Extreme Performance Using Oracle TimesTen In-Memory Database

ORACLE®

Table of Contents

Introduction

The Oracle TimesTen In-Memory Database is a memory-optimized relational database that delivers very low response time and very high throughput for performance-critical systems. It is targeted to run in the application tier, close to applications, and optionally in process with applications. It can be used as the database of record or as a cache to the Oracle Database.

This paper describes the Oracle TimesTen In-Memory Database, and gives a high-level description of how it may be used as a cache to the Oracle Database. For a more complete description of how to use TimesTen as a cache to the Oracle Database, see[1].

Milliseconds Matter

Oracle TimesTen provides application-tier data management for performance-critical systems, and is optimized for blazing-fast response and real-time caching of Oracle data. Companies can extend their software infrastructures with Oracle TimesTen to create systems that are instantly responsive, highly scalable, and continuously available. These systems are used to increase customer loyalty, attract new customers, streamline operations, and avoid the costly alternative of proprietary software development

Oracle TimesTen has a proven track record, with production deployments since 1998 in real-time enterprises and time-critical environments such as network telecommunication services, operational support systems, contact centers, airline and reservation systems, command and control systems, and securities trading. Thousands of companies worldwide use Oracle TimesTen in production applications—including Alcatel-Lucent, Aspect, Avaya, Bank of America Merrill Lynch, Bridgewater Systems, BroadSoft, Cisco, Deutsche Börse, Ericsson, JPMorgan, KDDI, NEC, NYFIX, Smart Communications, United States Postal Office and Verizon Wireless.

The Growth of Real-Time Applications

Network equipment manufacturers, telecom operators, securities exchanges and brokerages, airlines, shipping and logistics companies, and defense and intelligence agencies are examples of enterprises in which real-time applications are a necessity. The use of real-time processing to capture, analyze, and respond intelligently to key events is increasingly becoming the benchmark for corporate excellence.

Real-Time Industries

For many companies, real-time applications aren't elective—they are a necessity. Network equipment manufacturers, telecom operators, securities exchanges and brokerages, airlines, shipping and logistics companies, and defense and intelligence agencies are prime examples of enterprises that require real-time applications. In the past, building these applications also required the development of real-time infrastructure software. Fast but inflexible, these systems did the job as long as the applications remained static in their requirements. But dynamic industries quickly outpace static applications, and the cost to develop, test, and maintain specialized infrastructure software is rarely justified when commercial alternatives exist.

---

1 Using Oracle TimesTen Application-Tier Database Cache to Accelerate the Oracle Database. An Oracle White Paper, June 2014

Real-Time Enterprises

With the growing speed of messages moving through business networks, the use of real-time processing to capture, analyze, and respond intelligently to key events is becoming the benchmark for corporate excellence. This isn't important only for the execution and management of critical business processes. Customers expect highly tailored interactions and the utmost responsiveness from any company with which they do significant business.

Business activity monitoring, complex event processing, RFID/sensor-based applications, Web portals, and Web services are contributing to the movement of applications to the edge of the enterprise. Configured as dynamic collections of interrelated components, these applications are part of an overall approach known as a service-oriented architecture (SOA). However, most data sources still reside in the back office, dominated by large amounts of rarely touched legacy data surrounding smaller amounts of currently active information. A natural extension of SOA concepts includes lightweight, real-time data management in the application tier, connected to corporate data sources to provide real-time performance for currently active data.

Real-Time Data Management Software

The enterprise architectures that derive the greatest benefit from real-time processing provide event, data, and transaction management in the application tier, empowering front-line systems with rapid response and deeper insight. It is not sufficient to merely collect and cache data next to applications, as is often the case with first-generation in-house efforts. Nor is it practical to locate the corporate database on the same platform as one of the applications.

What's needed is a generation of lightweight infrastructure software that presents familiar, powerful interfaces and query languages that are widely in use—that can easily interface with existing back-office databases, messaging systems, and application servers—and that exploits the full performance potential of today's networked, memory-rich computing platforms. This is the generation of infrastructure software for real-time data management provided by the Oracle TimesTen In-Memory Database.

Application-Tier Deployment

Much of today's new application development is focused on improving interactions with customers and streamlining internal operations to eliminate delays and excess costs. These are real-time applications that reside near the network edge or, in some cases, within the network as hosted services.

This is the new enterprise application tier, with different platform, performance, and availability requirements than legacy back-office applications. Business events are embodied in network messages to which applications subscribe, triggering real-time processing and additional message publishing.

To meet the response time and scalability goals for these applications, it's often necessary to deploy the infrastructure software and applications on the same platform, including some or all the data that drives the application.

Oracle TimesTen is designed to integrate seamlessly into these environments, with an architecture that is optimized for application-tier deployment and configuration options that enable highly tailored solutions.

Oracle TimesTen In-Memory Database

Oracle TimesTen In-Memory Database (TimesTen) is a memory-optimized relational database that delivers to applications the instant responsiveness and very high throughput required by real-time enterprises and industries.

TimesTen is typically deployed in the application tier. TimesTen databases fit entirely in physical memory. They are persistent and recoverable, and access to them is provided via standard SQL interfaces.

TimesTen databases may be replicated between servers for high availability and load sharing. Data replication configurations can be set to active/standby or active/active, using asynchronous or synchronous transmission, with conflict detection and resolution, and automatic failover, recovery, and resynchronization after a failed server is restored.

