C-JDBC: Scalability and High Availability of the Database Tier in J2EE environments

Emmanuel Cecchet
INRIA Rhône-Alpes, Sardes project
emmanuel.cecchet@inrialpes.fr

Julie Marguerite
ObjectWeb Consortium
julie.marguerite@inrialpes.fr

1. Motivation

J2EE application performance is quickly bounded by database throughput especially when using open source software on commodity hardware. Nowadays, database scalability and high availability can be achieved but at very high expense. Existing solutions require large SMP machines or clusters with a Storage Area Network and high-end RDBMS. Both hardware and software licensing cost makes those solutions only available to large businesses and cannot address the open source users community.

2. Redundant Array of Inexpensive Databases

We introduce the concept of Redundant Array of Inexpensive Databases (**RAIDb**). RAIDb is the counterpart of RAID for databases. RAIDb aims at providing better performance and fault tolerance than a single database, at a low cost, by combining multiple database instances into an array of databases. As for RAID, a controller sits in front of the underlying resources. The clients send their requests directly to the RAIDb controller that load balance them among the set of RDBMS backends. The RAIDb controller gives the illusion of a single RDBMS to the clients.

Several RAIDb levels are defined to achieve various degree of performance and fault tolerance. *RAIDb-0* features full partitioning, *RAIDb-1* offers full replication and *RAIDb-2* provides partial replication.

3. C-JDBC: a RAIDb software implementation

JDBC™ is a Java API for accessing virtually any kind of tabular data. We have implemented C-JDBC (Clustered JDBC), a Java middleware based on JDBC, that allows to build all RAIDb level configurations. C-JDBC works with any commercial or open source RDBMS that provides a JDBC driver. The client application needs no modification and transparently accesses a database cluster like a centralized database. The RDBMS does not need any modification either. C-JDBC provides two components: a driver and a controller (see figure 1).

The **C-JDBC driver** is a standard JDBC 2.0 driver that replaces the database native JDBC driver used by the client. The C-JDBC driver forwards SQL requests to the C-JDBC controller and receives the corresponding results.

The **C-JDBC controller** is a regular Java application made of several components that implements the logic of a RAIDb controller. A virtual database is exposed to the driver and hides the distribution complexity. The ***authentication manager*** validates the login/password provided by the client and maps it on each database backends login/password. The ***scheduler*** receives the requests from the C-JDBC driver and synchronizes them to maintain a transactional-safe serializable order. An optional ***query cache*** can cache query results to minimize response time and improve performance scalability. The ***load balancer*** offers various degree of load balancing and fault tolerance according to the configured RAIDb level. Finally, the ***connection manager*** provides transparent connection pooling on top of database native JDBC drivers to access the backends. A ***recovery log*** manages checkpoints and allows for dynamic database backend adding or recovery.

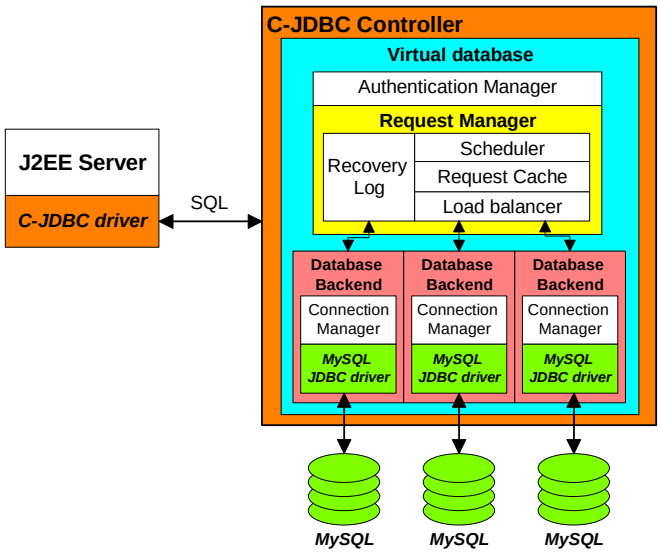

Figure 1. C-JDBC overview

4. Conclusion

C-JDBC is an open-source middleware that provides performance scalability and fault tolerance of the database tier in J2EE environments. C-JDBC is a software implementation of the RAIDb concept. It allows the user to control the degree of data replication and therefore the performance/fault tolerance tradeoff.

Preliminary results show that it is possible to obtain scalable results using open source software and commodity hardware. C-JDBC source code and documentation is available for download at *http://www.objectweb.org/c-jdbc*.