

An Oracle Technical White Paper
July 2009

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

| | |
|--|----|
| 1. Introduction..... | 2 |
| 2. Application-Tier Caching..... | 4 |
| 3. The Oracle TimesTen In-Memory Database | 5 |
| 3.1 Oracle TimesTen Performance | 8 |
| 4. Data Caching Using Oracle In-Memory Database Cache | 8 |
| 4.1 Defining the Content of a Cache | 10 |
| 4.2 Loading Data and Managing the Cache..... | 11 |
| 4.3 Sharing Data Across a Cache Grid | 12 |
| 4.4 Maintaining Data Consistency | 13 |
| 4.5 High Availability | 18 |
| 5. Performance..... | 21 |
| 6. Examples..... | 23 |
| 6.1 Read-Only Cache | 23 |
| 6.2 Read-Only Sliding Window Cache | 24 |
| 6.3 Updatable Cache | 24 |
| 6.4 Updatable Dynamic Cache | 25 |
| 6.5 Data Capture Cache with Uneven Arrival Rate | 25 |
| 6.6 Data Capture Cache with Constant High Arrival Rate | 26 |
| 6.7 Updatable User-Managed Cache..... | 27 |
| 6.8 Read-Only Dynamic Distributed Cache..... | 27 |
| 7. Conclusion..... | 27 |
| 8. References | 28 |

1. Introduction

The Oracle In-Memory Database Cache accelerates business processes, enables real-time business intelligence, and facilitates the personalization of customer-facing applications.

The *Oracle In-Memory Database Cache (IMDB Cache)* is an Oracle Database product option ideal for caching performance-critical subsets of an Oracle database in the application tier. Using the IMDB Cache improves an application's response time and throughput. The IMDB Cache consists of three key technology components – the Oracle TimesTen In-Memory Database (TimesTen) for application-tier real-time data management; caching technology to cache frequently-accessed tables from an Oracle Database server to the application tier and maintain consistency of cached data; and a transactional data replication component to ensure cross-tier high availability.

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

Applications may create and manage database tables in TimesTen or cache frequently-accessed subsets of an Oracle Database in the IMDB Cache. Cached tables and non-cached tables may coexist in the same in-memory database, and are all persistent and recoverable. Queries and updates to cached and non-cached data are performed by applications through SQL92 or PL/SQL using ODBC, JDBC, the Oracle Call Interface (OCI), or TTCclasses, as well as Pro*C.

Cache Grids are available for horizontal scalability in performance and capacity where a Cache Grid consists of a collection of IMDB 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, effectively making the aggregate of data cached in all grid members available to the application. 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.

The IMDB Cache manages the availability of data across the application tier and the database server tier. It ensures high availability and no transaction loss no matter where a failure may occur. Whether a failure occurs in one of the cache nodes, one of the Oracle RAC nodes, at the network level or even for an entire RAC cluster, high availability and no transaction loss are guaranteed.

TimesTen and the IMDB Cache have a proven track record with production deployments 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 TimesTen and the IMDB Cache in production applications, including Alcatel-Lucent, Amdocs, Aspect, Avaya, Bombay Stock Exchange, Bridgewater Systems, BroadSoft, Cisco, Deutsche Börse, Ericsson, JP Morgan, NEC, NYFIX, Smart Communications, and Sprint.

2. Application-Tier Caching

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

Various caching techniques have been developed to improve database access performance or to reduce contention on back-end database servers. Fast response time is particularly important for real-time applications and customer-facing applications. In addition, reducing the workload on the back-end database is important 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) 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.

The Oracle In-Memory Database Cache has full relational and SQL functionality, automatic maintenance of data consistency with the Oracle Database and real-time performance .

Oracle In-Memory Database Cache (IMDB Cache) takes a unique approach by enabling 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 the Oracle TimesTen In-Memory Database (TimesTen). Applications read and update cached data using SQL, PL/SQL, or Pro*C and the IMDB Cache automatically propagates updates from the Oracle Database to the cache and vice versa.

A collection of IMDB Caches may be configured as a Cache Grid. Cached data is distributed between grid members, and the Cache Grid provides applications with location transparency and concurrency control, effectively making the aggregate of all data cached in grid members available to applications. As an application's performance or capacity needs increase, additional nodes may be added to the Cache Grid with no interruption to service. Thus, the IMDB Cache offers applications the full generality and functionality of a relational database, incremental scalability coupled with location transparency, automatic maintenance of cache consistency with the Oracle Database, and the real-time performance of an in-memory database.