The Oracle TimesTen Application-Tier Database Cache (TimesTen Cache) is the pre-integrated SQL database cache option to the Oracle Database. It caches real-time, updatable subsets of an Oracle database in TimesTen in the application tier. It automatically synchronizes data between the caches and the Oracle database. It offloads computing cycles from back-end Oracle Database servers and enables remarkably responsive and scalable real-time applications. The TimesTen Cache automates pass-through of SQL requests for non-cached data, and automatically resynchronizes data after failures.

In-Memory Database Technology

In-memory database technology implements a relational database in which all data at runtime resides in the RAM. The data structures and access algorithms exploit this property for breakthrough performance.

In-memory database (IMDB) technology is the foundation technology for TimesTen. IMDB technology implements a relational database in which all data at runtime resides in RAM, and the data structures and access algorithms exploit this property for breakthrough performance. Compared to a fully cached RDBMS, IMDB technology requires far less processing power, because the overhead to manage memory buffers and account for multiple data locations (disk and memory) is eliminated. With IMDB technology, disks are used for persistence and recovery rather than as the primary database storage location.

The memory-optimized performance of TimesTen is complemented by transactional properties, persistence mechanisms, and recovery from system failures. A variety of choices is available for locking, multiuser isolation, and logging, accommodating a range of application scenarios from transient look-up caches to core transactional trading and billing systems.

Durability is achieved by logging the changes from committed transactions to disk and periodically updating a disk image of the database, termed a "checkpoint." The timing of the disk write for the log is configurable by the application; it can be either synchronous with the end of the transaction or deferred until after. Many situations favor higher throughput over synchronous logging, particularly when the monetary value of a transaction is low or the transaction data is short-lived, such as when tracking the location of mobile phones that communicate their cell location every few seconds.

The interfaces supported by TimesTen are standards-compliant. Applications issue SQL and PL/SQL commands using ODBC, JDBC, ADO.NET, the Oracle Call Interface (OCI), or TTClasses, as well as Pro*C. TimesTen C++ Interface Classes (TTClasses) is a C++ class library that provides wrappers around the most common ODBC functionality. It is easier to use than ODBC and promotes best practices while maintaining fast performance. The statements for defining databases, replication configurations, and cache groups also adhere to SQL syntax conventions. Simple network management protocol is used to issue standardized system management alerts.

An open transaction log API (XLA) with a standard Java message service (JMS) interface is provided for reading the transaction log. This is useful for creating applications that react to database updates. In this regard, XLA is a lightweight trigger. It's also a way to build custom data replication from Oracle TimesTen products to other database systems.

The Physical Structures of Oracle TimesTen

This section describes the system components of TimesTen. TimesTen consists of

» Shared libraries

» Memory-resident data structures

» System processes

» Administrative programs

» Checkpoint files and log files on disk

Application-Tier Shared Libraries

The components that implement the functionality of TimesTen are embodied in a set of shared libraries that developers link to their applications and execute as a part of the application's process. This shared library approach is in contrast to a more conventional RDBMS, which is implemented as a collection of executable programs that applications connect to, typically over a client/server network.

Normally, embedding data manager libraries into the application could make the database vulnerable to corruption if the application process were terminated abnormally. TimesTen solved this challenge. Using a patented algorithm known as MicroLogging, TimesTen libraries self-protect against application process failures. In-memory databases remain consistent, and other applications continue without impact.
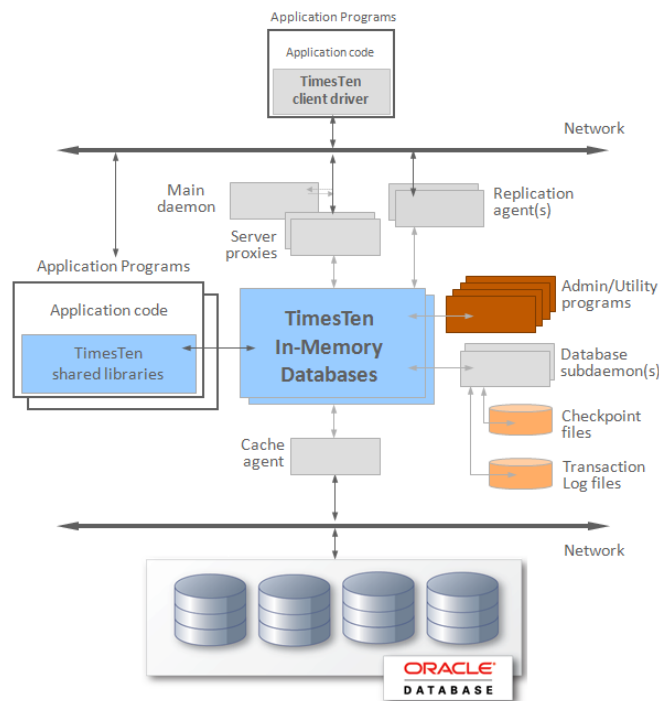

Figure 1. Components of the Oracle TimesTen In-Memory Database

IMDB technology is implemented as in-memory databases accessed by applications, utility programs, and system processes via shared library routines. Disk files for logs and checkpoints (backup copies) are maintained for recovery purposes.

Applications may use a client/server connection to access an Oracle TimesTen database, though in most cases the best performance will be realized with a directly linked application.

Memory-Resident Data Structures

In-memory databases are maintained in an operating system's shared memory segments, and contain all user data, indexes, system catalogs, log buffers, lock tables, and temp space. Multiple applications can share a TimesTen database, and a single application can access multiple TimesTen databases on the same system.

