

Using Oracle In-Memory Database Cache to Accelerate the Oracle Database

An Oracle Technical White Paper
March 2008

Using Oracle In-Memory Database Cache to Accelerate the Oracle Database

Oracle TimesTen is targeted at real-time applications where microseconds matter. It accelerates business processes, enables real-time business intelligence, and facilitates the personalization of customer-facing applications.

1. INTRODUCTION

Oracle In-Memory Database Cache is an Oracle Database product option ideal for caching performance-critical subsets of an Oracle database for improved response time in the application tier. In-Memory Database Cache consists of three key technology components – an in-memory database for real-time data management via Oracle TimesTen In-Memory Database, a transactional data replication component to ensure data high availability in the application tier, and an in-memory caching component to cache frequently access tables from Oracle database.

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. A TimesTen in-memory database may be used as the database of record, and/or as a cache to the Oracle Database.

TimesTen has a proven track record with production deployments since 1998 in real-time enterprises and time-critical industries that include network telecommunication services, operational support systems, contact centers, airline and reservation systems, command and control systems and securities trading. Hundreds of companies worldwide use Oracle TimesTen in production applications, including Amdocs, Aspect, Avaya, Bombay Stock Exchange, Bridgewater Systems, Cisco, Deutsche Borse Systems AG, Ericsson, JP Morgan, Lucent, NEC, Nokia, Salesforce.com, Smart Communications and Sprint.

Applications may create and manage database tables in the TimesTen in-memory database or cache frequently used subsets of an Oracle Database into in-memory cache tables. Cached tables from the Oracle database and regular TimesTen tables may coexist in the same in-memory database, and are all persistent and recoverable. Queries and updates to the cache are performed by the application through standard SQL. Applications running on different mid-tier servers may cache different or overlapping subsets of the same back-end Oracle database. In-Memory Database Cache automatically keeps the caches synchronized with the back-end database while delivering stellar performance to the applications.

Application-tier caching is typically used to improve data access latency and to reduce workload on the back-end database.

2. APPLICATION-TIER CACHING

Various caching techniques have been developed to improve database access performance or to reduce contention on the back-end database server. Fast response time is particularly important for real-time applications or applications that interact with an end user. In addition, reducing the workload on the back-end database is relevant to applications with an ever-growing community of users such as hosted software services, eCommerce sites or telecommunication services.

There are many choices as to what information to cache and where to cache it, with each option offering advantages and tradeoffs. Some of the caching techniques that have been developed include:

- *Query results caches.* This is typically done in the application tier and is managed by special software that hides the presence of the cache from the application. Under this scenario, the caching software automatically saves the results of queries that are submitted to the database system. A cache hit is recognized and serviced from the cache if a query is an identical match to a previously-submitted query, including identical values of parameters. The advantage of such caching is that it is simple and it caters to access scenarios where the same query is likely to be submitted over and over. However, it is limited in scope, as it cannot handle query processing on the content of the cache.
- *Object-Relational mapping tool caches.* Object-Relational mapping tools (O/R mapping tools) such as Oracle TopLink or JBoss Hibernate hide relational databases from object-oriented programmers by providing transparent mapping between objects and relational data. Once relational data is mapped to an object representation, it may be cached by the O/R mapping tool until it is no longer needed or until it becomes stale. Caching by O/R mapping tools is a common technique to avoid the expensive mapping between the programming language's object model and the database's relational model.
- *Object caches.* The word caching is somewhat of a misnomer here because objects that end up in these caches are not necessarily subsets of objects that are stored elsewhere. These "caches" are repositories of objects that are independent of the objects' origins. They are not typically transparent to applications. Applications "put", "get", "insert" and "delete" objects from the caches. There are a few products on the market that offer such caches and they vary in the level of functionality they support. The caches may be strictly memory resident or they may be backed to disk, or to another data management system. Some products provide concurrency control, some provide transparent distribution over multiple nodes in a network, and some provide high availability.

TimesTen caches have full relational and SQL functionality, automatic maintenance of data consistency with the Oracle Database and real-time performance.

Oracle In-Memory Database Cache takes a unique approach to caching that is not used by any other product on the market today. This product allows the caching of tables or table fragments from an Oracle Database to the application tier. The table fragments are described through an extended SQL syntax and are cached into in-memory cache tables. Applications read and update the in-memory cache using standard SQL, and In-Memory Database Cache automatically propagates updates from the back-end DBMS to the cache and vice versa. Thus, Oracle In-Memory

Database Cache offers applications the full generality and functionality of a relational database, the transparent maintenance of cache consistency with the Oracle Database, and the real-time performance of an in-memory database.

The In-Memory Database Cache approach has two major benefits that contribute to improving overall performance. First, applications that use the in-memory cache experience significantly reduced response time and increased throughput due to the in-memory architecture of the TimesTen database and the elimination of communication between the application tier and the database server because the TimesTen database can run in process with the application or via shared memory IPC in the same machine. Second, this approach reduces the workload on the back-end database thus improving overall throughput for all applications.

The ability to provide all the advantages of relational databases, coupled with real-time performance, small-footprint and embeddability are unique to Oracle TimesTen In-Memory Database. Consequently, In-Memory Database Cache is ideal for caching performance-critical subsets of an Oracle database, enabling both reads and updates, and automatically managing data consistency between the cache and the Oracle database.

The next few sections will give a brief introduction to the Oracle TimesTen In-Memory Database (more details may be found in [1]), a description of how to cache data using Oracle In-Memory Database Cache,; a few important caching scenarios will also be provided.

3. THE ORACLE TIMESTEN IN-MEMORY DATABASE

The Oracle TimesTen In-Memory Database is a memory-optimized relational database that supports standard SQL92 through the ODBC and JDBC APIs. Although TimesTen operates on data that is in main memory, TimesTen databases are persistent and recoverable in case of software, hardware or power failures. Durability is ensured through checkpointing and logging to disk. Applications may choose ACID properties for their transactions, but more relaxed options are also available for higher performance. TimesTen provides a cost-based query optimizer and applications may view and influence query plans. The TimesTen database is available as a library that may be linked by applications as well as through a client/server option. When TimesTen is accessed through the client/server option, each request to TimesTen incurs the overhead of inter-process communication even if the application and the TimesTen server are running on the same machine. By contrast, when TimesTen is linked with the application, the requests to TimesTen are nothing but subroutine calls that incur a negligible overhead and any data transfers between the application and TimesTen are nothing but inexpensive memory copying operations. High availability is provided through replication. A number of utilities are also available, including an interactive SQL utility, on-line backup and restore, and bulk loading. Database maintenance operations are also available through programmatic APIs.

