ENEA POLYHEDRA®: Polyhedra IMDB and High-Availability

Nigel Day
Polyhedra Product Manager

Integrated data management services are an integral part of any advanced network equipment software platform. Where large application-level data storage is required, say for an HLR (Home Location Register) or accounting/billing system, the data management solution may come in the form of a traditional disk-based relational database management system (RDBMS).

Many network infrastructure systems, however, require a separate, higher performance, in-memory database solution to manage the dynamic control, status and configuration information required for continuous "five nines" service delivery.

Enea Polyhedra provides a full-featured, fault-tolerant SQL RDBMS that is optimized for network control plane applications with demanding availability, performance and memory requirements. Equally important, Polyhedra works hand-in-glove with the Enea Accelerator platform, a carrier class software platform for building highly differentiated network equipment that delivers high-quality multimedia services over IP networks with 100% service availability. Featuring best-in-class networking, supervision, fault management, and database management, this flexible, reusable platform enables equipment makers to build scalable, upgradeable, "five nines" equipment that greatly reduces CAPEX and OPEX for service providers.

High-Availability?

High-availability and fault tolerance are phrases that are often used rather loosely, especially when it comes to software. In practice, it is not software that is fault tolerant or highly available. It is the complete system. So these terms, when applied to software, refer to the functions and features that the software must supply in order to facilitate the design of an HA system.

Embedded systems requiring high availability - say 'five 9s' (99.999%) or better of continuous operation – are typically configured with redundant cards, power supplies, etc. The individual components are not designed to be fault free. Rather, the system as a whole is designed in a redundant fashion that allows it to survive the failure of individual components. Redundancy, in addition to improving availability, also enhances flexibility, enabling components to be swapped out and upgraded without upsetting the running system.

While live upgradeability is not needed in some HA systems (for example, systems on board aircraft, which are turned off when the flight is over or the aircraft is serviced), it is crucial for applications like telecom where the systems must operate continuously for years on end. For example, in telecom infrastructure equipment, a field upgrade should not disrupt calls in progress. Similarly, in an industrial process control systems, stopping a production line for an upgrade could cost tens or hundreds of thousands of dollars of lost production. There might also be safety issues.

The data handling needs of HA embedded systems often have the following requirements and characteristics:

- Data changes must be fast, but quickly accessible.
- Data must be preserved in the face of a complete system failure, and must continue to be available in the face of a partial failure.
- The data structures, queries and types of changes are relatively static in a deployed systems, but will change frequently during application development, and will usually change when deploying new versions of the application.
- Field upgrades should not require scheduled downtime, and the period of vulnerability to component failure should be minimized.
- The system should minimize the amount of application coding needed to handle the HA configuration.
- The application should be scalable, avoiding performance killers such as polling.

Atomicity, Consistency, Isolation and Durability

For transactional data stores, reference is often made to the term 'ACID', a mnemonic for Atomicity, Consistency, Isolation and Durability. Atomicity means that each transaction either fails (leaving the data in the pre-transactional state) or fully commits - no halfway houses.

Consistency says that each transaction leaves the data store in a clean state, with all integrity conditions preserved. If not, the transaction is aborted, leaving the data in its pre-transactional state. Isolation says that transactions are independent, each completing before the next one is started (though judicious use of locking by the system can avoid the need for the implementation to be quite so restrictive).

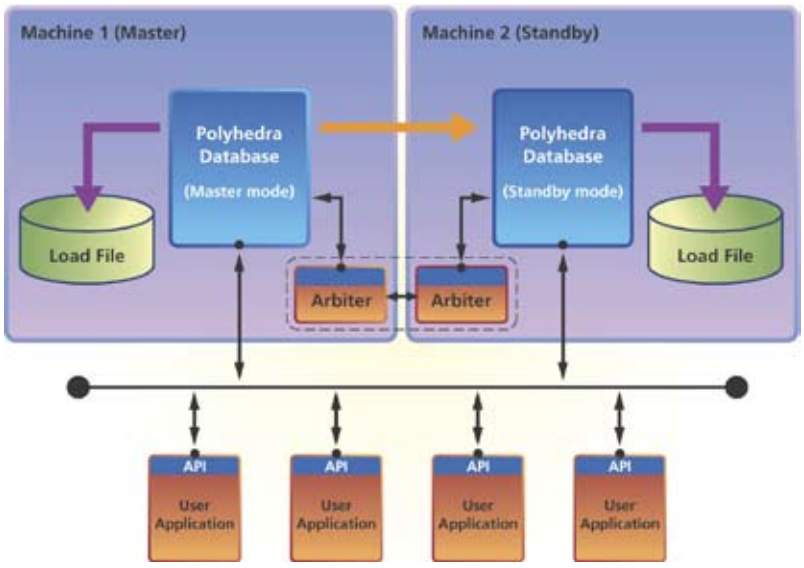
Durability means that if the data store says the transaction is complete, then the data is in a 'safe' state and will survive a subsequent system failure (within the capabilities of the underlying hardware and environment). In practice, the Durability requirement can significantly slow down a system (flushing files to disk can take some time, for example), so most data stores allow the level of durability to be tuned, balancing it against overall system responsiveness when operating normally. All true relational database systems are transactional, and offer a degree of ACID compliance. Most are also SQL based, and support on-the-fly schema changes. However, they may lack suitable HA characteristics, or be too slow for use in embedded systems.