Memory-resident databases are maintained in an operating system's shared memory segments and contain user data, indexes, system catalogs, log buffers, lock tables, and temp space. Multiple applications can share a database, and a single application can access multiple databases.
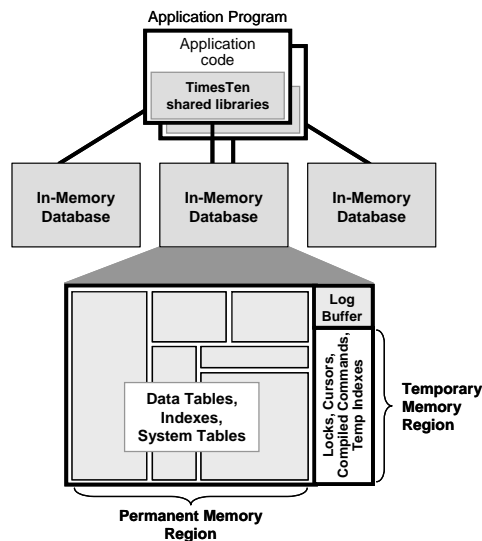

Figure 2. Oracle TimesTen in-memory database

System Processes

Background processes provide services for startup, shutdown, and application failure detection at the system level, and provide loading, checkpointing, and deadlock handling at the database level. There is one instance-wide TimesTen daemon (a system may have multiple instances, i.e., multiple installations of the TimesTen software), and a separate subdaemon for each database.

Administrative Programs

Utility programs are explicitly invoked by users, scripts, and applications to perform services such as interactive SQL, bulk copy, backup/restore, database migration, and system monitoring.

Checkpoint and Log Files

Changes to the database and transaction logs are written to disk periodically. Should a database need to be recovered, TimesTen merges the database checkpoint on disk with the completed transactions that are still in the log files. Normal disk file systems are used for checkpoints and log files.

Data Replication Technology

When high availability or workload distribution is desired, data replication can be configured to send updates between two or more servers. A master server is configured to send updates, and a subscriber server is configured to receive them, and a server can be both a master and a subscriber for bidirectional replication. Time-based conflict detection and resolution is used to establish precedence in the rare event of the same data being updated in multiple locations at the same time.

When replication is configured, a replication agent process is started for each database. If multiple databases on the same server are configured for replication, there is a separate replication agent for each database. Each replication agent can send updates to one or more subscribers, and receive updates from one or more masters. Each of these connections is implemented as a separate thread of execution inside the replication agent process. Replication agents communicate through TCP/IP stream sockets.

For maximum performance, the replication agent detects updates to a database by monitoring the existing transaction log, and sends updates to the subscribers in batches if possible. Only committed transactions are replicated. On the subscriber node, the replication agent updates the database though an efficient low-level interface, avoiding the overhead of the SQL layer.

Caching Technology

When TimesTen Application-Tier Database Cache (TimesTen Cache) is used to cache portions of an Oracle Database, one or more cache groups are created to hold the cached data, and a system agent (the cache agent) performs all asynchronous data transfers between the cache and Oracle Database.

A cache group is a collection of one or more tables arranged in a logical hierarchy via primary/foreign key relationships. Each table in a cache group is related to an Oracle Database table. A cache group table can contain all or a subset of the rows and columns in the related Oracle Database table. Cache groups can be created and modified via SQL and PL/SQL statements. Cache groups support the following features:

- Applications can both read from and write to tables in the cache groups.

- Cache groups can be refreshed from an Oracle database automatically or manually.

- Updates to cache groups can be propagated to an Oracle database automatically or manually.

- Changes to either Oracle Database tables or cache groups can be tracked automatically.

When rows in a cache group are updated by applications, the corresponding rows in Oracle Database tables are updated synchronously as part of the same transaction, or asynchronously immediately afterward depending on the type of cache group that was created. Changes that originate in Oracle Database are refreshed into the cache via the cache agent.

Cache Grids are available for horizontal scalability in performance and capacity where a Cache Grid consists of a collection of TimesTen Caches that collectively manage an application's cached data. Cached data is distributed between grid members, and the Cache Grid provides applications with location transparency. Cache Grids enable

incremental scalability through the online addition (and removal) of grid members. They maintain consistency of cached data between the Cache Grid members and the Oracle Database.

For more information on the TimesTen Application-Tier Database Cache, see [2].

IMDB Technology in Depth

By managing all data in memory and optimizing for that environment, in-memory database technology can operate much more efficiently, and thus offer dramatic improvements in performance.

In-memory database technology delivers impressive performance by changing the assumptions about where data resides at runtime. By managing all data in memory and optimizing for that environment, in-memory database technology can operate much more efficiently, and thus offer dramatic improvements in responsiveness and throughput.

But what's at the heart of this performance? Couldn't an application get the same results just by placing all its RDBMS data in main memory?

Much of the work that is done by an RDBMS is done under the assumption that data is primarily on disk. Oracle TimesTen, on the other hand, assumes the data resides in main memory and can therefore take more direct routes to it, reducing code-path length and simplifying both algorithm and structure.

In comparing these architectures, there are a number of key differences that clearly illustrate how a many-fold performance improvement is attainable. To illustrate, just a few of these differences can be found in query optimization algorithms, indexing, query processing, and buffer pool management.