A copy of the database resides in main memory at run time. It is managed in a shared memory segment that is accessed by all processes connected to that

The TimesTen In-Memory Database provides transactional access to data and relational functionality through standard APIs.

database. Figure 1 shows the architecture of a TimesTen In-Memory Database system.

TimesTen data structures and algorithms are optimized around the memory residence of data.

TimesTen data structures and access algorithms exploit the in-memory residence of the database for breakthrough performance. Compared to a fully cached disk-based database, TimesTen's memory-optimized architecture uses far fewer CPU cycles, because the overhead to manage memory buffers and account for multiple data locations (disk and memory) is eliminated.

Oracle TimesTen's memory-optimized performance is complemented by functionality that supplies transactional properties, persistence mechanisms and recovery from system failures. A variety of choices are available for locking, multi-user isolation and logging, accommodating a range of application scenarios from transient look-up caches to core transactional financial trading and telecommunications billing systems.

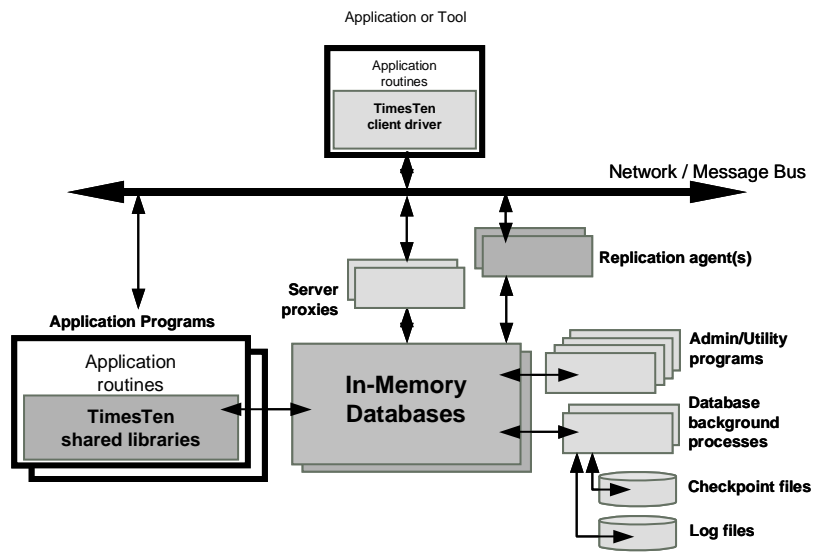


Figure 1. TimesTen Architecture

TimesTen databases are persistent and recoverable.

Durability is achieved in TimesTen by logging the changes from committed transactions to disk and periodically updating a disk image of the database through checkpoints. The timing of the disk write for the log is configurable by the application, either synchronous with the end of the transaction, or deferred until afterward, resulting in higher performance. 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 in a network which communicate their cell location every few seconds.

TimesTen allows applications to track changes to specific tables. This is useful in environments where applications are sensitive to the occurrence of certain events. For example, an application may want to know when the price of a certain stock has risen above a given threshold. This change notification feature is particularly

useful as it allows the tracking of changes not only to base tables, but to materialized views as well.

3.1 Oracle TimesTen Performance

TimesTen can achieve response times in the microseconds due to its in-memory architecture. With Oracle TimesTen In-Memory Database, a transaction that reads a database record can take less than 5 microseconds (a microsecond is one millionth of a second), and transactions that update or insert a record can take less than 15 microseconds.

Very low response times cannot be achieved through hardware additions. TimesTen can deliver very low latency due to its unique architecture.

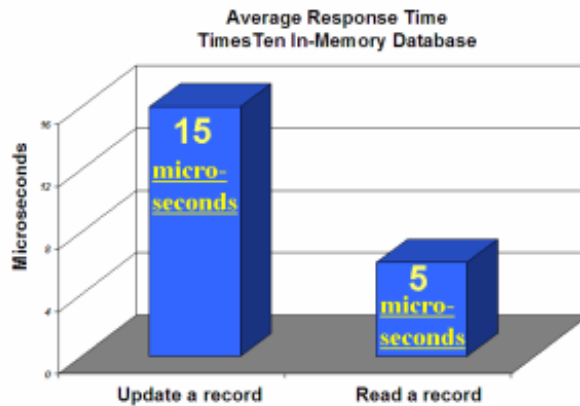


Figure 2. TimesTen Response Time

Figure 2 shows the response times for an application doing read and update transactions measured with Red Hat Linux on an AMD Opteron 1.8GHz processor.

4. DATA CACHING USING ORACLE IN-MEMORY DATABASE CACHE

Using Oracle In-Memory Database Cache, the in-memory database running in the application tier may contain tables that are cached from an Oracle Database as well as regular tables that belong strictly to the TimesTen database. The cached tables may be subsets of Oracle tables described through an extended SQL syntax that allows projections and selections on the cached tables.

An in-memory database cache contains subsets of Oracle Database tables.

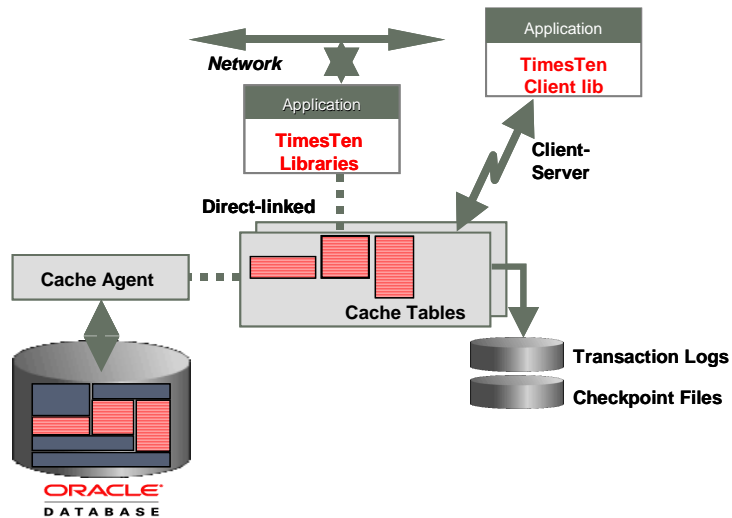


Figure 3. Caching with TimesTen

In-memory cache tables may themselves be replicated to other TimesTen databases. Alternatively, overlapping caches may be set up as separate caches that are independently synchronized with the Oracle database. The choice of configuration depends on the application. Examples of different configurations are given in the last section of this paper.