The IMDB Cache approach has two major benefits that contribute to improving overall performance. First, applications that use the IMDB Cache experience significantly reduced response time and increased throughput due to the in-memory architecture of TimesTen and the elimination of communication between the application tier and the database server. 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, incremental scalability, and automatic cache management are unique to the IMDB Cache. It is ideal for caching performance-critical subsets of an Oracle database, enabling both reads and updates of cached data, and automatically managing data consistency.

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 data is cached and managed by the Oracle In-Memory Database Cache, and a few illustrative caching scenarios.

3. The Oracle TimesTen In-Memory Database

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

The Oracle TimesTen In-Memory Database is a memory-optimized relational database that supports SQL92 and PL/SQL through the ODBC, JDBC, Oracle Call Interface (OCI), and

TTClasses¹ APIs, as well as through Pro*C/C++. By supporting standard interfaces and popular Oracle interfaces, TimesTen ensures ease of adoption by already-existing applications.

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, requests to TimesTen are nothing but local 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, a graphical tool for database development and cache configuration, 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 database. Figure 1 shows the architecture of a TimesTen In-Memory Database system.

The Oracle TimesTen In-Memory Database data structures and algorithms are optimized around the memory residence of data.

TimesTen data structures and access algorithms exploit the 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 supports transactional properties, persistence mechanisms and recovery from system failures. A variety of choices is available for locking, multi-user isolation and logging, accommodating a

¹ 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.

range of application scenarios from transient look-up caches to core transactional financial trading and telecommunications billing systems.

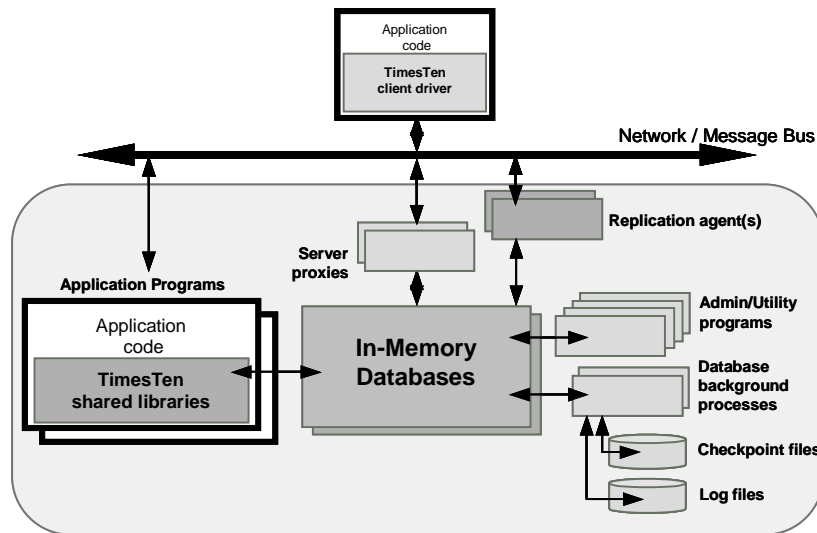


Figure 1. TimesTen Architecture

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

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

TimesTen can achieve response times in the microseconds due to its in-memory architecture. With TimesTen, a transaction that reads a database record can take fewer than 5 microseconds, and transactions that update or insert a record can take fewer than 15 microseconds.

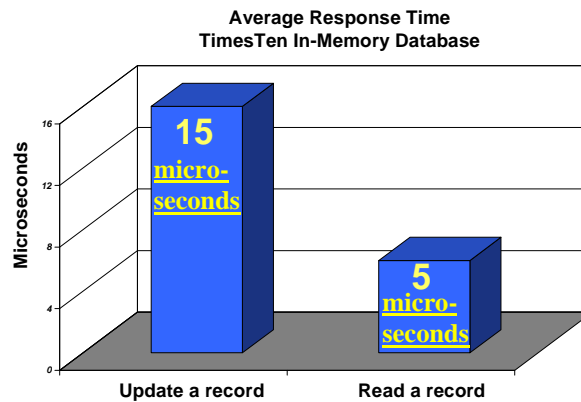


Figure 2. TimesTen Response Time

Figure 2 shows the response times for an application executing read and update transactions on a 2-CPU Intel E5450 (8-way@3GHz) system running Oracle Enterprise Linux 5.2.

4. Data Caching Using Oracle In-Memory Database Cache

An IMDB Cache contains subsets of an Oracle Database tables.

