OCI/Pro*C support and PL/SQL support, in helping Oracle TimesTen to adopt these enterprise critical features.

We are indebted to our partner teams within Oracle such as the Oracle CGBU Billing and Revenue Management team for contributing valuable ideas to the IMDB cache grid architecture, and to the Oracle BI/EE team for their many contributions to the analytics support in TimesTen.

Last, but most importantly, we owe our thanks to the thousands of customers who have adopted TimesTen and provided the most valuable feedback possible for our product based on real-world usage-based requirements.

References

[1] *MySQL Cluster*. http://www.mysql.com/cluster.

[2] O. America. *Using Oracle In-Memory Database Cache to Accelerate the Oracle Database. An Oracle Technical White Paper.* http://www.oracle.com/technetwork/database/focus-areas/performance/wp-imdb-cache-130299.pdf, June 2009.

[3] R. Kallman, H. Kimura, J. Natkins, A. Pavlo, A. Rasin, S. B. Zdonik, E. P. C. Jones, S. Madden, M. Stonebraker, Y. Zhang, J. Hugg, and D. J. Abadi. H-store: a high-performance, distributed main memory transaction processing system. *PVLDB*, 1(2):1496–1499, 2008.

[4] S. Manegold, P. A. Boncz, and M. L. Kersten. Optimizing database architecture for the new bottleneck: memory access. *VLDB J.*, 9(3):231–246, 2000.

[5] H. Plattner. A common database approach for OLTP and OLAP using an in-memory column database. In *SIGMOD Conference*, pages 1–2, 2009.