Torry Harris Business Solutions

How real is Real Time Database

Author - Nagendra Narayanamurthy

www.thbs.com

Imagine an application environment in which sub second response to thousands of events with transactional guarantees, gives an enterprise a distinct competitive advantage in the market. A Real Time database system (RTDBMS) is one in which transactions must not only maintain the consistency constraints of the traditional DBMS but also satisfy their timing constraints. It's a common norm in any type of industry to store and process data in traditional RDBMS. Here, Oracle, Sybase, mySQL, DB2 and Informix are some of the front-runners in play. When it comes to processing 1000's of transactions per second, which involve 1000's of queries to RDBMS, the traditional players fail to provide the required throughput. In such cases comes the need for Real Time Database. Real Time Databases as the term suggests are mainly built to provide extremely fast turnaround query time.

Introduction to Real Time DB (In-Memory Database)

One of the main architecture models to achieve real time responses from a datastore is to store required contents into a shared memory area and to provide PL/SQL interface to access these data. Like any RDBMS, in-memory databases are also organized into following DB components:

- Tables
- Indexes

IMDB requires an initial memory to be assigned to it, known as Permanent Size. The IMDB APIs will enable an application to connect to shared data store. The two types of connections possible are:

- Server: This type of connection uses IPC mechanisms such as shared memory and semaphores to bring the shared data store in to application address space by attaching to data store.
- Client Sever: Uses sockets to connect to data store maintainer process on the server and send/receive all requests/response though IP sockets. This type of connection may not be preferable when going for real time responses as network latencies might become a bottleneck.

In-Memory databases exhibit some/all characteristics of traditional DBMS, the main one being ACID model: Automicity, Consistency, Isolation and Durability. The ACID model is implemented as:

- Atomicity (All-or-Nothing): Through Transactions
- Consistency: Index and Value integrity constraints
- Isolation: locks (Row/Table)
- Durability: Logging

IMDBs also provide SQL interface to administer its components (DDLs) and data (DMLs). Most of IMDBs conform to SQL-92 standards. To enable consistent programming interfaces IMDB uses widely accepted JDBC/ODBC interfaces and provides API for all popular languages like C, C++, JAVA etc. Also, any programming language which can interface with ODBC should be able to interface with IMDB.