The IMDB Cache enables the caching of subsets of tables from an Oracle Database to the application tier. Cached tables are updatable and the IMDB Cache synchronizes data between the Oracle Database and the cache.

The database engine that manages the cached data is the Oracle TimesTen In-Memory Database. It is augmented by the ability to load and synchronize cached data. One of the background processes associated with the IMDB Cache is the Cache Agent, which manages some of this synchronization. Figure 3 shows the architecture of an IMDB Cache.

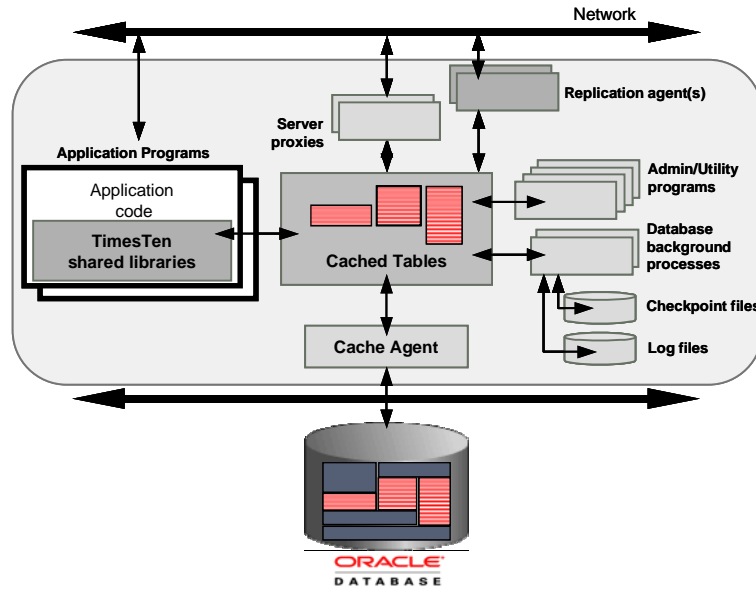


Figure 3. IMDB Cache Architecture

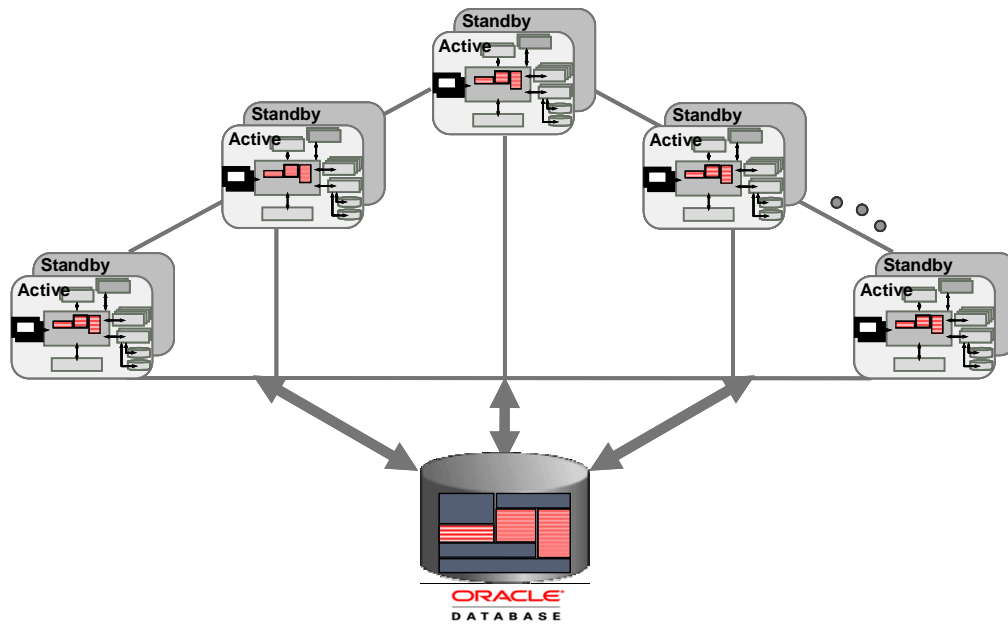


Figure 4. Cache Grid with Five Replicated Grid Members

A *Cache Grid* is a collection of IMDB Caches that collectively manage the application data. A cache grid consists of one or more grid members each backed by an IMDB Cache. Grid members cache tables from a central Oracle database or Real Application Cluster (RAC). Cached data is distributed across multiple nodes or IMDB Caches without shared storage. A Cache Grid ensures that data is consistent across nodes. Grid members may be replicated.

