Oracle Berkeley DB Java Edition vs. Apache Derby: A Performance Comparison

An Oracle Technical White Paper
November 2006

ORACLE

Oracle Berkeley DB Java Edition vs. Apache Derby: A Performance Comparison

Oracle Berkeley DB Java Edition is an open source pure Java embeddable database that can dramatically outperform EJB persistence solutions by removing the overhead of SQL processing.

OVERVIEW

In the tests described in this white paper, Berkeley DB Java Edition performance exceeds Derby performance by up to a factor of 10.

Oracle Berkeley DB Java Edition (JE) is an open source, embeddable, transactional storage engine written entirely in Java. The innovative architecture of JE delivers excellent performance under highly concurrent read-intensive and write-intensive workloads. JE's design provides Java programmers with a highly scalable transactional storage engine. For most applications it can provide persistence capabilities with performance that far exceeds more traditional, relational, object or object-relational database solutions.

This paper compares the performance of Oracle Berkeley DB Java Edition to Apache's pure Java RDBMS, Derby. In some areas, this comparison is apples-to-apples, and in others it is apples-to-oranges. For instance, both JE and Derby are embeddable, transactional, pure Java storage engines (apples-to-apples). On the other hand, JE provides schema and a higher level data abstraction without SQL using a faster direct access API, features Derby provides with SQL (apples-to-oranges). The reader should evaluate whether the additional features provided by a relational database system (RDBMS) are required, at the expense of reduced performance. In all cases comparisons are functionally equivalent despite different underlying implementations.

INTRODUCTION

Berkeley DB Java Edition and the Direct Persistence Layer are compared to Apache Derby, a Java RDBMS, and Hibernate, an EJB3 compliant ORM solution.

JE and Derby both have a variety of APIs targeted to different application environments, so we compare the systems at two different levels. At the lower level, JE's base API (i.e. a Java `byte[]` based interface with get/put/cursor based access) is compared to Derby's JDBC interface and SQL. At a higher level, JE has a functionally equivalent API to EJB3 called the Direct Persistent Layer (DPL) for object storage and ad-hoc retrieval. When persisting objects and graphs of objects, Java programmers sometimes access RDBMS systems directly using JDBC and SQL. When that is the case, the preferred method is to map objects into tables using an object to relational mapping solution (ORM) such as Oracle TopLink which implements the EJB3 and JPA standards. However, Berkeley DB Java Edition is an excellent alternative when ad hoc SQL access is not expected during the life of the application. These two APIs, EJB3 and the DPL, are compared in some of the tests to demonstrate their relative performance overhead.

Apache Derby[1] is an embeddable, transactional, relational database management system (RDBMS) implemented in pure Java. It provides a standard SQL and JDBC based API with all of the features commonly found in other relational database systems. A common open source EJB3 solution for Derby is Hibernate[2]. We compare Apache Derby using Hibernate for ORM against Oracle Berkeley DB Java Edition using the DPL.

Performance is measured over a set of simple Create, Read, Update, Delete (CRUD) benchmarks. All measurements are expressed as throughput (operations/second), so higher bars in the graphs below indicate better performance (more operations per second). In the following sections, we briefly describe the test hardware that was used and the methodology for acquiring the performance data. This is followed by a description of each test and the results.

The results demonstrate that Oracle Berkeley DB Java Edition has excellent performance across all the various operations tested.

SYSTEM UNDER TEST

The Berkeley DB Java Edition measurements were performed on a Sun V20Z system with dual Opteron 244 (1.8GHz) processors, 6GB of memory, and a 73GB 10k RPM SCSI disk. The system runs Solaris 10 with the Java 1.5.05_05-b05 JVM. For the JE tests, the JVM heap size was set to 256MB and the JE cache size to 128MB. For the Derby tests, these values were 512MB and 128MB, respectively – twice as much memory as used with JE. We used Derby version 10.1.2.1, Hibernate 3.1.3 and JE 3.0.30.