Query Optimization

Because disk input/output is far more expensive than memory access, disk-based RDBMSs have to assume that data resides on disk.

Query optimization algorithms are different for disk-based systems than for memory-based systems. RDBMS optimization decisions are based on the assumption that data resides primarily on disk. In a dynamic runtime environment, data might be on disk or cached in main-memory at any given moment. Because disk input/output (I/O) is far more expensive than memory access, disk-based RDBMSs have to reduce the probability of access to disk as much as possible. They are optimized to reduce the point of performance bottleneck, so a disk-based optimizer will not always produce the optimal plan for data that resides—primarily or fully—in main memory.

IMDB technology, on the other hand, knows that the data resides in main memory and optimizes its queries under simpler assumptions. It does not need to make a worst-case scenario based on disk residency, and therefore, its cost estimates can be simpler and more consistently accurate.

---

2 Oracle TimesTen Application-Tier Database Cache to Accelerate the Oracle Database. An Oracle White Paper, June 2014

Indexing

Indexes are as important to the performance of memory-based systems as they are for disk-based systems. Indexes are used for a variety of purposes during query processing, such as quickly qualifying rows in a SQL SELECT, UPDATE or DELETE statement, finding the minimum or maximum value of a given column or set of columns, or speeding up join processing. Indexes can also provide an ordered stream of records during query processing. Such ordered streams can eliminate the need for additional sort operations, or allow for fast duplicate elimination.

For all of these similarities with indexes in disk-based RDBMs, indexes can have significant differences with IMDB technology. Where each index entry in a disk-based RDBMS index usually contains an index key and a record identifier, which enables the RDBMS to locate the record on disk, with IMDB technology, keys do not need to be stored in the indexes, and record identifiers can be implemented as record pointers. A record pointer in an index entry points to the corresponding record that contains the key. By avoiding the duplication of key values in the index structure, IMDB technology greatly reduces the size of each index. The absence of key values can also lead to a simpler index implementation because it avoids the need to manage variable-size keys within the index nodes.

Query Processing

IMDB technology takes advantage of memory residency of all data. For example, the fastest way to execute a query with an ORDER BY clause in a disk-based RDBMS is if the base table happens to be stored in sorted order using the same column as in the ORDER BY clause. That is of course very unlikely. The next best case is if the order can be satisfied using an index scan. However, retrieving the data via an index scan results in random access to the records, and potentially results in very high I/O cost. It may even require that the same data block be read from disk multiple times. Since the base table can only be stored in sorted order via one set of columns, the order in which the records are stored can be only used to satisfy ORDER BY queries where the ORDER BY columns are a prefix of the sorting columns. This is not an issue with IMDB technology because index entries point directly to the memory address where the data resides, so random scan via an index is as cheap as a sequential scan.

Similarly, when implementing a sort function in a disk-based system, one has to decide whether the sort structure will contain entire or partial records, key/record identifier pairs, or only record identifiers. This is a space/performance tradeoff. If the attributes of interest are not stored in the sort structure, random access with potential I/O overhead will be needed to retrieve the data. By contrast, memory-based systems do not have this issue because data can be accessed directly via tuple pointers so the sort structure needs to contain only the tuple pointers.

Buffer Pool Management

Although necessary in disk-based data management solutions, buffer pools become unnecessary in memory-based data management because the data already resides within memory.

In conventional RDBMS architectures, buffer pools must be maintained for data that has been cached in main memory. When the query processor requires a page of data, it must first search the buffer pool for that data, and even if the data is there, in many cases it must be copied out of the pool for processing. This buffer pool maintenance and management, coupled with additional data copies, add significantly to the original burden of making the data available to an application.

Although necessary for disk-based databases, buffer pools are unnecessary for in-memory databases. The data already resides within main memory, so TimesTen has no buffer pool. Therefore, code path length and engine footprint are reduced, copying is avoided, algorithms are simplified, and data is delivered to the application more quickly.

The Benefits of the Difference

When the assumption of data residing on disk (or rather, the ambiguity of data location) is removed, complexity is dramatically reduced. The number of machine instructions drops considerably, buffer pool management disappears, extra data copies aren't needed, index pages shrink, and their structure is simplified. When data residing in memory is the bedrock assumption, everything gets simpler, more compact, and faster.

A New Look at Price Performance

For many applications, leveraging in-memory database technology offers a new way to gain a competitive edge, improve user satisfaction, and boost return on investment.

Performance improvements can be measured in multiples rather than fractions. An application that spends half its processing time managing data and half its time within its application space or business logic can nearly double its capacity by using an in-memory data management product. This dramatically increases the options in terms of market strategies, architectural economies, and system choice.