4.1 Defining the Content of a Cache

To define what is to be cached, Oracle In-Memory Database Cache provides Cache Administrator; a browser-based easy-to-use tool that allows the application designer to view the schema of a chosen Oracle back-end database. From that schema, the user chooses the sub-schema that should be cached using the concept of Cache Groups. A *Cache Group* is a set of TimesTen in-memory tables that corresponds to a set of related (through foreign key constraints) and frequently used tables in the Oracle Database. SQL syntax is used to define Cache Groups and to choose the columns and rows that should be cached from a set of related tables. The Cache Administrator assists the user in defining Cache Groups and automatically generates the appropriate SQL syntax. Users may also define Cache Groups programmatically or via the interactive ttlsq utility, using SQL syntax.

Example:

Assume that the following tables exist in the Oracle Database:

- Customer (CustId, Name, Age, Gender, StreetAddress, State, ZipCode, PhoneNo)
- Order (CustId, OrderId, PurchaseDate, Amount)
- CustInterest (CustId, Interest)

An application may want to cache the profiles of customers who have placed orders since January 1, 2007. To that end, it may define the following two cache groups:

- The first cache group contains subsets of the three tables above for customers who have placed orders since January 1, 2007, and who

The content of a TimesTen cache is defined through an extended SQL syntax.

also live in the Pacific region of the United States. Furthermore, the application may choose to cache only a subset of the tables' columns. For example, it may cache the following columns:

- Customer (CustId, Name, Age, Gender, State)
 - Order (CustId, PurchaseDate, Amount)
 - CustInterest (CustId, Interest)
- The second cache group contains the same information as the first cache group, but for customers in the Mountain region of the United States.

The two cache groups may be cached on different nodes running Oracle In-Memory Database Cache.

An additional concept used by In-Memory Database Cache is that of a Cache Instance. A *Cache Instance* is a collection of related records that are uniquely identifiable, and is used to model a complex object. Cache Instances form the unit of cache loading and cache aging as will be described below. In the example above, all records in the Customer, Order, and CustInterest tables that belong to a given customer ID (CustId) belong to the same Cache Instance, and are related to each other through foreign key constraints. CustId uniquely identifies the Cache Instance and is referred to as the *Cache Instance Key*.

TimesTen supports the same data types as the Oracle Database.

In addition to supporting its own data types, Oracle TimesTen supports the same basic data types as the Oracle Database so there is no need to map Oracle data types to TimesTen data types. But it is possible to map Oracle data types to more efficient TimesTen implementations. For example, an application may map an Oracle NUMBER data type to a TimesTen INTEGER data type.

Note that application developers can create indexes on the in-memory cache tables. The in-memory cache indexes may match the indexes in the Oracle Database or may be different. The application designer can use the flexibility of TimesTen in-memory database to create multiple indexes on the same table and may define indexes over multiple columns.

4.2 Loading Data and Managing the Cache

Once a Cache Group has been defined, the application must decide how to load the data described by the Cache Group from the Oracle database into the in-memory database cache for processing. The following techniques are available for loading data:

- Load the entire Cache Group at once. This is a suitable technique to use if the content of the entire Cache Group can fit in the cache. The ability to unload an entire Cache Group is also available.
- Load Cache Instances transparently. Transparent Loading is useful when cache content is dynamic. In this case, the records that make up a Cache Instance are loaded into the cache automatically when a SELECT query does not find the requested data in the cache. If the Cache Instance is already in the cache, the SELECT query is serviced directly from the cache. This technique

Dynamic caches use Transparent Loading and Cache Aging.

is useful if the capacity of the cache is not large enough to hold all the data described by the Cache Group.

Transparent Loading is typically coupled with the automatic *Cache Aging* facility. Cache Instances can be automatically aged out of the cache when the cache capacity is exceeded. In-Memory Database Cache supports *Usage-Based Aging* and *Time-Based Aging*. Usage-Based Aging uses an LRU (least recently used) scheme to age out Cache Instances when the cache capacity is exceeded. Time-Based Aging grants Cache Instances a *Lifetime* of certain duration in the cache, and requires the presence of a timestamp column in one of the tables of the Cache Group. The value of the timestamp column is managed by the application. Cache Instances can remain in the cache as long as their timestamp value plus Lifetime does not exceed the current time. Note that Cache Aging can be used independently of transparent loading, and in fact may be used with regular TimesTen tables that are not cached from an Oracle database.

An application may choose to have some Cache Groups subject to aging and others not. For example, the application may want to keep catalog information in the cache all the time, but may want to load users' profiles on demand when users connect to the application, and age out the profiles automatically when users disconnect. Cache Instances can also be explicitly unloaded by the application.

- Load Cache Instances “by WHERE clause”. In this case, a WHERE clause is used to describe the subset of the Cache Group that should be brought into the cache. Automatic Cache Aging may be used with loading by WHERE clause. Applications can also unload Cache Instances “by WHERE clause”.

Data that has been loaded into in-memory cache tables is available for SQL processing through JDBC or ODBC.

4.3 Maintaining Data Consistency

During normal processing, applications read and update data cached in the in-memory database cache. Applications residing on the same node or on different nodes can share caches. Furthermore, different caches of the same back-end Oracle database may reside on the same node or on different nodes. These caches may be identical, may contain different subsets of the Oracle database, or may partially overlap. For example, an application tier may consist of several servers each dedicated to serve a subset of subscribers. The subscribers may be partitioned according to postal code, country code, user identifier, etc. With such a scheme, the data cached on each server will contain a different subset of the Oracle database.