Figure 4 shows a Cache Grid consisting of five replicated grid members. Additional grid members may be added incrementally with no interruption to operations. The replication configuration that must be used with the IMDB Cache is the active standby pair configuration.

4.1 Defining the Content of a Cache

The content of an Oracle IMDB Cache is defined through an extended SQL syntax.

A *Cache Group* is a set of IMDB Cache tables that corresponds to a set of frequently-used Oracle database tables that are related through foreign key constraints. SQL syntax is used to define Cache Groups and to choose the columns and rows that are to be cached from the Oracle database tables. Users may define Cache Groups programmatically or via the interactive `ttIsql` utility.

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, 2009. 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, 2009, and who 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 IMDB Cache.

An additional concept used by the IMDB 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, TimesTen supports the same basic data types as the Oracle Database so there is no need to map Oracle Database data types to TimesTen data types. But it is possible to map Oracle Database data types to more efficient TimesTen implementations. For example, an application may map an Oracle Database 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 to create multiple indexes on the same table and may define indexes over multiple columns.

4.2 Loading Data and Managing the Cache

An application must decide how to load Cache Group data into the IMDB Cache for processing. The following techniques are available for loading data:

- **Explicit Loading.** This can be done in one of several ways:
 - 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 “by WHERE clause”. In this case, a WHERE clause is used to describe the subset of the Cache Instances that should be brought into the cache. Applications can also unload Cache Instances “by WHERE clause”.
 - Load Cache Instances “by ID”. In this case, a list of Cache Instance Ids is used to specify Cache Instances that should be brought into the cache. Applications can also unload Cache Instances “by ID”.
- **Dynamic Loading.** This technique is available for loading cache instances. Dynamic Loading is useful when the Cache Group is too large to fit in the cache, and hence only an application’s working set is to be kept in the cache. In this case, the records that make up a

Cache Instance are loaded into the cache automatically on a cache miss, i.e., when a SQL statement² does not find the requested data in the cache. If the Cache Instance is already in the cache, the statement is handled directly from the cache.

Dynamic Loading is typically coupled with automatic *Cache Aging*. Cache Instances can be automatically aged out of the cache when the cache capacity is exceeded. IMDB 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 a 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 dynamic 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 at all times, 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.

Data that has been loaded into in-memory cache tables is available for SQL, PL/SQL, Pro*C processing through JDBC, ODBC, TTClasses, and OCI.

4.3 Sharing Data Across a Cache Grid

Cache Groups may be local or global. With *Local Cache Groups*, cached data is not shared across members of the same Cache Grid. Grid members may have disjoint data or overlapping data, and it is up to the application to determine how data is distributed among them. For example read-only catalog data may be cached in all grid members for best performance, and updatable customer information may be partitioned by geography in different grid members. Committed updates on cached tables are propagated to Oracle tables with no coordination with other grid members. A Local Cache Group can be defined as explicitly loaded or dynamically loaded. Cache Groups are local by default, unless they are defined as global.

² Dynamic loading of a Cache Instance is available for SQL statements with an equality expression on the primary key or foreign key of any of the records in the Cache Instance.

In a *Global Cache Group*, cached data is shared across members of the same Cache Grid. Concurrency control is enforced across the grid and a transaction running anywhere in the grid always sees the most recent committed version of a Cache Instance. Committed updates to the same Cache Instance by different grid members are propagated to the Oracle database in the order in which they were committed within the grid to ensure data consistency.

4.4 Maintaining Data Consistency

The Oracle IMDB Cache supports updates to cached data and automatically maintains consistency between caches and the Oracle Database.

Cached data may be updated in the IMDB Cache or in the Oracle Database. The IMDB 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 cache and the back-end database. There are, however, scenarios where it is appropriate to allow updates in both. 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 Cache Groups*. These Cache Groups may not be updated in the cache. They may be updated in the Oracle database, and the IMDB Cache manages the propagation of updates from the Oracle database to the cache.
- *Asynchronous Writethrough (AWT) Cache Groups*. These Cache Groups may be updated in the cache but not in the Oracle database. The IMDB Cache propagates updates from the cache to the Oracle database asynchronously after the commit of a transaction.
- *Synchronous Writethrough (SWT) Cache Groups*. 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.

Read-only, AWT, SWT, and User-Managed Cache Groups may all be Local Cache Groups. However, only Dynamic AWT Cache Groups may be specified as Global Cache Groups.