Exceptional Performance

Scalability

TimesTen takes advantage of multiple CPUs on symmetric multiprocessor computers. Figure 3 shows the transaction throughput of TimesTen on an Intel E5-2680 @2.7GHz 2 sockets 8cores/socket system running Oracle Linux. Each transaction executes a single SQL select (read) or update operation, as indicated. The results are shown for 1, 4, 8 and 16 concurrent processes.
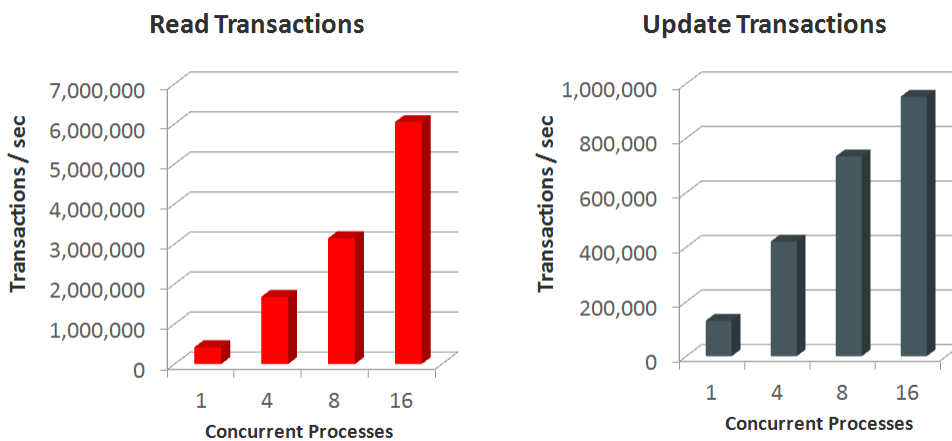

Figure 3. The throughput of TimesTen

The simplest and therefore highest volume workload is the "100% reads" test, which measures how many individual records can be retrieved using a key value lookup. With 16 concurrent processes, 6,050,188 reads were completed every second, on average. With only one process, the read rate was 421,795 per second.

Update operations require more processing than read operations, partly because changes must be logged for recovery purposes. In this result, more than 952,778 records were updated per second with 16 concurrent processes, and more than 130,375 with a single process.

Oracle TimesTen products are capable of exceptional throughput, ranging from tens of thousands of update transactions to hundreds of thousands of read operations per second, when the transaction contains a single SQL operation.

Although these pure workloads aren't representative of any particular business application, these results validate the extreme efficiency of TimesTen, and any realistic mixture of SQL operations would likely result in a maximum throughput in the tens-of-thousands-per-second range. If higher volumes are required, TimesTen scale well beyond a 16-way system. Note that performance results vary on different processor platforms.

Response Time

Throughput is a byproduct of individual transaction response times. As throughput increases, average response time decreases. The exceptional throughput of Oracle TimesTen products yield microsecond-level response times on systems across a range of data management workloads. The average response times for the throughput workloads previously described are shown in Figure 4.

Oracle TimesTen products' response time is measured in microseconds for single-record operations. Data-intensive applications will benefit the most, since overall response time for the application includes more than just the data management portion.

Average Response Time
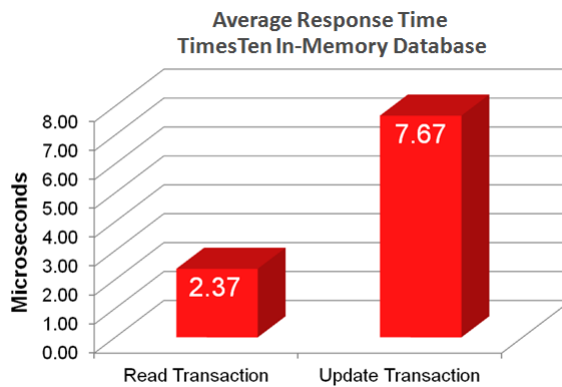**TimesTen In-Memory Database**

Figure 4. Latency in Oracle TimesTen products

It's important to keep in mind that a transaction's response time is the aggregate of the response times of the constituent processing steps, such as business logic, networking, I/O, and data management. The performance of TimesTen's portion is not the only determinant of overall response time. Sometimes one of the steps, usually I/O or networking, will dominate the others.

For instance, if client/server network is used instead of a direct connection between the application and the data manager, an additional millisecond or more of response time can be added for each round trip over the network. Similarly, if synchronous I/O is used for logging changes, the relative slowness of a disk operation is included in the response time. In both cases, if the data management component is completed within microseconds, the overall application response time is still going to be a factor of the network and I/O latencies.

For applications that have a dominant data management component, the speedup of TimesTen technology could achieve a major decrease in response time. The highest-performing application is one that is directly connected, able to use asynchronous logging, and is data management–intensive. Search-intensive online reservation systems, presence, and location-based communication services and contact center routing engines are good examples.

As always, performance results vary greatly depending on many factors, such as the size of a transaction, the amount of non-data access logic, and platform performance.

Real-Time Functionality

This section presents a functional view of the Oracle TimesTen In-Memory Database, describing the interfaces and features used by developers when writing real-time applications.

Data Management

The data manager feature in TimesTen implements a true relational database model. It is not a hybrid model or a different model with a relational view grafted on top. Developers who are familiar with popular RDBMS products should be immediately productive.

Relational Database Model

In the relational model, user data is stored as rows (records) within tables. A row is a list of columns (data fields), each of a specified data type. Indexes are explicitly or implicitly created to speed key-based or value-range searches of the tables. Oracle TimesTen products support both hash indexes and range indexes. Defined constraints, such as uniqueness or referential integrity (parent/child relationships between tables), are enforced automatically.

Within a table, a primary key—comprised of one or more columns—may be designated. A primary key is the unique identifier for a row, and Oracle TimesTen automatically prevents the insertion of additional rows with the same primary key value. One or more foreign keys may also be designated per table, each associated with the primary key of another table. Foreign keys enforce a parent/child relationship between two tables. For example, a row with a primary key will not be deleted if an associated foreign key in another table has the same value.

A view is a virtual table whose content consists of the result of the query that makes up the view definition. Views can be included in queries just as ordinary tables would be.