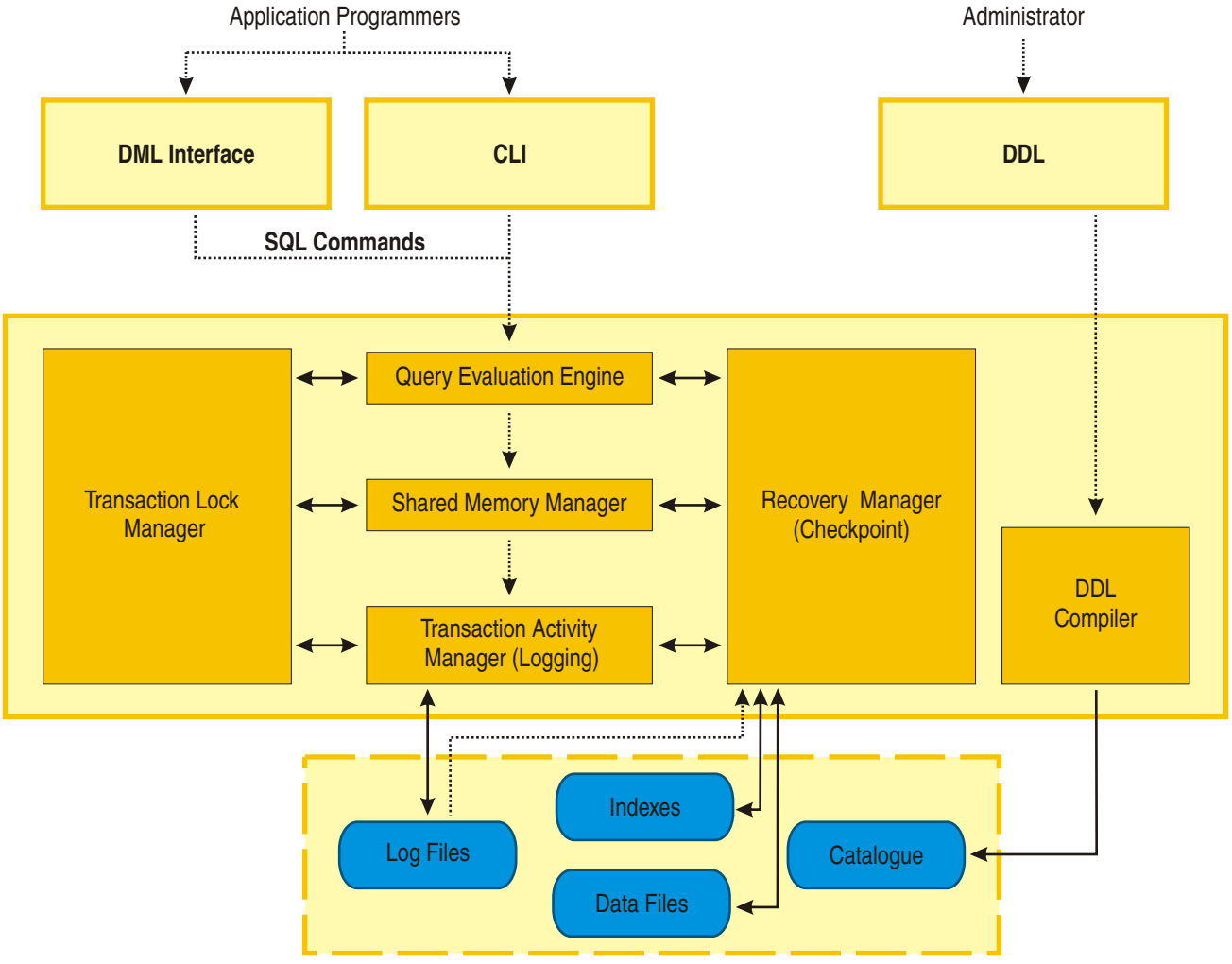
The additional features supported by IMDBs are:

- Checkpoint: Checkpointing is an online asynchronous process used to maintain on disks an up-to-date copy of the database and thereby provides a consistent starting point for log recovery. When a system failure occurs, as checkpoints provide almost up-to-date copy of the database, most data in logs are not needed. The recovery procedure only needs to process the log records generated after the last complete checkpoint.
- Caching from Conventional RDBMS: Most of the In-memory databases provide mechanisms to automatically sync contents to/from conventional RDBMS. This allows enterprises to efficiently make use of fast access feature of IMDB.

- Replication: Replication is provided as a second point of defence to address durability property of any RDBMS. In the event of failure of the system hosting IMDB, a copy maintained on replication server can be used to quickly make the system available.
- Batch Logging: Unlike conventional RDBMS systems, which tend to log individual transactions, IMDB generally tends to do a batch logging thereby decreasing disk I/O and increasing throughput. Batch logging to some extent defeats the durability property of IMDB.
- Memory de-fragmentation utilities: IMDBs maintain all their data in the memory. The continuous deletion and insertion of data into IMDB tables will leave lot of internal de-fragmented memory. To enable rearranging internal memory and fill up memory holes most IMDBs provide de-fragmentation utilities.

Architecture of In-Memory Database

Most of the In-Memory Database is based on the architecture shown below. The components shown in the yellow space are all in-memory components.



Commercial Real Databases

In-Memory Database

- TimesTen - http://www.oracle.com/timesten/index.html
- Polyhedra - http://www.polyhedra.com/
- eXtremeDB - http://www.mcobject.com/
- ERDB - http://www.entitydatabase.com/
- DataBlitz - http://www.bell-labs.com/project/dali/

Non-In-Memory Database

- ANTs Data Server 3.4 - http://www.ants.com/
- Kdb+ - http://www.kx.com/products/database.php

