the Availability Digest

solidDB – A Five-9s Memory-Resident Database

May 2008

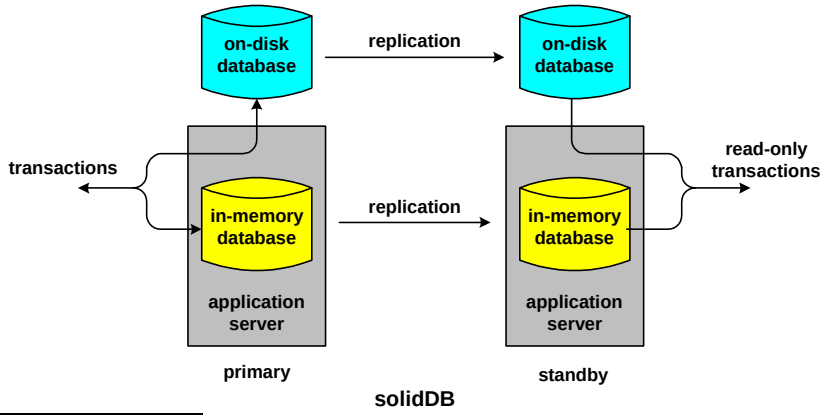
Disk-based database systems are on a collision course with Moore's Law.[1] New multicore processors can support massive amounts of memory. The databases for many real-time applications can now comfortably fit in these large, high-speed memories. solidDB, from Solid Information Technology (www.soliddb.com), is a memory-resident database that takes advantage of this new reality.[2]

So why not simply configure a system with enough cache memory to hold the database? This will certainly eliminate most disk activity and will provide very fast database access. However, the database structures that are optimum for disk-resident databases are far from optimum for memory-resident databases. Therefore, a database specifically structured for memory residence can have a significant performance improvement over a disk-resident database.

There is one problem, however. If the intent is to eliminate disk activity, what happens to the "durable" in ACID?[3] solidDB solves this dilemma through efficient disk logging or by providing a synchronized in-memory copy of the database in another server. In the latter case, should the primary database fail, the secondary database can take over with a complete, up-to-date copy of the database in tens of milliseconds.

What is solidDB?

solidDB, is a memory-resident SQL database that is extremely fast.. It is multithreaded so that it can take advantage of multicore processors. Performance tests have shown linearly-scalable



[1] http://en.wikipedia.org/wiki/Moore%27s_law
[2] In February 2008, Solid Information Technology was acquired by IBM.
[3] The ACID properties of a transactional database are atomicity (either all operations in a transaction are completed or none are), consistency (the database is always left in a legal state by a transaction), isolation (the operations in a transaction are independent of those in other transactions), and durability (the outcome of a transaction will persist through any subsequent failures of the database).

transaction processing rates of thousands of transactions per second per core.

solidDB is also highly reliable. Properly configured, it can achieve availabilities in excess of five 9s. It accomplishes this via the strategy espoused in our active/active system descriptions - “let it fail, but fix it fast.” solidDB can be run as a tightly synchronized active/standby pair. Should the active copy of the database fail, the standby copy can take over in tens of milliseconds with no loss of data. Even sessions, prepared SQL statements, and session attributes are maintained.

solidDB is highly flexible in terms of trading availability for performance according to an application’s needs. The degree of transaction protection can be adjusted over a range from total synchronization for full data protection to asynchronous synchronization for maximum performance. The less data protection that is choosen, the greater the performance.

solidDB is well-suited for applications whose databases can fit entirely in memory. However, it can also provide a disk-resident database for tables that are too large for memory or that are seldom accessed, such as archive tables. Beyond establishing the configuration, the location of tables is totally transparent to the programmer.

solidDB has a small footprint, requiring only three megabytes of memory. Used in many embedded systems, it requires no manual administration.

Applications

Hundreds of customers have deployed over 3,000,000 instances of solidDB around the world. It is especially used in telecommunications applications for embedded systems but has also found application in retail, finance, healthcare, and other enterprise applications.

In the telecommunications industry, solidDB is used to support home-location registers (HLRs) that track a cell-phone user’s location and that maintain the services to which he is entitled. It is also used extensively in VoIP (Voice over IP) and in Service Control Points (SCPs).