Polyhedra IMDB and HA

The Polyhedra in-memory database system is designed for use in embedded systems, where state and configuration information has to be kept readily accessible, rapidly alterable… and safe. To ensure fast access, Polyhedra keeps the data in RAM, but backs it up using a variety of configurable mechanisms, including snapshots, transaction journals, and even hot standby where appropriate.

To support fault-tolerant configurations, Polyhedra provides a hot standby mechanism for fault-tolerant pairs. On start up, the master server gives the hot standby server a complete copy of the database (including the schema and any CL code attached to the database). Once it is fully running, the master sends copies of each transaction record as soon as it commits to the transaction. This keeps the standby up to date so that it can take over at a moment's notice.

The master-standby pair runs under the control of an external arbitration mechanism, which controls when failover occurs, and decides which is the master when both come up together after a cold boot.



Enea Polyhedra IMDB and HA.

When only a few systems are being installed, a software-only solution is possible, with a third machine used for arbitration. More complex configurations, however, usually require a two-board solution, with a hardware mechanism to assist in determining which board should be master. Techniques vary, but the aim is to provide a reliable way for one board to know whether it should be master, and to avoid having both boards operate in master mode, even when the data connection between the two is not working.

Standard Polyhedra release kits provide sample applications for both two-board and three-board solutions. For a 2-board solution, however, the code would need to be tailored to interact with the customer-supplied hardware-based arbitrator.

The Polyhedra client-server protocol features a client-controlled heartbeat mechanism, which allows communication failures to be detected quickly if the underlying transport is unable to provide timely information about the failure of the server. For example, when TCP/IP is used, a process failure would normally be detected by the operating system, which would then close any open ports. In the case of a complete board failure, however, it is impractical to wait for the TCP/IP stack at the other end to detect the problem, as this could take up to a half an hour.

The Polyhedra client libraries allow users to set up fault-tolerant connections to a list of servers. If the connection to the master server of a fault-tolerant pair fails (whether reported by the transport or by the heartbeat mechanism described above), the library will automatically and repeatedly try all the servers in the list in turn. The delay between attempts is configurable, as is the maximum retry count.

Once the connection is re-established to the current master, the library will re-establish all open active queries, and work out the 'delta' between the previous state and the current one. Consequently,

only minimal code changes are needed to convert a Polyhedra client application to one that works with an HA database configuration. The change could be as simple as changing the code that opens the initial connection. If needed, clients can monitor the changing status of the database connection and fine-tune the configuration parameters.

Polyhedra allows client applications to control transaction 'durability'. Normally, the server acknowledges a successful transaction as soon as it has been fully committed in-store on the master. But clients can opt to have the response delayed until the transaction has been fully logged to disk (where enabled) and applied to (and acknowledged by) the standby (if it is running).

Field Upgrades

In continuously-running systems, there is often the need to change the software or data structures on the fly, with no downtime. A simple case in point would be the addition of a new type of line card to a telecom rack in a basestation. The new card may be very similar to an existing card, but need additional configuration information, or want to report additional status information. The simplest way of handling this would be for the software on the new line card to check the central database on startup. If the columns it wants are not present in the tables it uses, it can just create them using the 'add' form of the 'alter table command'. For example:

```
alter table linecard add
(config2 integer default 49, status2 integer)
```

Another approach would be to just add the columns without checking first if they exist. No harm will be done if they already exist, and the type can be checked when performing queries.

Once these changes have been made, the new line card can start its application as normal. Provided other clients have not used the 'select *' form of query when inspecting the tables, their active queries and prepared statements will not be invalidated by this change, and they will see no interruption in the database service. All that needs to be done is to update the software on the management computers. This allows the clients to set the new configuration columns and monitor the new status columns.