Cached data may be updated in the in-memory database cache or in the Oracle Database. Oracle In-Memory Database Cache provides the ability to automatically propagate updates from the cache to the Oracle Database, and vice versa. However, an underlying assumption is that a Cache Group is either mostly or exclusively updated in the cache, or in the Oracle Database. It is a major design flaw to cache a set of tables that are expected to be heavily updated in both the

TimesTen caches are updatable and TimesTen automatically maintains consistency between cached data and the Oracle Database.

cache and the back-end database. There are, however, scenarios where it is appropriate to allow updates in both the cache and the back-end Oracle database. The updates in the Oracle database may, for example, occur only at night for maintenance reasons while updates take place in the cache(s) during the day; or updates to central data may occur in the Oracle database, while updates to regional data occur in the cache(s).

Cache Groups may be *System-Managed* or *User-Managed*. There are three types of System-Managed Cache Groups:

- *Read-Only*. These Cache Groups may not be updated in the cache. They may be updated in the Oracle database, and In-Memory Database Cache manages the propagation of updates from the Oracle database to the cache.
- *Asynchronous Writethrough*. These Cache Groups may be updated in the cache but not in the Oracle database. In-Memory Database Cache propagates updates from the cache to the Oracle database asynchronously after the commit of a transaction.
- *Synchronous Writethrough*. These Cache Groups may be updated in the cache but not in the Oracle database. Updates in the in-memory cache tables are propagated to the Oracle database synchronously with the commit of a transaction.

System-Managed Cache Groups have well-defined semantics and restrictions to enforce these semantics. By contrast, the semantics of User-Managed Cache Groups are left to the application. For example, a User-Managed Cache Group may be updatable in both the cache and the Oracle database.

In-Memory Database Cache applications can send SQL statements to either a Cache Group or to the Oracle Database through a single connection to a TimesTen database. This single-connection capability is enabled by a *PassThrough* feature that checks if the SQL statement can be handled locally by the in-memory cache tables or if it must be redirected to the Oracle Database. The *PassThrough* feature provides settings that specify what types of statements are to be passed through and under what circumstances. One particularly useful setting is the one that specifies that all statements that update the database are to be passed to the Oracle Database. This setting allows an application to have updates executed in the Oracle Database and reads executed in the In-Memory Database Cache through a single connection.

The sections below describe the In-Memory Database Cache operations available to maintain the consistency of cached data. Some of these operations are initiated automatically by In-Memory Database Cache; others must be initiated explicitly by the application.

4.3.1 Update Propagation from the Back-End Database to In-Memory Database Cache

For a Cache Group that is updated in the Oracle database, the following mechanisms are available to keep the content of the cache in synch with the Oracle database:

- *Refresh*. This is an explicit request from the application to refresh either an entire Cache Group or specific Cache Instances. It is equivalent to an unload operation followed by a load operation.
- *Full Autorefresh*. With Full Autorefresh, the application indicates how frequently refreshes ought to take place, and In-Memory Database Cache automatically refreshes the Cache Group at the time intervals indicated by the application.
- *Incremental Autorefresh*. Unlike Full Autorefresh, an Incremental Autorefresh updates only the records that have been modified in the Oracle database since the last refresh. As with Full Autorefresh, the application must indicate the frequency of the refreshes, and In-Memory Database Cache automatically performs the incremental refresh at that frequency.

Incremental Autorefresh may be used with Time-Based Aging to keep a sliding window in the cache. For example, a customer support application may want to keep in the cache all incidents reported in the last 5 days. In this case, it can specify that the Cache Group should use Incremental Autorefresh, and Time-Based Aging with a Lifetime of 5 days. As new incidents are inserted into the Oracle database, Incremental Autorefresh will propagate them automatically to the in-memory cache tables. If these incidents are updated in the Oracle Database, the updates will be propagated automatically to the in-memory cache tables. The incidents must have a timestamp that is maintained by the application. Once the value of a timestamp is more than 5 days older than the current date, the associated incident will be aged out of the cache automatically.

The three techniques described above are useful under different circumstances. Assume a Cache Group that needs to be refreshed only once a day at 2 a.m. when activities at a content provider site are minimal. In this case, a Full Autorefresh may be the best choice. On the other hand, a Cache Group that needs to be refreshed once every five minutes should use an Incremental Autorefresh. Finally, a Cache Group that needs only infrequent refreshes, but at unpredictable times that are known only to the application, should use the Refresh option.

An application or multiple applications may configure several caches against the same back-end database. The caches may be completely disjoint, partially overlapping, or identical. The propagation of updates from the Oracle Database to all the caches will be managed by Oracle In-Memory Database Cache.

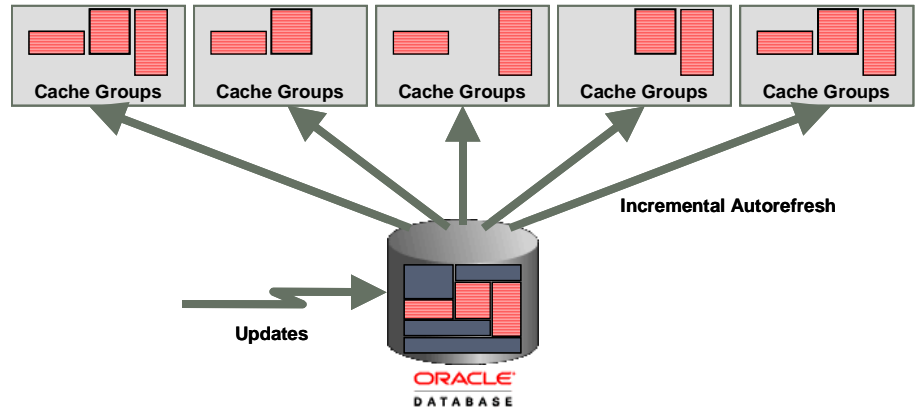


Figure 4. Incremental Autorefresh of Cache Groups

4.3.2 Update Propagation from the In-Memory Cache to the Oracle Database

For a Cache Group that is updated in the cache, the following mechanisms are available to keep the Oracle database in sync with the cache:

- Propagate.* With the propagate option turned on, all modifications to a Cache Group, i.e., all insert, update and delete operations are automatically propagated to the back-end Oracle Database. The time at which the propagation takes place differs for Synchronous and Asynchronous Writethrough Cache Groups. With Synchronous Writethrough Cache Groups, when the application completes a transaction that has modified one or more Cache Groups, the transaction is first committed in the Oracle database, and then in the in-memory cache database. This technique allows the Oracle Database to apply any required logic related to the data before it is committed in the in-memory cache database. With Asynchronous Writethrough Cache Groups, when the application completes a transaction, the transaction is committed in the cache database and control returns to the application. The changes made in the transaction are then asynchronously propagated to the Oracle Database.
- Flush.* This operation is driven by an explicit request from the application and may be applied to Cache Groups or Cache Instances. It is only allowed on Cache Groups or Cache Instances that have the Propagate option turned off. The operation updates the records in the Oracle Database with the values of the records in the cache. This operation is useful when frequent updates take place for some period of time over the same set of records. Rather than propagate a play-by-play of each update, the final image of each record is sent and applied to the Oracle Database.

An application may configure several Writethrough caches against the same back-end Oracle database. The propagation of updates from the caches to the back-end database will be managed by Oracle In-Memory Database Cache, but it is recommended that different Writethrough caches against the same back-end be non-overlapping lest different updates against the same data take place at the same time in different nodes, resulting in unpredictable data values on the back-end.

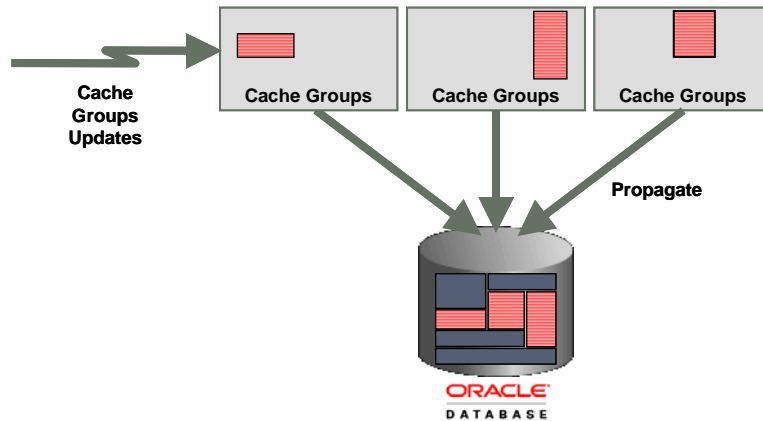


Figure 5. Update Propagation

4.3.3 Update Propagation between TimesTen Nodes

TimesTen replication may be configured between different nodes to maintain the consistency of cached data across the nodes for load balancing or high availability. Replication may be configured between Read-Only Cache Groups as well as between Writethrough Cache Groups.

Consider for example an application that wants to deploy 100 cache nodes with identical Read-Only Cache Groups. Rather than set up 100 different caches all getting their Incremental Autorefreshes directly from the Oracle Database, the application might instead configure 10 caches only, but then configure each of these caches to replicate its data to 9 other TimesTen databases.

Similarly, for high availability purpose, an application may want to maintain a replica of each of its Writethrough Cache Groups in case the node holding the master Cache Group goes down.

TimesTen supports high availability across the middle tier and the back-end database.

4.4 High Availability

When Oracle TimesTen is used exclusively as the database of record as opposed to an in-memory database cache to the Oracle Database, it ensures the high availability of its data through replication, various on-line operations and a number of utilities that support failover, recovery, and on-line upgrades. Similarly, the Oracle Database supports high availability of its data through a set of features that includes Oracle Real Application Clusters (RAC), Oracle Automatic Storage Management (ASM), and Oracle Data Guard. In addition, the Replication component of In-Memory Database Cache provides a number of features that ensure the high availability of cached data, and automatic recovery from failures that span the application tier and the Oracle database in the database tier.

4.4.1 Handling of Failure in the Oracle Database

If the Oracle Database becomes inaccessible to In-Memory Database Cache for any reason such as network failure, hardware failure, or Oracle Database failure, In-Memory Database Cache is designed to be resilient to such failures. The in-memory cache will continue to be accessible to applications. Furthermore, in case of an Asynchronous Writethrough Cache Group, updates to the cache will continue to be logged in Oracle TimesTen so that once the Oracle Database becomes accessible again, the updates are propagated to it. Similarly changes to Read-Only Cache Groups that were made on the Oracle Database but not yet propagated to the in-memory cache will remain recorded on the Oracle database and will be propagated to the cache(s) once the Oracle database is accessible again.

In addition, In-Memory Database Cache takes full advantage of RAC's high availability features. A RAC configuration consists of a single physical database that is accessible by several nodes. The runtime configuration on a single node is called an *instance*¹. RAC provides load balancing, high availability, and data consistency across all instances.

In-Memory Database Cache recovers quickly from a RAC node failure without requiring user intervention. To do this, In-Memory Database Cache uses Oracle's Transparent Application Failover (TAF) and Fast Application Notification (FAN) features whenever they are available, i.e., depending on the version of the Oracle client, server and the TAF configuration. If an Oracle instance to which In-Memory Database Cache is connected fails, the connection is automatically switched to another instance. If a Refresh, Full Autorefresh or Incremental Autorefresh operation was in progress when the failure occurs, the operation will automatically rollback the changes that took place in the in-memory database, and will restart the operation. If a Propagate operation for an Asynchronous Writethrough Cache Group was in progress when the failure occurs, the transaction will automatically rollback the changes that took place in the Oracle Database if they need to be rolled back, and will restart the Propagate operation.

4.4.2 Handling of Failure of an In-Memory Cache Node

To protect against failure of cache nodes and ensure continuous availability of cached data, TimesTen replication provides failure and recovery handling of cache nodes. The Active Standby Pair replication configuration with multiple Read-Only Subscribers is designed to include an Oracle Database as part of the configuration and to provide failover and recovery of cache nodes.