The table below summarizes the various Cache Group loading, Cache Grid sharing, and consistency maintenance options that are available.

| | | Loading data into a Cache Group | | | |
|------------------------------|--------------------------|---------------------------------|--------------------|-------------------|--------------------|
| | | Explicit Loading | | Dynamic Loading | |
| Maintaining Data Consistency | Read-Only Cache Group | x | | x | |
| | AWT Cache Group | x | | x | x |
| | SWT Cache Group | x | | x | |
| | User-Managed Cache Group | x | | x | |
| | | Local Cache Group | Global Cache Group | Local Cache Group | Global Cache Group |

Sharing data across a Cache Grid

IMDB Cache applications can send SQL statements to either a Cache Group or to the Oracle Database through a single connection to an IMDB Cache 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 IMDB Cache through a single connection.

The sections below describe the IMDB Cache operations available to maintain the consistency of cached data. Some of these operations are initiated automatically by IMDB Cache; others are initiated explicitly by the application.

4.4.1 Update Propagation from IMDB Cache to Oracle Database and Among Cache Grid Members for Global Cache Groups

As we have already seen, Global Cache Groups are also Dynamic AWT Cache Groups. An application with Global Cache Groups will be connected to one of the grid members. Most of the time, it will access Cache Instances that are already cached in the grid member. However, in case it tries to access a Cache Instance that is not in the grid member, the IMDB Cache will load that Cache Instance dynamically either from another grid member or from the Oracle Database, depending on where the most recently updated version of the Cache Instance resides. This is done automatically with no intervention from the application. The IMDB Cache determines

where the most recent copy resides and uses peer-to-peer communication to exchange information with other IMDB Cache databases in its grid.

If a transaction updates a Cache Instance in any of the grid members, the following mechanism is available to keep the Oracle database in sync with the cache:

- *Propagate.* The IMDB Cache propagates the updates to the Oracle Database after the transaction commits. If another transaction updates the same Cache Instance in another grid member shortly thereafter and commits, the IMDB Cache guarantees that the commits are propagated to the Oracle Database in the correct order.

Figure 5 shows a Cache Grid consisting of three Grid Members. All grid members cache the same Global Cache Group, and each grid member has in its cache only a few Cache Instances from the Global Cache Group. These instances are cached because they have been recently accessed in their respective grid members. As time elapses, each instance either continues to be accessed in its grid member and thus remain there, or gets accessed in another grid member and is moved to that member, or is not accessed at all, in which case it ages out of the Cache Grid entirely.

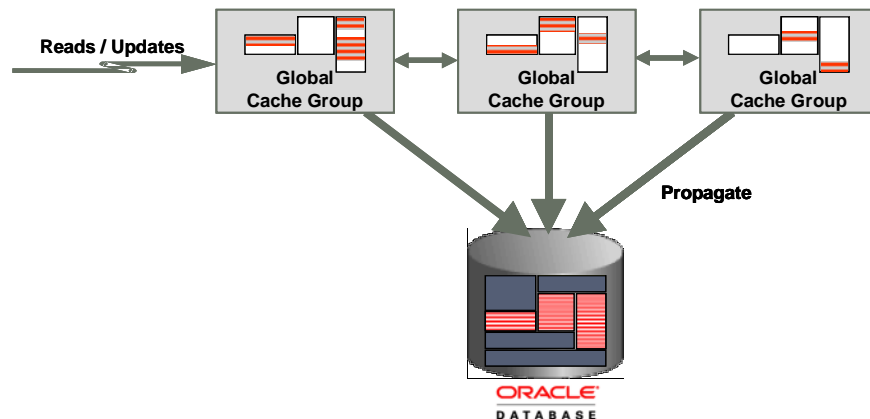


Figure 5. Update propagation and Cache Consistency for Global Cache Groups

4.4.2 Update Propagation from IMDB Cache to Oracle Database for Local Cache Groups

For Local Cache Groups that may be 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 Oracle Database. The time at which the propagation takes place differs for SWT and AWT Cache Groups. With SWT 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 IMDB Cache. This technique allows the Oracle Database to apply any required logic related to the data before it is committed in the IMDB Cache. With AWT Cache Groups, when the application completes a transaction, the transaction is committed in the IMDB Cache and control returns to the application. The changes made by 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 a Cache Grid with several updatable Local Cache Groups in different grid members. The propagation of updates from the grid members to the Oracle Database will be managed by the IMDB Cache, but it is recommended that Local Cache Groups in different grid members 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.