A materialized view is a read-only table derived from regular (base) tables. It can include parts or all of the data from one or more tables, and can include calculated values (such as totals and averages). Since materialized views are automatically updated by the system whenever necessitated by a change to a base table, they provide applications with faster access to frequently derived data. Without materialized views, the overhead of joins and other calculations would be incurred whenever an application needed to read the data. Materialized views are especially useful in support of event processing (discussed in more detail later in this article).

If the TimesTen Cache option is used, one or more cache groups can be defined in an in-memory database. A cache group is a hierarchical set of tables (enforced through foreign keys) that map to a corresponding set of Oracle Database tables, and contain some or all of the same data. Cache groups are both read and write, and updates to the data can propagate automatically between TimesTen and the Oracle Database.

Data management operations are expressed via the industry-standard Structured Query Language (SQL), and via PL/SQL and Pro*C. Tables, indexes, views, materialized views, and cache groups are created and maintained within the databases. Applications connect to a TimesTen database or to a TimesTen Cache database and perform operations against the data within. Multiple databases may exist on the same computer system and be accessed by the same application. However, no single operation (for example, a join across two tables) may span more than one database. Applications can participate in a distributed transaction through TimesTen's support of the standard XA (or JTA for Java) interface for two-phase commit.

Durability and Concurrency

TimesTen conforms to the atomicity, consistency, isolation, and durability properties of data management systems.

TimesTen transactions conform to the atomicity, consistency, isolation, and durability properties. These properties ensure that, in a multiuser system, each transaction operates as if it were the only transaction being executed at the time, and that the system can guarantee that the effects of a committed transaction are not lost. These are the most rigid properties required of data managers, and TimesTen ensures full conformance.

A common misperception is that in-memory data managers cannot prevent data loss from system failures. In fact, the same techniques that make transactions and data durable in a conventional database are used in TimesTen. As in all transaction-oriented systems, durability is achieved through a combination of change logging and periodic refreshes of a version of the database that resides on a disk.

TimesTen offers applications control as a tradeoff for the degree of durability and the total throughput. More of one means less of the other, and TimesTen provides choices along this spectrum.

Disk Operations

Oracle TimesTen databases may be permanent or temporary, and for exclusive or shared access.

If an application requires that no changes be lost, log records are flushed to disk as part of committing the transaction. If maximum performance is more important than possibly losing some transactions, log records can be written to disk less often, asynchronously from each transaction commit. In either case, TimesTen will attempt to "group commit" multiple transactions together to minimize disk writes.

Periodically, the database is checkpointed to disk automatically based on user-specified frequency and log volume. TimesTen supports "fuzzy" checkpoint operations that run in the background with minimal impact on running applications. Recovery from a system failure is a matter of merging log records with the latest checkpoint file.

TimesTen databases may be permanent or temporary. Permanent databases (checkpoint files) remain on disk when not in use. Temporary databases reside solely in memory and are removed when not in use. Permanent databases are the most commonly deployed configurations. Temporary databases are used by applications for which the data is so transient that there is no value in recovering the database after a system failure, for example, highly dynamic state information in a call center application.

Transaction logging enables the recovery of transactions against persistent databases after a system failure, and the rollback of transactions.

Locking and Isolation

TimesTen applies locks to prevent a user from changing data that is currently being read or modified by another user. A lock is placed on database objects at the database, table, or row level. Row-level locking provides the greatest multiuser concurrency, and is the default.

Applications can call a procedure to change the locking level to row or database during runtime. Table-level locking is used when the TimesTen optimizer determines that it's advantageous, or when the application calls a procedure that directs the optimizer to apply table-level locking for the duration of a transaction.

Users can select an isolation level to specify the behavior of locks that are applied in response to read operations. The two isolation levels are serializable or read-committed (the default).

Applications select either the serializable or read-committed isolation level. Read-committed isolation provides the greatest multiuser concurrency. It guarantees that readers will only see committed data values, but doesn't wait for writers to release or place locks on records that are being read.

Serializable isolation is used for transactions that demand consistent, predictable data values through the life of a transaction. With serializable isolation, records that are read are prevented from being updated or deleted by other users until the transaction commits or is rolled back. During this time, other users aren't allowed to insert new records if they would be part of the result set from these read operations. In other words, serializable isolation guarantees that a read operation could be repeated with the same results. With database-level locking, transactions are effectively operating with serializable isolation by default.

Read-committed isolation provides the greatest multiuser concurrency. It guarantees that readers will see only committed data values, but doesn't wait for writers to release or place locks on records that are being read. TimesTen creates two copies of a record that is being updated: the preupdated and thus "committed" values for readers, and the updatable version for the writer. Readers are not blocked access to the read version, and writers don't wait for readers.

Query Processing

*A primary objective of TimesTen is to adopt relevant open standards. Applications can communicate with TimesTen databases through SQL, PL/SQL, Pro*C, JDBC, ODBC, TTClasses, JMS, and OCI interfaces.*

Most specialty products designed for high performance require proprietary APIs. Oracle TimesTen does not. The only way to access an Oracle TimesTen database is through standard languages and interfaces and through Oracle's popular languages and interfaces. In particular, Oracle TimesTen databases are accessed through: the structured query language (SQL), the procedural language/structured query language (PL/SQL), Pro*C, Java database connectivity (JDBC), open database connectivity (ODBC), TTClasses, Java Message Service (JMS), and the Oracle Call Interface (OCI). The implementation of these interfaces has been highly tuned for the architecture of TimesTen products.