With Active Standby Pair, all updates are always applied first on the Active node. Updates are then replicated to the Standby node, and are afterwards replicated from the Standby node to all Read-Only Subscriber nodes. Thus, the Standby node is always ahead of all Read-Only Subscriber nodes, and therefore if the Active node goes down, there is no doubt as to which of the subscriber nodes should become the new Active node.

¹ There can actually be multiple instances on one node.

4.4.2.1 Read-Only Cache Groups

The Active Standby Pair replication configuration is designed to work with both Read-Only and Writethrough Cache Groups. With Read-Only Cache Groups, updates that are applied in the Oracle database are propagated to the Active node only. TimesTen replication then propagates the updates to all the other nodes by going through the Standby node first

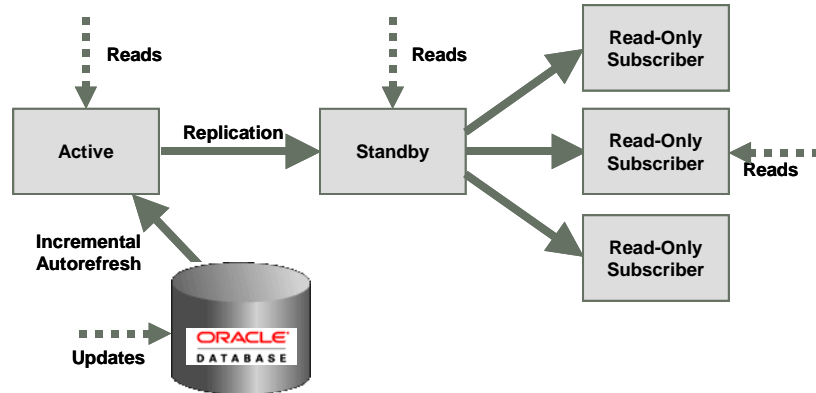


Figure 6. Read-Only Cache Groups Using Active-Standby Pair Configuration

If the Active node fails, the Standby node becomes the new Active node. From that point on, updates from the Oracle Database are propagated to the new Active node, i.e., the former Standby, and are then replicated to the Read-Only Subscriber nodes. Once the old Active node is brought back online, it becomes the new Standby node. TimesTen Replication handles automatically the switch of update propagation from the Active to the Standby, and the recovery of the failed node.

Similarly, if the Standby node fails, replication from the Active node will automatically be redirected to the Read-Only Subscriber nodes. Once the Standby node is brought back on line, Replication will ensure that it catches up with all the updates that it missed before resuming its role as a Standby node.

4.4.2.2 Writethrough Cache Groups

With Writethrough Cache Groups, updates are applied to the Active node. They are then replicated to the Standby node. Once there, they are propagated to the Oracle Database, and replicated to all the Read-Only Subscriber nodes.

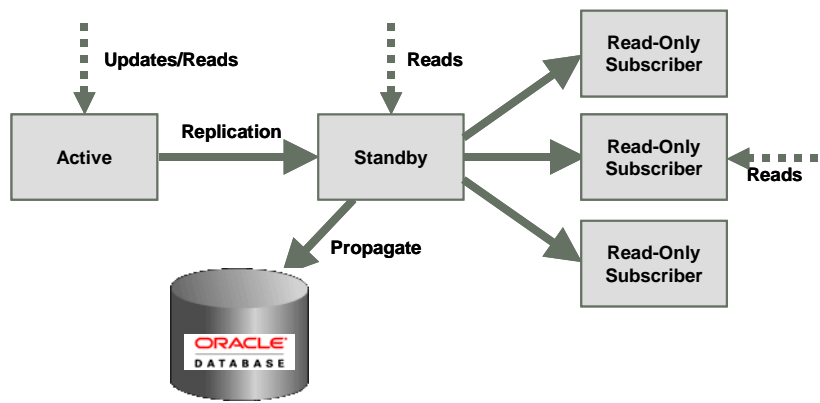


Figure 7. Writethrough Cache Groups Using Active-Standby Configuration

If the Active node fails, the Standby node becomes the new Active node. From that point on, all updates must be sent to the new Active node, i.e., the former Standby node. Updates to the new Active node will be propagated to the Oracle Database and replicated to the Read-Only Subscriber nodes. Once the old Active node is brought back online, it becomes the new Standby node. TimesTen Replication handles automatically the recovery of the new Standby node and the transfer of responsibility of propagating updates to the backend to the new Standby node.

Similarly, if the Standby node fails, replication from the Active node will automatically be redirected to the Read-Only Subscriber nodes, and the Active node will start propagating updates directly to the Oracle Database. Once the Standby node is brought back on line, Replication will ensure that it catches up with all the updates that it missed before resuming its role as a Standby node, and will have the Standby node resume propagating updates to the Oracle Database and the Read-Only Subscribers.

5. PERFORMANCE

To measure the performance of In-Memory Database Cache, we developed a benchmark that simulates a Home Location Register (HLR) application that is used in cellular networks. The benchmark consists of a set of 7 transactions, each of which models a typical operation executed by an HLR such as setting up or deleting call forward or updating information about a mobile phone subscriber.

We ran the benchmark in two different configurations. In the first configuration, the HLR application ran against the Oracle Database with the benchmark application running on one server, and Oracle Database 10g on a second server. In the second configuration, we added In-Memory Database Cache in front of the Oracle Database; the HLR application program was linked directly with the TimesTen cache database running on one server, and Oracle Database 10g running on the other server. The cached data was stored in Asynchronous Writethrough Cache Groups, enabling all updates against cached data to be propagated automatically to the Oracle Database.

The application was implemented in Java using JDBC for data access. All four servers had identical configuration with 6GB of physical RAM, two Intel Xeon 2.4GHz processors with hyper-threading running on SUSE Linux Enterprise Server 9.

We measured the average response time for each type of transaction when run against the Oracle Database and when run against In-Memory Database Cache. The graph below shows a significant reduction in application response time when using In-Memory Database Cache.