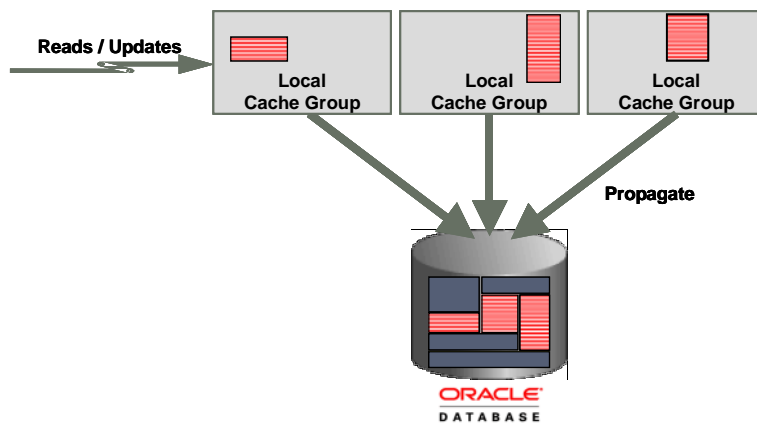


Figure 6. Update Propagation for Updatable Local Cache Groups

4.4.3 Update Propagation from Oracle Database to IMDB Cache for Local Cache Groups

For a Local Cache Group³ that is updated in the Oracle database, the following mechanisms are available to keep the content of the cache in sync 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 IMDB 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 IMDB 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, 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 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.

³ Note that Global Cache Groups may not be updated on the Oracle Database.

An application may configure a Cache Grid with several read-only Local Cache Groups in different grid members. The Cache Groups in the different grid members may be completely disjoint, partially overlapping, or identical. The propagation of updates from the Oracle Database to all the grid members will be managed by the IMDB Cache.

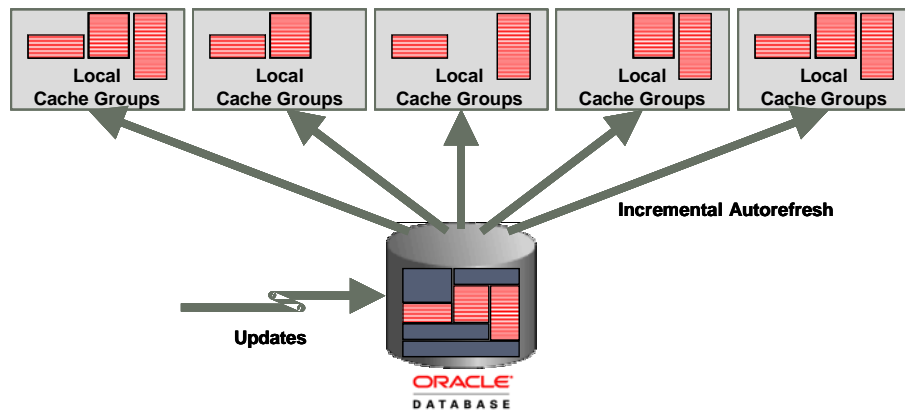


Figure 7. Incremental Autorefresh of Read-Only Local Cache Groups

4.5 High Availability

The IMDB Cache supports high availability across the application tier and the database server tier.

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. Automatic failure detection and failover of database and applications is available through integration with Oracle Clusterware. 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 IMDB 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. They are described below.

4.5.1 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.

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.

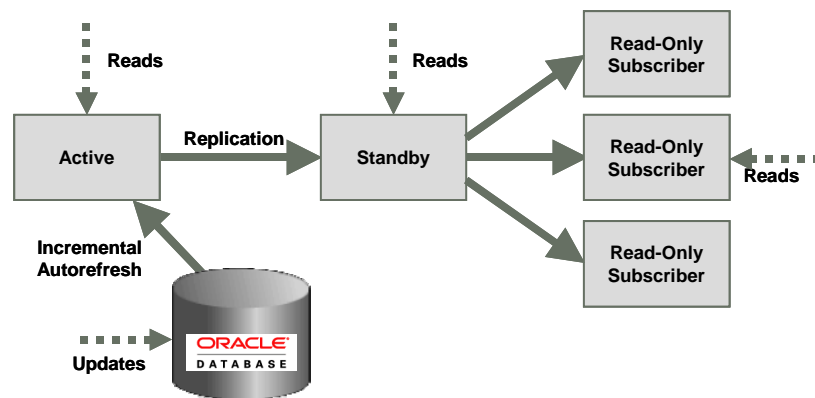


Figure 8. Read-Only Cache Groups Using Active-Standby Pair Configuration

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 (the former Standby), and are then replicated to the Read-Only Subscribers. Once the old Active node is brought back online, it becomes the new Standby. 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.

