

Appendix A - Test result performance upgrade CopperCore for D4LD

This document describes the first performance results after improving the overall CopperCore performance. In order to test this performance of CopperCore we used different setups of the environment posted by Alex Little, as was used by the OUUK on their server. This environment was using the latest version of CopperCore with the HSQL database engine. We tested the system using different database engines and software versions. The first step was converting all data from HSQL to SQLServer and PostgreSQL allowing the same test to be run on all three database engines. The table below shows all results.

Machine	DataBase	Version	GetRoles	WIG	CLFriday
XP (198)	Hsql	Original	15"	34"	2'08"
2000 (196)	Hsql	Original	14"	31"	2'09"
2000	Hsql	+index	15"	44"	2'45"
2000	Hsql	Query EAR	1.8"	1.8"	2.5"
2000	Hsql	Query EAR + index	1.4"	1.8"	2.6"
2000	SQL2K	Original	1.8"	2.6"	5.8"
2000	SQL2K	+index	1.8"	3.0"	5.4"
2000	SQL2K	Query EAR	1.9"	2.5"	4.2"
2000	SQL2K	Query EAR + index	1.8"	2.5"	3.5"
2000	Postgres	Original	1.7"	3.0"	5.5"
2000	Postgres	+index	3.0"	3.1"	5.1"
2000	Postgres	Query EAR	2.2"	3.2"	6.4"
2000	Postgres	Query EAR + index	2.6"	3.1"	3.9"
2000	Postgres	server hotspot + 5.08	3.6"	4.1"	9.8"
2000	Postgres	client hotspot + 5.08	2.7"	2.9"	3.9"
2000	Postgres	Pk cache	1.6"	2.2"	2.3"
2000	Hsql	Cache + jboss 4.0.4	1.2"	1.6"	2.4" (2.1" cached)
2000	Postgres	PK + jboss 4.0.4	1.2"	1.8"	2.9" (2.5" cached)

For each test three actions were measured. Firstly, we measured how long it took to retrieve the first page containing all unit of learnings, roles etc. using Sled. The timing of these are captured by the column GetRoles. Next it was measured how long it took to retrieve the first page of the 'What is Greatness' course. The results are captured by the column with heading 'WIG'. Last, the timings of retrieving the first page of the 'CLFriday' course were captured by the column with 'CLFriday' heading. We used an old computer with an 800Mhz Pentium III processor and 256 MB of RAM. The measured values should be regarded as indications rather than hard values. All tests were performed and timed by hand!

The first test we did was using a kind of reference machine. The purpose of this test was to determine if we could expect great differences in performance in any particular setup of a computer. The hardware was exactly the same as of the computer we did all other tests with. However it was using a different operating system. The second test was done on the actual test computer. Results show that the performance of the system is not greatly influenced by the particular setup of the computer. We could also replicate the terrible slow response times Alex mentioned earlier.

We executed the same test on the PostgreSQL and SQL Server configurations and we got much better performance for both these databases. So we conclude something was wrong with the HSQL configuration. We analyzed the database and added indexes on all tables we thought would profit from such an index. By doing other measurements we saw a boost in raw database performance but this helped rather little in overall performance.

By analyzing the HSQL performance in more detail we were able to conclude that HSQL does a very bad job in optimizing queries when using Views and joins. Therefore we removed the use of views in our code and split the join of a particular query into two separate queries. As the table shows, this improved the HSQL performance dramatically and helps with SQL Server as well but decrease performance of PostgreSQL marginally. Next we combined the two measures, index and new software. This improved performance in all situations.

Next we implemented a new cache for caching primary keys of the entity beans. This results in fewer database accesses of the J2EE layer. Because this measure does not involve any database specific element, we did not measure the results for all databases. The results show that caching increased performance by up to 60%.

We tried to use Server HotSpot to see if this would lead to any performance gains. With our test setup we could not measure any benefits. In fact we saw a performance decrease.

Finally we tested a migration to JBoss 4.0.4 again using Sled (so Sled runs on JBoss 4.0.4). The performance improved again considerably as can be seen in the table.

Concluding can be said that by adding indexes to the databases, modifying some of the queries, adding a cache and migrating to JBoss 4.0.4 we improved performance by 50% in general and in the case of HSQL by considerably more.