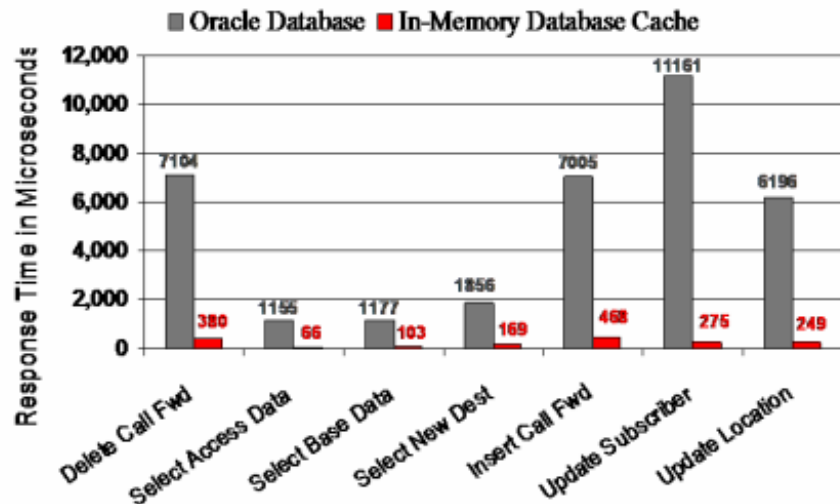


Figure 8. Response Time Comparison for HLR Benchmark Application

We also measured the combined throughput of all transactions in the two configurations. The chart below shows a significant increase in application throughput when using Oracle In-Memory Database Cache.

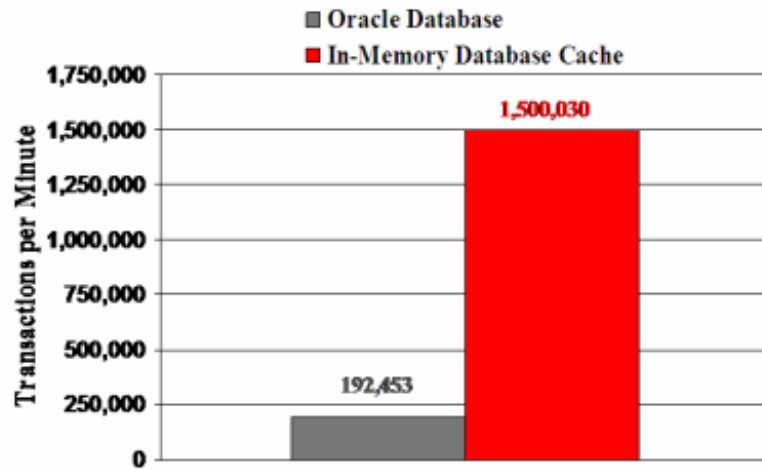


Figure 9. Throughput Comparison for Benchmark HLR Application

This benchmark shows the benefits of using Oracle In-Memory Database Cache. As shown in the above graphs, the application response time achieved 10 to 40 times improvements and overall throughput improved more than 7 times. In general, the In-Memory Database Cache improvement ratio varies depend upon the hardware and platforms.

6. EXAMPLES

In this section, we examine a few caching scenarios and the recommended Oracle In-Memory Database Cache configuration for these scenarios.

6.1 Read-Only Cache

There are many applications that can benefit from read-only caches. The main characteristic of these applications is that some set of records is queried over and over. These records may be frequently or infrequently updated, but the ratio of reads to writes is high. Examples of such records include price lists for online shopping applications, airline schedules for airline reservation applications, and room availability for hotel booking applications.

The best cache configuration for such data is a (System-Managed) Read-Only Cache Group with Incremental Autorefresh. The data may be updated on the back-end database. The updates will be propagated automatically to the cache. The frequency of propagation is determined by the application and should depend on the frequency of updates on the back-end and the currency of the data needed by the application.

If the cache is to be deployed on more than one node, then independent caches may be set up on each node where each cache is updated directly from the back-

Read-Only Cache Groups with Incremental Autorefresh are ideal for caching frequently-referenced data.

end database or replication may be set up between some nodes where updates first go to an intermediary node, and are then replicated to other nodes.

Note that an online shopping application will typically have no need to update the price list. Changes to the price list are likely to be made by some central administrative process. On the other hand, a hotel booking application, while it is read intensive, also needs to update the database with new room reservations. The updates are best applied to the Oracle Database with Incremental Autorefresh taking care of the update propagation to all the caches that include the updated data. The application can set up a single connection to the TimesTen cache database and use the PassThrough option to route all updates to the Oracle Database while executing all reads in the cache.

6.2 Read-Only Sliding Window Cache

Read-Only Cache Groups with Incremental Autorefresh and Time-Based Aging are ideal for caching frequently-referenced data that falls in a sliding window.

In many cases, the read-only data that is needed by an application is data with a time component where newer data is more frequently accessed than older data. New data is constantly generated, and older data gets less valuable for this specific class of applications. So, one can think of a fixed-length time interval that is constantly moving forward with data coming into the interval on one end, and falling off the interval on the other end. The application is only interested in the data that is within the interval, typically referred to as a sliding window.

Examples of applications that may have a need for data that falls in a sliding window are a stock trading application that may want the last 3 days of trading, or a news delivery application that may want the last 24 hours of news clips.

To cache data that falls in a sliding window, we want new data to be brought into the cache automatically and old data to age out of the cache automatically. We also want data that is in the cache to be updated automatically should changes be made to the back-end. The best cache configuration for such data is a (System-Managed) Read-Only Cache Group with Incremental Autorefresh and Time-Based Aging.

As in the previous example, if the cache is to be deployed on more than one node, then independent caches may be set up on each node where each cache is updated directly from the Oracle Database or replication may be set up between some nodes where updates first go to an intermediary node, and are then replicated to other nodes.

6.3 Updatable Cache

Asynchronous Writethrough Cache Groups are ideal for updatable caches.

Some applications need immediate, real-time update to cached data with eventual propagation of updates an Oracle Database. For example, an application that manages and provisions phone subscription services and authenticates access to such services will typically cache subscriber information in the in-memory cache database. Changes to a user's service must be reflected immediately in the cache, and should be propagated to the back-end database.

The best configuration for such data is an Asynchronous Writethrough Cache Group.

Asynchronous Writethrough Cache Groups with Usage-Based Aging are ideal for capturing data with uneven arrival rates.