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.

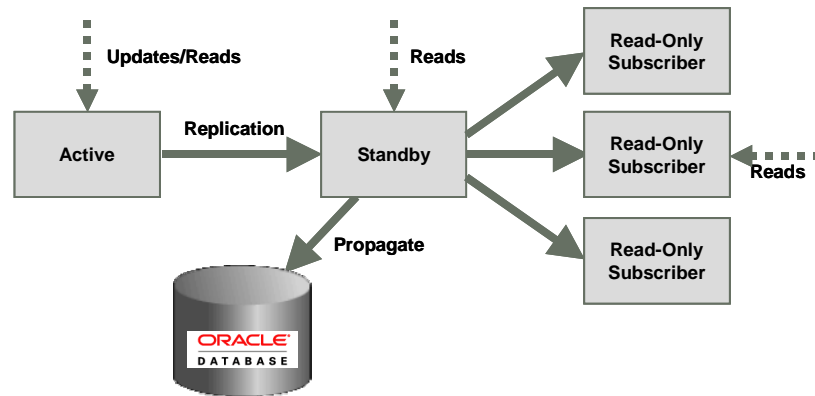


Figure 9. Writethrough Cache Groups Using Active-Standby Configuration

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.

4.5.2 Handling of Failure in the Oracle Database

If the Oracle Database becomes inaccessible to IMDB Cache for any reason such as network failure, hardware failure, or Oracle Database failure, IMDB 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 AWT 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, IMDB 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*. RAC provides load balancing, high availability, and data consistency across all instances.

IMDB Cache recovers quickly from a RAC node failure without requiring user intervention. To do this, IMDB 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 IMDB 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 AWT 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.

If the Oracle database is replicated to a standby database using Synchronous Data Guard, then in the event of a failure of the active Oracle database, the IMDB Cache will failover automatically to the standby Oracle database with no data loss.

5. Performance

To measure the performance of IMDB Cache, we developed a benchmark that simulates a Home Location Register (HLR) application as 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 IMDB 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 AWT 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 configurations with 6GB of physical RAM, two Intel Xeon 2.4GHz processors with hyper-threading running on Oracle Enterprise Linux 5.2.

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

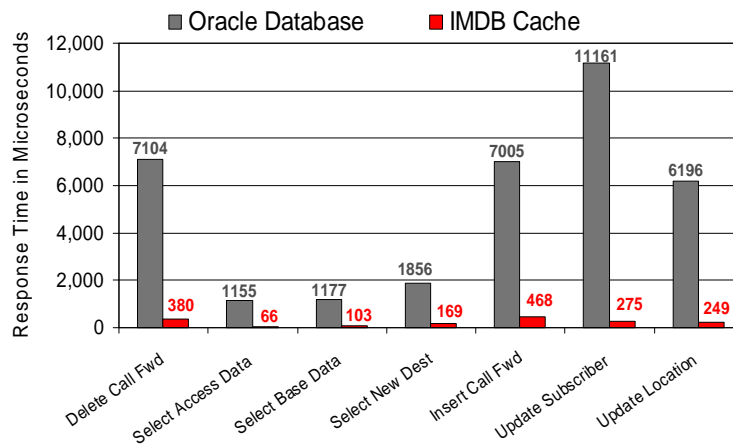


Figure 10. Response Time Comparison for HLR Benchmark Application

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 IMDB Cache.

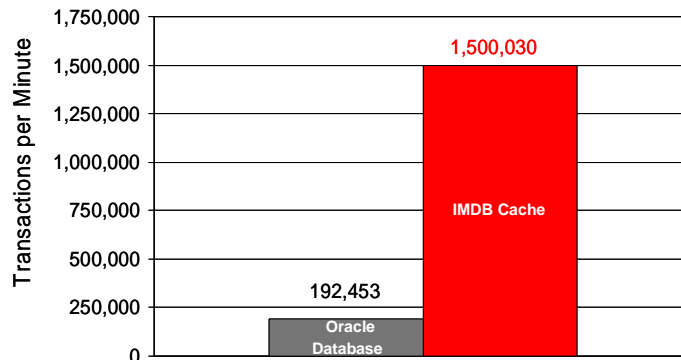


Figure 11. Throughput Comparison for HLR Benchmark Application

This benchmark shows the benefits of using IMDB Cache. As shown in the above graphs, the application response time improved by a factor of 10 to 40 and overall throughput improved more than 7 times. In general, the IMDB Cache improvement ratio varies depending upon the hardware and platforms.