SQL has been widely adopted for years as the query language standard for relational databases. Abstraction from the underlying storage and indexing details is one of the main benefits of SQL. It's also an easy-to-use language

that expresses which action is to be done, rather than how to perform it. SQL has no references to indexes, data types, or physical layouts—just tables, columns, and search conditions. The optimizer feature in TimesTen determines the fastest way to respond to a query based on factors such as the presence of indexes and the distribution of key values.

This level of abstraction allows the underlying data model to be tuned or extended without affecting existing applications. New services can be quickly added into a production environment simply by adding application modules and any required data tables and columns. Without a query language such as SQL shielding the application from the internals of the data manager, most applications would be affected by the addition of new data fields.

Data Replication

Replication is configured through SQL statements and can apply to designated tables or an entire database. To enable high efficiency and low overhead, Oracle TimesTen products use a transaction-log-based replication scheme.

Replication is the process of copying data between databases. This can help make data continuously available to mission-critical applications with minimal performance impact. In addition to its role in failure recovery, replication is also useful for distributing user loads across multiple databases for maximum performance and for facilitating online upgrades and maintenance.

TimesTen follows a master/subscriber replication model in which committed changes are copied from their source to one or more subscriber databases. Replication is configured through SQL statements and can apply to designated tables or an entire database. To enable high efficiency and low overhead, a transaction-log based replication scheme is used.

Replication at each master and subscriber database is controlled by replication agents that communicate through TCP/IP stream sockets. The replication agent on the master database reads the records from its transaction log and forwards any detected changes to the replication agent on the subscriber database. The replication agent on the subscriber then applies the updates to its local database. If the subscriber agent is not running when the updates are forwarded by the master, the master retains the updates in its transaction log until they can be applied by the subscriber.

Balancing Performance and Consistency

The master and subscriber databases have internal mechanisms to confirm that the updates have been successfully received and committed by the subscriber. These mechanisms, which are completely invisible to the applications, ensure that updates are not lost and are applied by a subscriber only once.

The replication mechanism in TimesTen is by default asynchronous. When using asynchronous replication, an application updates a master database and continues working without waiting for the updates to be received by the subscribers. The master and subscriber databases have internal mechanisms to confirm that the updates have been successfully received and committed by the subscriber. These mechanisms, which are completely invisible to the application, ensure that updates are not lost and are applied by a subscriber only once.

Asynchronous replication provides best performance, but the application is decoupled from the receipt process of the replicated elements on the subscriber. TimesTen also provides two return service options for applications to confirm that the replicated data is consistent between the master and subscriber databases.

» The return receipt service loosely couples or synchronizes the application with the replication mechanism by blocking the application until replication confirms that the update has been received by the subscriber

» The return twosafe service enables fully synchronous replication by blocking the application until replication confirms that the update has been both received and committed by the subscriber

Applications that use the return services trade some performance to ensure higher levels of data integrity and consistency between the master and subscriber databases.

Replication Topologies

TimesTen supports many replication topologies, but recommends the active standby pair configuration for highest availability. This is also the only configuration that is supported with the TimesTen Cache.
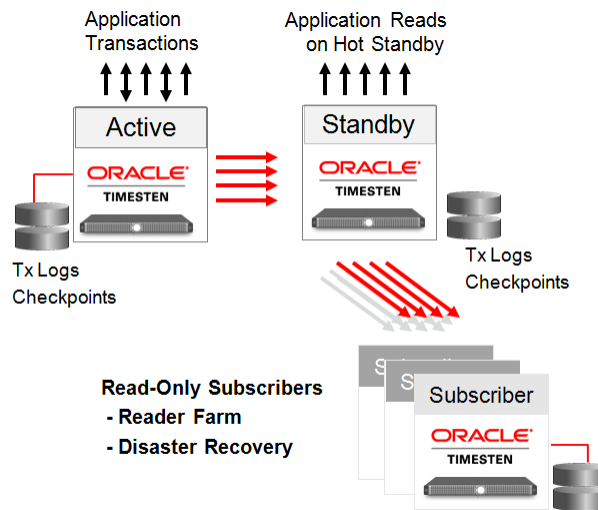
Figure 5. Active Standby Pair Replication

An active standby pair includes an active master database, a standby master database, and optional read-only subscriber databases. The active master database is updated directly. The standby master database cannot be updated directly. It receives the updates from the active master database and propagates the changes to read-only subscriber databases. This arrangement ensures that the standby master database is always ahead of the read-only subscriber databases and enables rapid failover to the standby database if the active master database fails.

Only one of the master databases can function as an active master database at a specific time. If the active master database fails, the role of the standby master database must be changed to active before recovering the failed database as a standby database. The replication agent must be started on the new standby master database.

If the standby master database fails, the active master database replicates changes directly to the read-only subscribers. After the standby master database has recovered, it contacts the active standby database to receive any updates that have been sent to the read-only subscribers while the standby was down or was recovering. When the active and the standby master databases have been synchronized, then the standby resumes propagating changes to the subscribers.

Automatic failure detection and failover of database and applications is available through integration with Oracle Clusterware.

Active standby replication can be used with the Oracle TimesTen Application-Tier Database Cache to achieve cross-tier high availability. Active standby replication is available for both read-only and updatable caches.

"Leading service providers rely on our products to manage time-critical billing data, so bottlenecks and downtime cannot be tolerated. With Oracle TimesTen and Oracle Real Application Clusters, we gain a reliable, scalable, proven real-time data management foundation to better serve our customers."   *Ed McKee,*
Director of Applications, Interact, Inc.