---

[1] Derby is the name given to the open source version of IBM Cloudscape™ database (http://www.ibm.com/software/data/cloudscape/). Derby is managed as a project of the Apache Software Foundation (http://db.apache.org/derby/). Sun Microsystems™ packages and supports the same code within their Java EE distributions calling it Java DB (http://developers.sun.com/prodtech/javadb/).

[2] http://www.hibernate.org/

TEST RESULTS

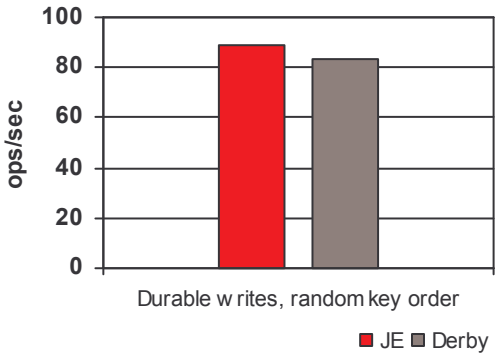
In all test cases Derby was run within the same JVM as the test code (direct mode) so there is no client/server or inter-process communication (IPC). Oracle Berkeley DB Java Edition always avoids this overhead by operating within the same JVM as the application. When interacting directly with Derby we use the **`PreparedStatement`** interface via JDBC. **`AUTOCOMMIT`** is off and for tests requiring durability we call the **`commit()`** method on the JDBC connection.

CREATE PERFORMANCE

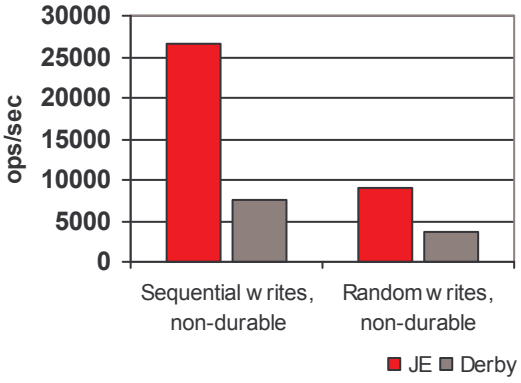
The first and most basic task of any database is storing new data. Berkeley DB Java Edition significantly out performs the competition for record creation performance.

The first and most basic operation of any database is adding new data. With SQL this is accomplished using an **`INSERT`** statement, with JE a **`put(key, data)`** operation. In both cases new data is written to disk. For our tests with Derby the keys (6 bytes) are inserted into a column designated as a primary key, and the data (294 bytes) into a second column. By doing this, Derby and most other RDMBS systems will keep a primary index based on the key. JE performs the functional equivalent of this with its BTREE implementation.

For both tests we use JE's base API and access Derby using SQL via JDBC. The on-disk write cache is disabled such that transaction commits will force data to be written to the disk itself.



The first graph shows the speed at which both products can add (insert or put) new data into their databases when committing each operation all the way to disk. While both products are disk bound, JE is still faster than Derby.



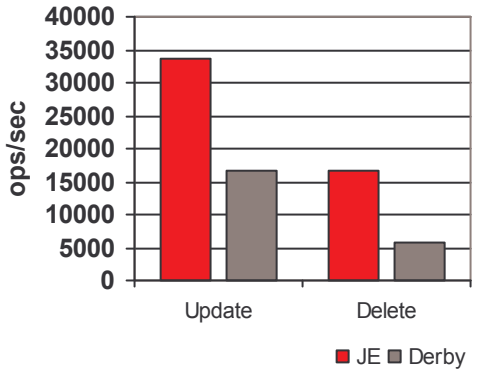
In the second graph we've removed the bottle neck of the disk by enabling the on-disk write cache. As expected the operations per second have gone up dramatically for both products. We test sequential and random order inserts (**`put()`** operations in JE) and here again JE is the clear winner over Derby.