6. Examples

In this section, we examine a few caching scenarios and the recommended IMDB Cache configurations and Cache Group types for these scenarios. Note that while each example focuses on a specific type of Cache Group, different types of Cache Groups may co-exist in the same IMDB Cache to best suit an application's needs.

6.1 Read-Only Cache

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

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 Local 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 multiple grid members of a Cache Grid, then Local Cache Groups should be defined on the grid members and each member will be updated directly from the back-end database.

Note that an online shopping application will typically have no need to update cached price lists frequently since changes to price lists are infrequent. On the other hand, a flight tracking application, while it is read intensive, also needs to keep the cached status of flights fairly current. The updates are best applied to the Oracle Database with a reasonably small Incremental Autorefresh interval, e.g., 5 minutes, defined for the Read-Only Cache Groups. The IMDB Cache will automatically propagate all the updates to the grid members that include the updated data. An application connected to a grid member can set up a single connection to the IMDB Cache 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 multiple grid members of a Cache Grid, then Local Cache Groups should be defined on the grid members and each member will be updated directly from the Oracle Database.

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 to 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 IMDB Cache. 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.

If the subscriber population is large enough to require that the application be deployed over multiple grid members of a Cache Grid, then Local Cache Groups should be defined on the grid members with the Cache Group in each member owning a subset of the subscriber population and with no overlap among the subsets owned by different grid members. For example, the subscribers may be partitioned by area code.

6.4 Updatable Dynamic Cache

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

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 a call center that manages a large volume of concurrent customer sessions. The application will typically be deployed over several application server nodes. Customers who contact the call center, are automatically routed to an available application server node, and the appropriate customer profile should ideally be available on that server node.

The best configuration for this scenario is an AWT Global Cache Group with dynamic loading and usage-based aging and with a grid member configured on each application server node.

With this configuration, when a customer is routed to an available server node, the appropriate customer profile is dynamically loaded from the Oracle database to the grid member. When the customer completes a call, changes to his/her profile are propagated from the IMDB Cache to the Oracle database. LRU aging will automatically remove the profiles of inactive customers from the IMDB Cache. If the same customer contacts the call center shortly after the first call and is routed to a different node, the customer profile will be dynamically loaded in the new node from either the Oracle database or from the grid member where it was previously loaded depending on where the most recent copy resides. The IMDB Cache determines where the most recent copy resides. It also manages concurrent updates to data within the grid.

All of the customer data is stored in the Oracle database. The Oracle database is much larger than the combined IMDB Cache databases and is best accessed by applications that do not require the real-time performance of IMDB Cache but do require access to large amounts of data. Such applications may include a billing application and a data mining application.

As the customer base increases and demands to serve more customers concurrently increases, the call center may decide to deploy additional application server nodes. New IMDB Cache members can join the IMDB Cache grid with no disruption to ongoing requests in the grid. Similarly, failures or removal of individual nodes do not disrupt operations in the rest of the grid.

6.5 Data Capture Cache with Uneven Arrival Rate

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

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 IMDB Cache.

The best cache configuration for such data is a (System-Managed) Asynchronous Writethrough Cache Group with Usage-Based 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.

6.6 Data Capture Cache with Constant High Arrival Rate

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 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. IMDB 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.

6.7 Updatable User-Managed Cache

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

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

Read-Only tables with Dynamic Load and Usage-Based Aging are best suited for a 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 Dynamic Load and Usage-Based Aging with these cache tables. What will then happen is that each node will have the traders' profiles that it 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 the 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 SQL and PL/SQL commands 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 IMDB Cache is superior to other caching techniques as it brings full relational functionality, incremental scalability coupled with location transparency, stellar performance, automatic maintenance of data consistency with the Oracle Database, and cross-tier high availability 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. Extreme Performance Using Oracle TimesTen In-Memory Database. *An Oracle White Paper, July 2009.*



Using Oracle In-Memory Database Cache to
Accelerate the Oracle Database
July 2009

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



| Oracle is committed to developing practices and products that help protect the environment

Copyright © 2008, 2009, Oracle and/or its affiliates. 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 is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.