In some applications, it is used to synchronize the databases of mobile devices with a central master database. These applications are characterized by the fact that the mobile device is not permanently connected to the master database. Rather, when it does connect, it must exchange updates with the master database in order to synchronize the in-memory mobile database with the master database.

The solidDB Transactional Interface

SQL Standard Support

solidDB is a transaction-oriented relational database. Both its in-memory database and its on-disk database comply with the SQL-92 standard. solidDB also provides numerous features of the SQL-99 and SQL-2003 standards. Feature support includes stored procedures, triggers, events, encryption, and user and role security. solidDB incorporates a cost-based optimizer.

Data access is provided either by ODBC or JDBC statements.

Linking

solidDB can be incorporated into an application as a linked library with application programming interfaces implemented as C-language calls. This type of linkage eliminates the overhead of Interprocess messaging that would be suffered if solidDB were run as a separate process. When running at the transaction processing speeds of in-memory solidDB, processing overheads such as those imposed by interprocess messaging become significant.

Concurrency Options

Several levels of concurrency control are provided. *Read-concurrency* options include:

- Uncommitted Reads – Also known as "dirty reads," a row that is being modified by a transaction can be read by other applications.

- Repeatable Reads – If a transaction reads a row, it will always get the same data on subsequent reads of that row.

- Read Committed – Only the results of committed transactions are available to be read.

- Serializable – Once a transaction reads a row, no other transaction may modify that row until the original transaction has been committed.

Write-concurrency options include:

- Pessimistic Row-Level Locking – Before modifying a row, the row is locked so that no other transaction can modify that row (nor read it if Read Committed concurrency is configured).

- Optimistic Row-Level Locking – This is used if it is unlikely that two transactions will be attempting to simultaneously update the same row. The row is not locked by a transaction prior to updating it. Rather, it is versioned. If a concurrency violation does not occur, the overhead of locking is avoided. If a concurrency violation does occur, the transactions are aborted and must be retried.

Recovery

solidDB is a classic transaction-oriented relational database. The results of all committed transactions are carried in a transaction log.

As described later, there are various availability options that affect the currency of the transaction log. They provide compromises between availability, recovery time, and performance. However, should the system crash, either because it is a single system without a backup or because of a dual primary/backup failure, the database can be recovered by rolling forward those completed transactions that have made it to the transaction log. If strict logging is being used (i.e., each transaction is written to the transaction log before the transaction is committed), no completed transactions will be lost.

Configurations

solidDB can be configured to meet a variety of application needs.

Single System

solidDB can be run in a single server without a backup. To ensure durability, updates to the transaction log can be written to disk as each transaction completes. Furthermore, the entire in-memory database can be periodically checkpointed, with the checkpointing interval specified in time or in the number of updates.

If performance is more important than durability in an application, transaction log updates can be buffered and written to disk periodically. This improves performance because a disk write is not needed for each transaction. However, should the system crash, all transactions whose log entries have not been written to disk are lost.

Primary/Standby

solidDB can be run in a primary/standby configuration on two servers. All updates must be made to the primary database. These updates are replicated to the standby database in such a way that the standby database is always in a consistent state. The standby database can therefore be used by read-only applications. solidDB can provide automatic load balancing in this configuration by routing read requests to the least loaded system.

Should the primary database fail, the secondary database is available to immediately take over operations. Because of this, asynchronous disk logging can be used on both the primary and standby servers without the risk of losing any transactions in the event of a single failure. This further improves performance.

Master/Replica

A database can be configured to be the master database in a master/replica configuration. All updates must be made to the master database. This database can then be replicated via asynchronous replication to one or more replica databases, all of which may be used in a read-only mode. To ensure durability, the master database can also be backed up with a standby database that is kept synchronized by synchronous replication.

Replicas can be instantiated on-the-fly to accommodate load variations or changing locality requirements in the field.

Multimaster/Multitier

The target of solidDB's replication can be a full mirror of the primary system, or it can hold a subset of the primary's database. For instance, a master database might be replicated in part to several replicas to distribute data to the users. This is often the case in mobile applications.

A database can hold copies of many primary databases. In this way, a database can act as a standby for two or more primary databases.