UPDATE & DELETE PERFORMANCE

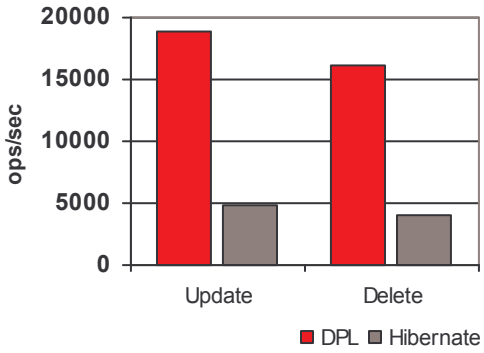
Once data exists in a database there is a good chance that it will eventually be changed or removed. The next two tests examine update and delete performance. The on-disk write cache is enabled.

Transactional updates and deletes can be a bottleneck in your application. Berkeley DB Java Edition can update and delete data much faster than the EJB based alternative and eliminate those performance issues.

The first graph shows updates and deletes using JE's `put(key, data)` call and the SQL `UPDATE` statement in Derby. For both of these tests, records are a 6 byte key and 294 bytes of data, with only a primary index. Each operation is a modification to the data portion (the 294 bytes) within its own transactional context. Note that again, in both cases, JE holds a substantial lead over Derby.



The second graph on this page shows updates and deletes as well, but this time comparing JE's Direct Persistence Layer (DPL) against the prescribed ORM (EJB3, JPA) method using Derby and Hibernate. This test shows the overhead of the ORM technique for storage as the performance gap between JE/DPL and Derby/Hibernate is even more pronounced than before.

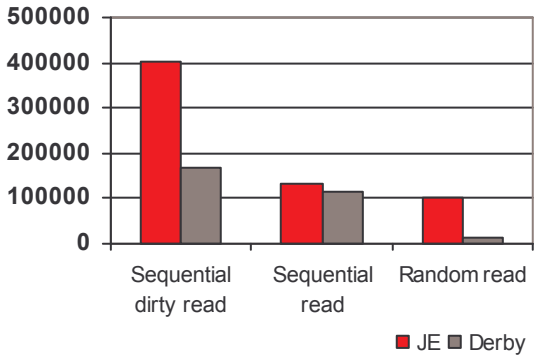


READ PERFORMANCE

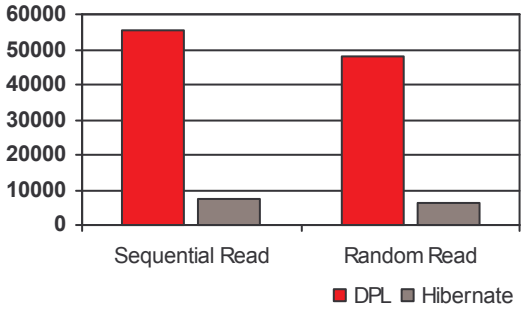
Read performance is critical for any database.

High speed access to data is a critical component of any Java persistence solution. Berkeley DB Java Edition demonstrates a significant performance advantage for reading Java object data when compared to EJB.

The first graph shows read performance in three different situations. A "dirty read" is when transaction isolation has been turned off, so the data which is read may be that of an in-flight transaction. This reduces locking overhead and is fast, but sacrifices transactional integrity. Sequential reads are table scans using cursors.



These graphs show that even when no ORM is present, JE's read performance is superior to Derby in all three cases, probably due to the overhead of SQL processing in Derby. When we compare read performance with the DPL layer and ORM (the second graph) the difference in performance becomes much more pronounced.



These tests demonstrate the overhead of using SQL from within an ORM for object persistence. This overhead can be eliminated in cases where the application's data is only accessed via the ORM and never using ad-hoc SQL.