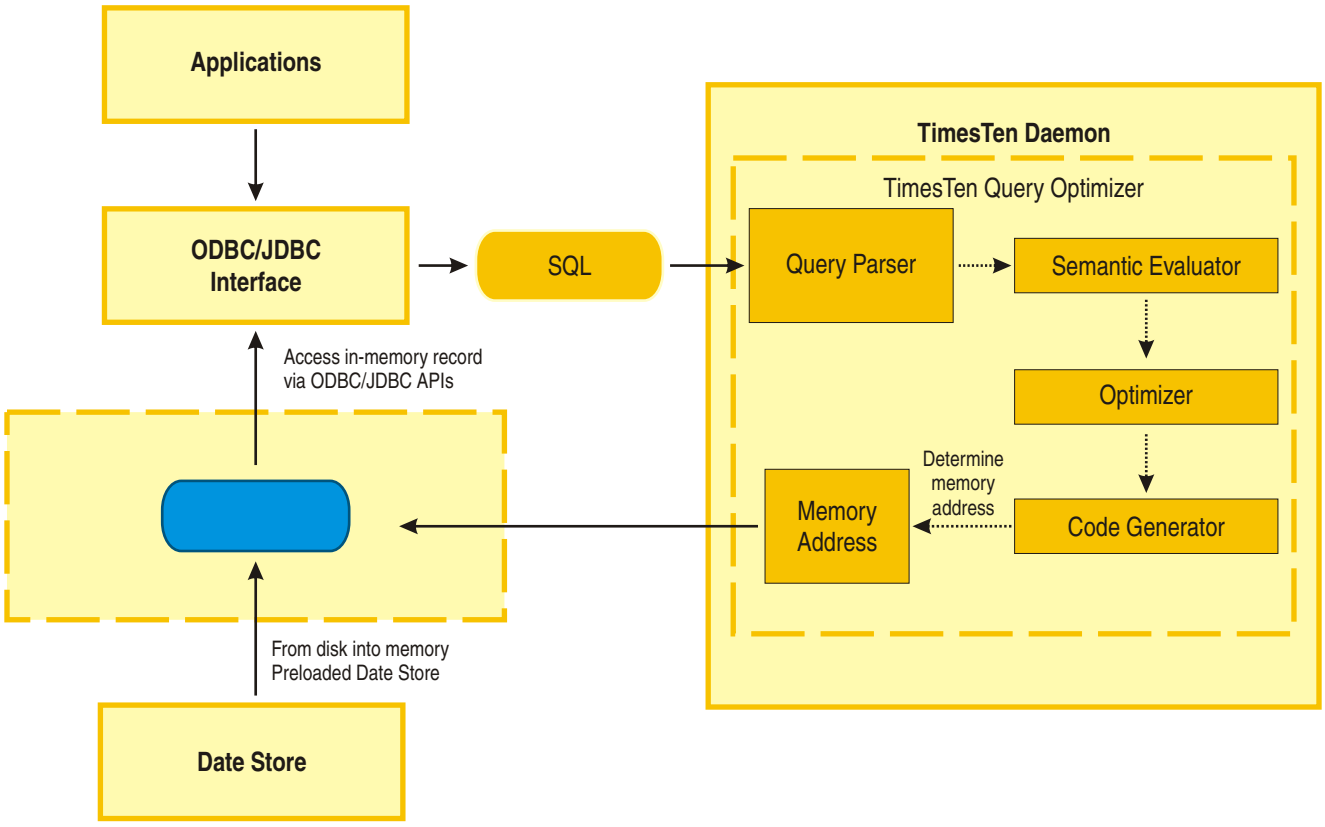
Lets get real with TimesTen

Architecture of TimesTen

A real time database product TimesTen from Oracle is one of the leading players in the real time DB market. TimesTen is an in-memory application-tier data manager that is designed to operate efficiently in an application's address space. TimesTen can be configured for:

- Diskless environment to operate entirely in memory: The data once loaded from disk (known as data store) will be stored back to disk only when the TimesTen daemon is stopped. If the data store is corrupted or destroyed for any reason, all the updates will be lost.
- Disk based environment to optionally log and checkpoint data to disk: All the transactions are logged and in case of corruption, data store can be restored by replaying transaction/updates from the log. Checkpointing is the process of keeping data store in sync with the memory resident copy.

Diskless operation doesn't satisfy Durability criteria of ACID model. TimesTen is faster than conventional RDBMS, as in RDBMS client application communicates with a database server process over IPC, which adds substantial overhead to all SQL operations. In TimesTen an application can load the database directly into its heap space or a shared memory segment to eliminate the IPC overhead and streamline Query processing. The figure given below illustrates how TimesTen operates.