In addition, multiple primary databases might replicate to each other. As an example, two systems running independent applications might replicate their databases to each other. In this way, each system acts as a backup for the other system while at the same time being used actively for its own applications.

Replication

In-memory and on-disk database copies are kept synchronized by data replication from the primary (or master) database to the standby database or its replicates. solidDB provides two replication mechanisms – synchronous replication and asynchronous replication.

Synchronous Replication

Synchronous replication provides carrier-grade high availability. With synchronous replication, a transaction is not completed until it is at least safe-stored on the backup system. Therefore, no data is lost following a database failure. Several options described later are provided for the degree of safe storage. These options provide compromises between performance and recovery time.

Because the primary database has to wait for the backup database to indicate that it has the transaction, communication latency between the systems must be minimal. Therefore, primary

and backup systems must be collocated. They must either be in separate servers located near each other or be in blades in a blade cabinet. solidDB communicates with the standby database at the TCP/IP level and uses whatever underlying communication channel is provided between the primary and backup systems.

Asynchronous Replication

solidDB's Smartflow asynchronous replication engine is used to synchronize replicate databases with a master database. SmartFlow uses a publish/subscribe mechanism to replicate updates to the replicas. It buffers updates and distributes them periodically to a replica upon request from that replica. The replication engine can be configured to optionally notify each replica when a change in which it is interested is available.

The replication period is configurable and typically is measured in seconds to minutes. As an option, the asynchronous replication engine can be configured to replicate every transaction as it occurs. These configuration options offer compromises between lost data and performance.

With asynchronous replication, the performance of the master database is not impacted by the replication activity. Asynchronous replication proceeds independently of the application. Therefore, replicas may be any distance from the master database.

Mobile Applications

A primary goal of solidDB is to support mobile applications. In these applications, a mobile device, typically with an in-memory database, is occasionally connected to the master system. Furthermore, the connection may be a relatively low-speed or poor quality connection.

While the mobile device is not in communication with the master, updates may be made independently both to the mobile device and to the master database. Upon connection, SmartFlow provides bidirectional asynchronous replication to exchange updates between the two databases. Data conflicts are detected and typically resolved manually.

Disaster Recovery

Recovery from a site disaster that takes down the primary database and its local backup can be achieved by providing a replica at a location separated from the primary site location by an appropriate distance. In addition to synchronously replicating transactions to its local standby, the primary system asynchronously replicates transactions via SmartFlow to its remote replica using tight asynchronous replication (such as replicating each transaction as it completes). Should the primary site be lost, the replica can be promoted to master and can continue the processing function.

Any data in the asynchronous replication pipeline at the time of the site outage will be lost, but the replicate database is otherwise consistent and can be used to continue operations.

Availability Configuration Options

The extreme availability of solidDB is achieved by synchronously backing up the primary database with a hot standby. However, synchronous replication can have a significant impact on performance.

solidDB offers several levels of replication to provide compromises between performance, durability, and recovery time.

Safe Durable

To achieve the highest level of availability, the primary's transaction commit must wait for both the primary and standby databases to be updated and for the transaction to be written to the respective logs. Using this level of replication, no transactions will be lost following the failure of the primary database or even the failure of both databases.

Safe Visible

At the next level of data protection, the primary transaction waits for the transaction to be written to the standby database; but it does not wait for the log record to be recorded by the standby. Should the primary fail, the standby database loses no transactions; and the results of all committed transactions are immediately visible to the applications running on the standby. However, failover following a primary fault is not complete until all transactions have been recorded in the log.

Safe Received

The lowest level of synchronous replication is for the transaction to complete as soon as the standby has acknowledged that it has received the transaction but before it has updated its database or has logged the transaction. In this case, following the failure of the primary system, failover is not complete until the standby system has applied all pending transactions.

SmartFlow Asynchronous Replication

Finally, if performance is truly more important than data loss, asynchronous replication can be used. In this case, any transactions in the replication pipe line are lost should the primary fail. However, transaction commits at the primary do not have to wait for the standby database to take any action.