6.4 Data Capture Cache with Uneven Arrival Rate

There is a class of applications where new data is generated at a very high rate for some periods of time and moderate rate at other periods. During the periods of high activity, the back-end database is often unable to keep up with the high throughput demanded by the application. Such applications can benefit from a cache that, in effect, ends up “smoothing” the arrival rate of the newly-generated data.

For example, a stock ticker application will have the rate of arrival of new values vary greatly over time. It will be particularly high when the market opens and closes, and will be lower at other times. The peak arrival rate cannot be typically handled by a disk-based database, but can be sustained by In-Memory Database Cache.

The best cache configuration for such data is a (System-Managed) Asynchronous Writethrough Cache Group with Usage-Base Aging. Inserts to Writethrough Cache Groups are automatically propagated to the back-end Oracle database. And, Usage-Based Aging will automatically remove data from the in-memory cache to free up space.

6.5 Data Capture Cache with Constant High Arrival Rate

There is another class of applications where new data is also generated at a high rate, but where the high arrival rate does not necessarily subside. Caching temporarily data whose arrival rate is too high to be absorbed by a back-end database does not solve the problem if the arrival rate does not subside as there is no interlude for the back-end database to catch up. But, it is often the case for such applications that the newly-generated data can be aggregated into a more condensed form prior to being permanently stored in the back-end database. It is also typical for these applications to analyze the data they collect in real time to detect interesting or anomalous patterns.

An example of such an application is one that collects data from sensors or RFID readers. The data is often repetitious and can be easily aggregated and it often needs real-time analysis.

The best configuration for such applications is to insert the data as it arrives in one or more tables that are managed by Oracle TimesTen only, i.e., there is no image of this data in the back-end database. The non-cached TimesTen-only tables can be configured with Usage-Based Aging. Once the data is aggregated by the application, it can be inserted in the cache in a (System-Managed) Asynchronous Writethrough Cache Group with Usage-Based Aging. In-Memory Database Cache will automatically propagate all the aggregates to the back-end database. Since both the TimesTen-only tables and the cache tables are configured with Usage-Based Aging, least-recently used records will be aged out automatically to make room in the cache for new records.

Asynchronous Writethrough Cache Groups with Usage-Based Aging coupled with regular TimesTen tables with Usage-Based Aging are ideal for capturing data with a constantly high arrival rates.

Asynchronous Writethrough Cache Groups with Transparent Loading and Usage-Based Aging are ideal for updatable dynamic caches.

6.6 Updatable Dynamic Cache

For some applications, access to active data must be very fast, but the set of active data varies over time and is a subset of a much larger amount of data that cannot be held entirely in a cache because it is too large. The active data needs to be brought in the cache on demand, and the content of the cache needs to be dynamic so that active data can replace stale data.

An example of such applications is the use of personalization to better serve online customers. When a customer reaches a web site, the customer profile is loaded into the web site. As the customer interacts with the site, his/her profile may get updated. Sometime after the customer leaves the site, the profile is no longer needed and can be deleted from the cache.

The best configuration for such data is an Asynchronous Writethrough Cache Group with Transparent Loading and Usage-Based Aging. As soon as the application issues a SELECT for a user's profile, the profile will be loaded automatically in the cache if the profile is not already there. If the application updates the profile, the updates will be propagated automatically to the Oracle Database. Finally, as more space is needed in the cache, profiles that have not been accessed for a while will be deleted automatically from the cache.

6.7 Updatable User-Managed Cache

Some applications need to execute multiple updates in the cache for best performance, but need to permanently record the final transaction in the Oracle Database. An example of such an application is an eCommerce application where the application might maintain a number of shopping carts for active users. The shopping carts will be updated repeatedly in the cache. These updates need not be propagated to the Oracle Database as they are of little value. However, once a user executes a purchase, the transaction needs to be permanently recorded in the Oracle Database.

The best configuration for such data is a User-Managed updatable Cache Group where the application issues explicit Flush requests whenever it needs to record a transaction in the Oracle Database. This configuration may be coupled with Usage-Based Aging so that abandoned shopping carts get deleted from the cache automatically.

6.8 Read-Only Dynamic Distributed Cache

In some cases, an application may be distributed over many nodes to handle a throughput rate that cannot be handled by a single node, and the set of active data needed by the application is dynamic, and is at any given time, a much smaller subset of the full data set. An example of such an application may be a trading application where the active data is the profile of active traders.

The best configuration for such data is to configure the in-memory cache on each node with a Read-Only cache group over the same set of tables in the Oracle Database, and to use Transparent Load and Usage-Based Aging with these cache tables. What will then happen is that each node will have the traders' profiles that it

User-Managed Cache Groups with explicit Flush are best suited for applications with frequent updates, but infrequent business transactions.

Read-Only tables with Transparent Load and Usage-Based Aging are best suited for a read-only dynamic distributed cache.

needs loaded automatically as it needs them, and the profiles will be aged out of the caches when they are no longer needed to make room for needed profiles.

7. CONCLUSION

Oracle In-Memory Database Cache enables you to improve application transaction response time by caching a performance-critical subset of tables and table fragments from an Oracle database to the application tier. In contrast to simple result cache mechanisms, applications can execute new SQL queries over the cached data since the cache tables are managed as regular relational database tables in the TimesTen In-Memory Database. The caches can be shared among different applications. Updates can be applied to the caches, and the caches are kept consistent with the Oracle Database. Caching data using In-Memory Database Cache is superior to other caching techniques as it brings full relational functionality, stellar performance, and automatic maintenance of data consistency with the Oracle Database to applications running in the application tier

By bringing data closer to the application, and by processing queries in an in-memory database, Oracle In-Memory Database Cache reduces response time significantly. By offloading some of the data processing work from the Oracle database server, overall application throughput is significantly improved without interfering with the centralized management and administration of the back-end database.

8. REFERENCES

1. Oracle TimesTen Products and Technologies. *An Oracle White Paper, December 2005.*



Using Oracle In-Memory Database Cache to Accelerate the Oracle Database
March 2008

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com

Copyright © 2008, Oracle. All rights reserved.

This document is provided for information purposes only and the contents hereof are subject to change without notice.

This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission. Oracle, JD Edwards, PeopleSoft, and Siebel are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.