Query Optimization: TimesTen includes a cost-based query optimization that speeds up data access by automatically choosing the optimal way to answer queries.

TimesTen Vs Conventional Database

TimesTen, apart from having really fast turnaround time, has most (if not all) of the conventional RDBMS features, which are:

- Standard ODBC/JDBC interfaces.
- SQL support.
- Distributed transactions: TimesTen supports distributed transactions through the XA and JTA interfaces.
- Database connectivity: Like most of other database systems, TimesTen supports client/server connections.
- Logging: TimesTen keep a log of changes. The log is used to a) redo committed transaction in case of data store crash b) undo transactions that are rolled back and c) replicate changes to other TimesTen data stores.
- Checkpoints: Checkpoint operation in TimesTen is designed to take place in the background and have very little impact on the database application. This is known as 'fuzzy' checkpoint.
- Replication: TimesTen provides asynchronous & return-receipt mechanism based replication subsystem for transmitting transactions between TimesTen subsystems.
- Query Optimization.
- Concurrency: TimesTen provides full support for shared data store.
- Administration and Utilities: TimesTen supports "typical" database utilities such as interactive SQL, back/recovery, copy and migrate.

Oracle Data Caching

Oracle Connect feature allows TimesTen to cache data stored in one or more tables in an Oracle database as a cache group. A cache group describes a group of tables in a TimesTen data store that map to all or portions of the tables in the Oracle instance. Oracle Connect supports applications to read-from or write-to cache groups. Cache groups can be refreshed (bring Oracle data into cache group) both automatically and manually. Cache groups can be flushed (propagate cache updates to Oracle) both automatically and manually. Changes to either Oracle or the cache group can be tracked automatically.

Real world scenario: Mobile Fraud Management System

A mobile service provider has a homegrown fraud management system. The fraud detection component of this system (FDS) receives real time feed of calls detail records (CDRs) of each billable call from the network systems. This system also receives the subscriber base information from the customer management systems. Subscriber base information is held in the Oracle tables. When details of a mobile call arrive at the FDS, the system has to enrich the call details with subscriber information. The volume of inflow of call details is in the order of 200 million records per day. To manage such a huge inflow the system has to operate at the speed of (200,000,000 / (24*60*60)) = 2342 call records/second. This means on an average 2342 queries to database per second. The mobile service provider has a huge customer base of 30 million subscribers. The turnaround requirement of 2342 responses per second on a table(s) holding 30 million subscribers was impossible to achieve in a traditional database. The challenge was addressed using In-Memory (Real Time) database. TimesTen is used as an intermediate caching mechanism. Oracle-Connect feature of TimesTen is used to sync subscriber data between Oracle and TimesTen. This allows the FDS system to use TimesTen APIs & ODBC connectivity to query 30 million subscribers base to associate subscriber information with call detail records. An astonishing 14,000-call details/second is achieved though the help of TimesTen, which should cater for load requirements of next few years.

Highs and Lows – IMDBs

Highs

1. Blazing fast database query.
2. SQL interface like any other traditional DBMS.
3. Easy administration. Most IMDBs sell with the punchline like "If you know RDBMS you know us".
4. Good integration with RDBMs to enable offline data sync (e.g. Oracle Connect feature of TimesTen).
5. No additional programming effort needed to perform data sync from/to traditional DB (TimesTen offers this).
6. Good for holding subset of information from master DB like Oracle, Sybase, mySQL etc.
7. Standard interfaces like JDBC/ODBC.
8. Easy integration with traditional programming languages. Stand languages: C, C++, JAVA and all ODBC complaint languages.

Lows

1. Amount of memory required is directly proportional to amount of data being stored.
2. Maximum size of DB cannot exceed available physical memory.
3. Cost constraints, as memory doesn't come cheap.
4. Needs lots of evolving by the time it provides all SQL queries available in traditional DB.
5. Lacks integration with different traditional RDBMSs.
6. Frequent de-fragmenting is required to clear memory holes.
7. When holding huge information (in order of Giga bytes), can have very high (in order of hours) start-up time, which involves loading data on to memory.