Adaptive Logging

solidDB can be configured to change its synchronization strategy based on circumstance. For instance, in a primary/standby configuration, it is common to configure solidDB for "relaxed logging." In this mode, transaction logs are not written as part of the transaction at the primary system. However, one of the safe synchronization options is used at the standby system. solidDB depends upon a current copy of the database at the standby system to provide durability.

However, should the standby system fail (or should the primary system fail and the standby become a standalone primary), the surviving system reverts to "strict logging." To provide durability, it will log each transaction to disk before committing the transaction.

Durability Flexibility

The use of these availability configuration options is quite flexible. The level of synchronization can be specified at the system, session, or transaction level.

Availability

With solidDB running in a primary/standby configuration with synchronous replication, failover from a primary database failure can be achieved within tens of milliseconds. There is no loss of database connections, prepared SQL statements, or session attributes. solidDB can support an availability of five 9s.

How is this claim substantiated? Since most of the installations of solidDB are in embedded systems deployed by its customers, the company cannot obtain meaningful field statistics.

However, it does know failover time. If failover time is, say, fifty milliseconds, then a five-9s availability would allow some 6,000 failures per year per database (five minutes per year). It is easy to determine that this failure rate is not the case.

Solid Information Technology also points to a Japanese customer that has been running solidDB for two-and-a-half years with no failure.

Scalability

The scalability of solidDB is multidimensional. First, it is scalable across multiple cores in a processor. Tests described below indicate that solidDB can not only handle thousands of transactions per second per core but that it is linearly scalable up to at least eight cores.

In addition, new database instances can be initiated on demand. They can then act as replicas for the master database to provide on-demand scaling of read-only applications.

solidDB Performance

Solid Information Technology uses the TM1 benchmark to measure the performance of solidDB. The TM1 (Telecom One) benchmark is a widely recognized benchmark in the telecommunications industry and is used to provide a comparative measure of performance in telecommunications applications. It is used by companies such as Sun, IBM, Intel, Nokia, and AMD for this purpose. TM1 is available as open source from www.soliddb.com.

TM1 emulates typical telecommunication transactions such as those used in home location registers that are characterized by short, predominantly read-only transactions. It comprises seven different transactions that exercise read, insert, update, and delete operations with a read/write ratio of 80/20.

TM1's metric is Mean Qualified Throughput (MQTh). MQTh is essentially a measure of transactions per second handled by the system. However, as opposed to OLTP benchmarks such as TPC-C, it counts both committed transactions and aborted transactions.

TM1 is run at a speed to saturate the target server. In a test with 32 concurrent clients and 100,000 subscribers running on dual-core AMD Opteron processors with 64-bit SUSE Linux, solidDB showed the following TM1 performance:

two core	6,518 tps
four core	18,449 tps
six core	36,533 tps
eight core	66,910 tps

In another TM1 test, updates made to an IBM DB2 database were replicated to four instances of solidDB, each running in its own T HS20 blade. The DB2 database alone supported a transaction rate of 1,476 tps. Each solidDB blade supported a transaction rate of 15,000 tps per blade for a total of 60,000 tps. Response time to TM1 transactions was less than one millisecond.

Platform Support

solidDB runs on Linux, Windows, AIX, and HP-UX platforms. It is integrated with several popular availability managers, including HP Openview, HP Serviceguard, Sun Netra, and GoAhead SelfReliant.

Utilities

solidDB provides several important utility functions, including:

- Solid SpeedLoader, a bulk loading utility for initially populating a database.
- Online backup, which uses snapshots to provide a consistent copy of the database while it is in full operation.
- Solid Export, a utility for fast exporting of consistent views of the database.

Solid Information Technology

Solid Information Technology was founded in Helsinki, Finland, in 1992. Its first product release was a disk-resident database in 1994. In the late 1990s, the company recognized the need in the telecommunications industry for high-speed, highly reliable, in-memory databases and introduced solidDB. solidDB currently has over 3,000,000 deployments made by hundreds of customers, including Alcatel, Cisco, EMC^2, Nokia, HP, and Siemens. Applications include telecommunications, retail, finance, healthcare, online shopping, and mobile access to central data.

Solid Information Technology has its corporate offices in Cupertino, California, and regional offices in America, Europe, and Asia.

In December, 2007, Solid Information Technology was acquired by IBM. It will be integrated into IBM's Information Management Software Division.