Caching

Customized Integration

It's common for TimesTen to be deployed in conjunction with a disk-based RDBMS, so that TimesTen is used when data needs to be captured or processed in real time. As the data transitions to a non-real-time state (for example, when a stock trade is complete or a call detail record has been rated) the information is transferred from TimesTen to the back-end RDBMS. There are multiple ways in which this integration can be achieved.

Applications can connect to both TimesTen and the back-end database, and move the data through regular API requests. This is the most flexible approach, but is the least transparent to the applications and could require complex programming. For example, an application that needs to cache all active subscribers could first request a record in a TimesTen database, and if results are not found, connect to the back-end RDBMS and repeat the same request, and insert the results into the TimesTen database. If there were any updates to the data, they would need to be made in both databases by the application. However, changes made directly to the data in the back-end RDBMS could result in cache coherency issues.

A variation of this approach is to connect TimesTen and the back-end RDBMS through a publish-and-subscribe message bus, write new modules separate from existing applications that listen for changes, and then duplicate changes picked up from the bus. An application can use TimesTen's transaction log API (XLA) to register for and receive notice of updates. Most RDBMSs offer a trigger feature that can signal changes, or provide an API similar to XLA.

Oracle TimesTen Application-Tier Database Cache

A pre-integrated caching solution for the Oracle Database is through the Oracle TimesTen Application-Tier Database Cache (TimesTen Cache) option. This option enables an application to cache subsets of rows and columns from Oracle Database tables in the TimesTen Database. Cached data may be pre-loaded by the application, or may be faulted in the cache on demand. Cache data can be read and updated, and the TimesTen Cache synchronizes data between the two databases automatically.

A TimesTen cache grid is a collection of TimesTen Caches that collectively manage an application's cached data. A cache grid consists of one or more grid members each backed by a TimesTen Cache. Grid members cache tables from a central Oracle database or Real Application Cluster (RAC). Cached data is distributed across multiple nodes without shared storage. A cache grid provides applications with location transparency for cached data and ensures that data is consistent across nodes.

Applications access data via SQL or PL/SQL statements using one of the TimesTen supported APIs, namely ODBC, JDBC, ADO.NET, OCI, or TTClasses, a proprietary C++ interface.

Event Processing

Event processing applications can sense and respond to events deemed to have business relevance.

There are multiple ways in which Oracle TimesTen enables event-processing applications. At the most basic level, TimesTen is designed to support applications that must react in real-time to business events. The ability to co-exist with applications on a wide range of computing servers provides the flexibility to support event-processing applications.

However, simple events by themselves often do not reveal their relevance and must be correlated with other information or staged until a pattern is evident. In these cases, the event data must be captured in a data manager and possibly compared to reference data that was pulled from another data source. When an event that requires action is identified, taking the appropriate response must, in many cases, be instantaneous. Such processing is ideally suited to the functionality of TimesTen.

The TimesTen Transaction Log API (XLA) enables the detection of database updates. Applications use XLA to monitor changes in a TimesTen database and take actions based on those changes. Notification may be desirable when, for example

» The value of any customer's total trades for the day exceeds US$1,000,000
» A customer on the "A" list makes a purchase
» A prepaid balance is depleted
» A network element is taken offline
» A flight changes departure gates

Multiple applications can simultaneously read transaction log updates, and each application can maintain its own bookmark in the log file to maintain its position. Bookmarks are persistent across database connections, shutdowns, and system failures, so event notifications can pick up where they left off.

XLA is often used to build a custom data replication solution to a non-TimesTen database. TimesTen's event processing functionality can be achieved in a conventional RDBMS through triggers and stored procedures. However, XLA is designed for lower overhead and higher performance, consistent with real-time expectations. Also, when combined with materialized view functionality, XLA events can be targeted at very granular subsets of the database. Traditional database triggers execute their logic every time a change is made to any record in the table.

Materialized views combined with XLA functionality provide an efficient way to implement fine-grained event notification. That is, a materialized view pulls together data that is related to specific events that might have actions taken on it. An XLA application needs only to monitor update records that are of interest from a single materialized view table. Without a materialized view, the XLA application would have to monitor all the update records from all the base tables, including records reflecting updates to rows and columns of no interest to the application.

Conclusion

With the growing speed of messages moving through business networks, the use of real-time processing to capture, analyze, and respond intelligently to key events is becoming the benchmark for corporate excellence. This isn't important simply for the execution and management of critical business processes. Customers expect highly tailored interactions and the utmost responsiveness from any company with which they do significant business.

What's needed is a generation of lightweight infrastructure software that presents familiar, powerful interfaces and query languages that are widely in use—that can easily interface with existing back-office databases, messaging systems, and application servers—and exploits the full performance potential of today's networked, memory-rich computing platforms. This is the generation of infrastructure software for real-time data management provided by the Oracle TimesTen product.

TimesTen provides application-tier data management for performance-critical systems, optimized for blazing-fast response and real-time caching of Oracle data. Thousands of companies worldwide, including Alcatel-Lucent, Aspect, Avaya, Bank of America Merrill Lynch, Bridgewater Systems, BroadSoft, Cisco, Deutsche Börse, Ericsson, JPMorgan, KDDI, NEC, NYFIX, Smart Communications, United States Postal Office and Verizon Wireless use Oracle TimesTen in production applications.