The heterogeneity built into Polyhedra, together with its high level of inter-version interoperability, means that new line cards do not have to use the same operating system or processor type as the cards in the database server. It doesn't even have to use the same release version of the Polyhedra software.

More complex changes come in two categories: changes to the application software on the control cards or line cards; and changes to the underlying software used by the application software. Let us consider these two cases separately, starting with the simplest one.

Upgrading the Polyhedra Database Software

From time to time, it may be necessary to upgrade the Polyhedra code on a system to a later version. For example, a new version of the application software may want to take advantage of a Polyhedra feature that was not present in the currently used version. Or, the new version of the application may incorporate a bug fix that was adversely affecting the application.

In both cases, the inter-version interoperability principles enforced by Polyhedra greatly simplify the upgrades process. These principles are:

- Old (already-built) clients can connect to the new server.
- New clients can connect to older servers and use the features they provide. For example, when using ODBC, client applications can interrogate the database to find out the release information and to determine which features are implemented.
- New servers can read saved database files written by the previous release of Polyhedra.
- New servers can act as standby to a master running the previous release of Polyhedra.

Thus, to upgrade the control cards in a rack with a new version of the Polyhedra server code, the user simply:

- Tells the database server to produce a snapshot to a named file. This will cause the current state of the database to be recorded on both systems (in case of problems with the upgrade). In practice, this is unlikely, but the belt-and-braces approach should be ingrained in those developing and installing HA systems!
- Stop the standby card, install the new version of the Polyhedra server software, and restart the card. The database server will be told it was standby and connect to the master to obtain a new copy of the database. If the new software used a different format for the saved database, it would automatically create a local copy in the new format. The standby would then be able to receive transaction logs from the master, which would be applied to the local database to keep it in step with the master.
- The standby card would now be promoted to master, causing the server on the old master to relinquish control. The old master can then be upgraded in turn, and started up (as a standby, naturally).

If it is necessary to upgrade the client software on the line cards, this can be done either before or after upgrading the server code, depending on which is more convenient. In many cases, though, the client software will only need upgrading in the rare event that the application is affected by a bug in the Polyhedra client libraries.

There is no need for all the system components to use the same version of Polyhedra. In fact, the client-server protocol is common to all members of the Polyhedra DBMS family, so they share the library code. Thus, there would be no need to upgrade the code on the line cards when upgrading the

database on the control cards – say, from from Polyhedra32 to Polyhedra64.

Upgrading the Application Code

If the new application code needs a new version of Polyhedra, it may be simpler to do this first, as a separate stage, using the procedure outlined above. Once the system is up and running the correct version of Polyhedra on both the master and standby, the database schema can be updated. The changes will automatically be applied on both the master and the standby.

Polyhedra allows schema changes to be grouped together into a single transaction using the 'alter schema' command. The changes are checked for correctness before execution, and if there are any problems (such as an incompatibility of column names, or lack of room for temporary structures when transforming the database), the database will revert to its earlier state.

Data Management, Part of Enea's Total Solution

Data management is a key factor in the design of any embedded network system. Polyhedra was developed from the ground up to address the data management needs of HA network applications, combining the benefits of standard SQL database technology with a powerful set of features that enhance reliability, performance, memory efficiency, and data quality. These features make Polyhedra ideal for a broad range of applications, from fault-tolerant distributed infrastructure systems, to mobile devices with tight memory and power constraints.

Enea recognizes the need for integrated data management services as part of an overall solution for advanced software platform development. That's why Polyhedra works hand-in-glove with Enea's operating system and middleware products. And that's why Polyhedra is an integral part of Enea's system-level Accelerator platforms.

The Polyhedra family provides an optimized high-reliability SQL database solution for a broad range of embedded applications, whether the priority is high availability or low memory. Using a commercial, standards-based database like Polyhedra simplifies modeling and design, and requires fewer specialists, all of which speed development and reduce development cost. This is particularly true for distributed applications that run on heterogeneous hardware and operating system platforms.

All instances of Polyhedra databases can communicate transparently with each other, regardless of version. This not only simplifies the design of distributed applications that require multiple RDBMSs, but also provides a seamless upgrade path that enables equipment makers to take advantage of the latest database technology without having to make substantial changes to their existing application code.

Data consistency and integrity can easily create problems over time if not addressed properly with a suitable data management solution. The days when a database was regarded as overhead for embedded systems are long gone. With Polyhedra's unique combination of reliability, tunable high performance and other advanced features, data management need no longer be a maintenance burden.

ENEA