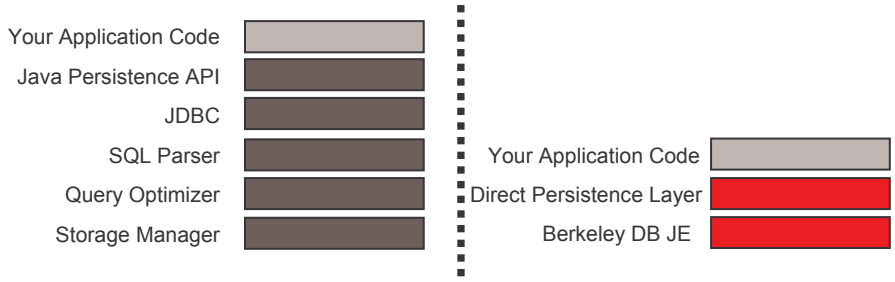
SQL AND OBJECT-RELATIONAL MAPPING OVERHEAD

Relational databases offer sophisticated tools for data storage and analysis. However, in most cases Java object data that is persisted with an ORM is never analyzed using SQL queries. Instead it is usually retrieved and reconstituted as Java objects. Using an RDBMS in this case introduces unnecessary overhead. The full analytical power of the relational model is not required to persist Java objects efficiently. Berkeley DB Java Edition does not have the overhead of a query language like SQL and so does not incur this penalty. This performance penalty is clearly demonstrated by the preceding performance results.

Relational databases, while a de-facto standard for Java persistence, incur a performance penalty on every database access if you don't need ad-hoc SQL queries. Berkeley DB Java Edition and the DPL eliminate those unnecessary layers and provide higher transactional thru-put as a result.

The performance difference demonstrated in these benchmarks also arises due to architectural differences between the two solutions. When using any RDBMS and ORM solution to store Java objects your application code is very distant from the underlying storage manager, which actually stores the data to disk. The penalty for SQL is this overhead; the payoff is an ability to execute queries against the data stored in the database. If, however, your application only uses SQL as a conduit for an ORM such as EJB3 and you never access that data in any other way then you are paying a large penalty for flexibility and features you are not using.

Contrast that with Berkeley DB Java Edition and its Direct Persistence Layer. Data moves directly from objects into the storage manager and onto disk. None of the intermediate steps are required, and so performance is dramatically better. Manageability is also simpler as there are fewer moving parts in the solution.



DURABILITY

Performance tests are by design used to illustrate differences between competing solutions and do not always mimic real world recommendations for the configuration of deployed systems. In our tests we have enabled the disk drive write cache in most tests to eliminate a common bottle neck. Writing to a hard disk drive can be non-deterministic and is the most time consuming (by orders of magnitude) operation of any database, but it is also generally a necessary component of durable commits. Commercial applications requiring durability should disable this cache or force it to flush to the disk as part of a transaction commit.

Databases in deployment generally service many operations at once. This concurrency allows both JE and Derby to interleave work while waiting for the disk to flush write transactions to the log.

For the purposes of these tests we wanted to examine the efficiency of the database itself while not constrained by disk thru-put which is why we chose to enable the disk cache, disabling it would have skewed the results.

DISCUSSION

These results clearly demonstrate the excellent performance of Berkeley DB Java Edition relative to Derby. It is important to mention that Derby is a SQL database system whereas Berkeley DB Java Edition provides a record level API. While SQL does provide additional flexibility in certain situations it can also add significant performance overhead. In a wide variety of situations, the added flexibility of SQL is outweighed by the runtime performance penalty. Berkeley DB Java Edition provides the application developer a simple, high performance API in a lightweight, robust, transactional database engine.

CONCLUSION

The Berkeley DB Java Edition Persistence API is a high performance, complete solution for Java object persistence. Berkeley DB Java Edition performance exceeds Derby performance in every test by up to a factor of 10. For use cases where ad hoc queries and direct SQL access to persistent object data is not required, these tests demonstrate how Berkeley DB Java Edition is a high performance alternative for Java object persistence.

The following sites have additional information, architectural overview, documentation, and the complete product download including source code.

Oracle Berkeley DB Java Edition Information and Download:
http://www.oracle.com/database/berkeley-db/je

Join the discussions and our community on the Oracle Technology Network:
http://forums.oracle.com/forums/forum.jspa?forumID=273

Oracle Berkeley DB Java Edition vs. Apache Derby: A Performance Comparison
November 2006

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