CUBRID 9.1 QA Completion Report

This document is the verification report of CUBRID 9.1 in terms of functionality, performance and stability.

Table of Contents

1.Test Overview

1.1 Test Objectives

The objectives of this test are to perform functionality, performance and stability tests for the final release candidate build of CUBRID 9.1 (hereinafter referred to as 9.1), which is under development for release in March 2013, and to determine its release based on the test results. To test the stability of CUBRID, test environments were configured as described below. Based on comparisons between the performance test results of CUBRID 9.1 and those of CUBRID 9.0 Beta (hereinafter referred to as 9.0 Beta), we have tested to determine whether the performance of CUBRID 9.1 has improved or not.

- CentOS 5.6 (32/64-bit) or compatible
- CentOS 5.3 (32/64-bit) or compatible
- CentOS 4.7 (32/64-bit) or compatible
- Windows 2003 (32/64-bit) or compatible
- Final test build: 9.1.0.0212 (Linux 64-bit/32-bit, Windows 64-bit/32-bit)

1.2 Test Environment

1.2.1 Test Procedures

Tests to verify the CUBRID product are shown below. The test sequence used may be different from the one described here. To verify product stability, functionality, performance and other tests were performed for 4 types of builds as shown in the figure below. The details of each test suite are described in the appendix of this report.

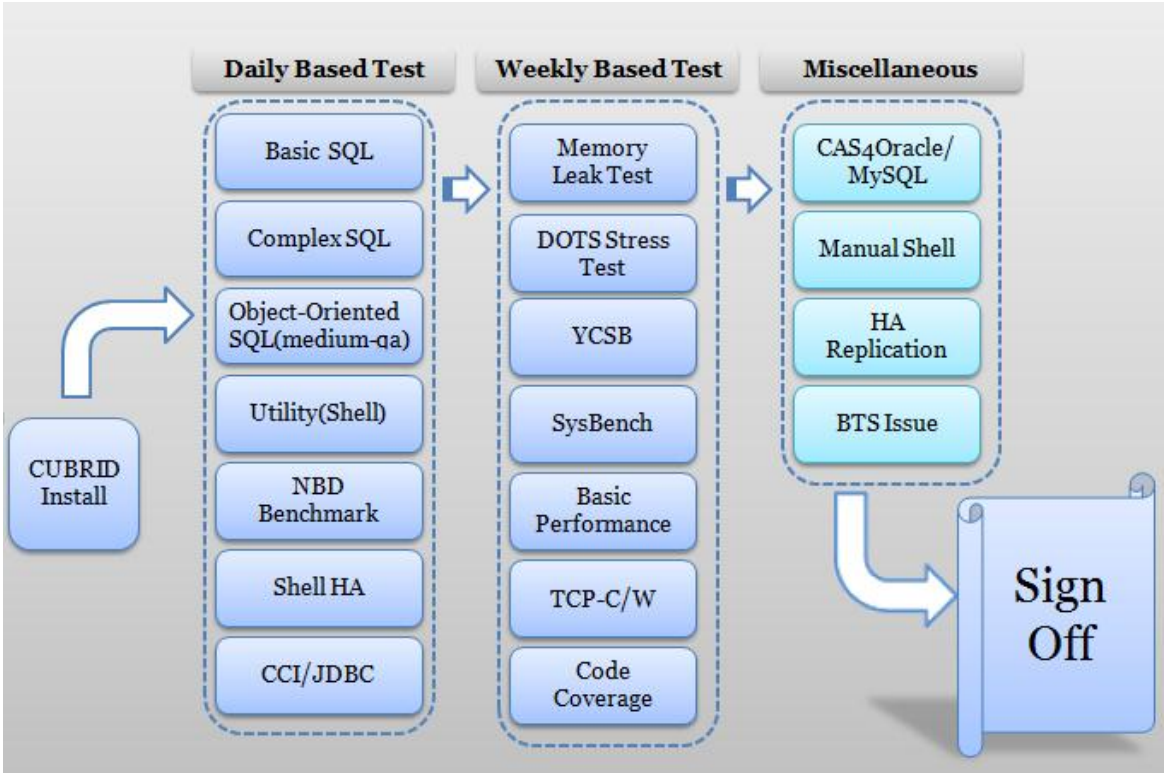


Figure 1. CUBRID Test Procedure

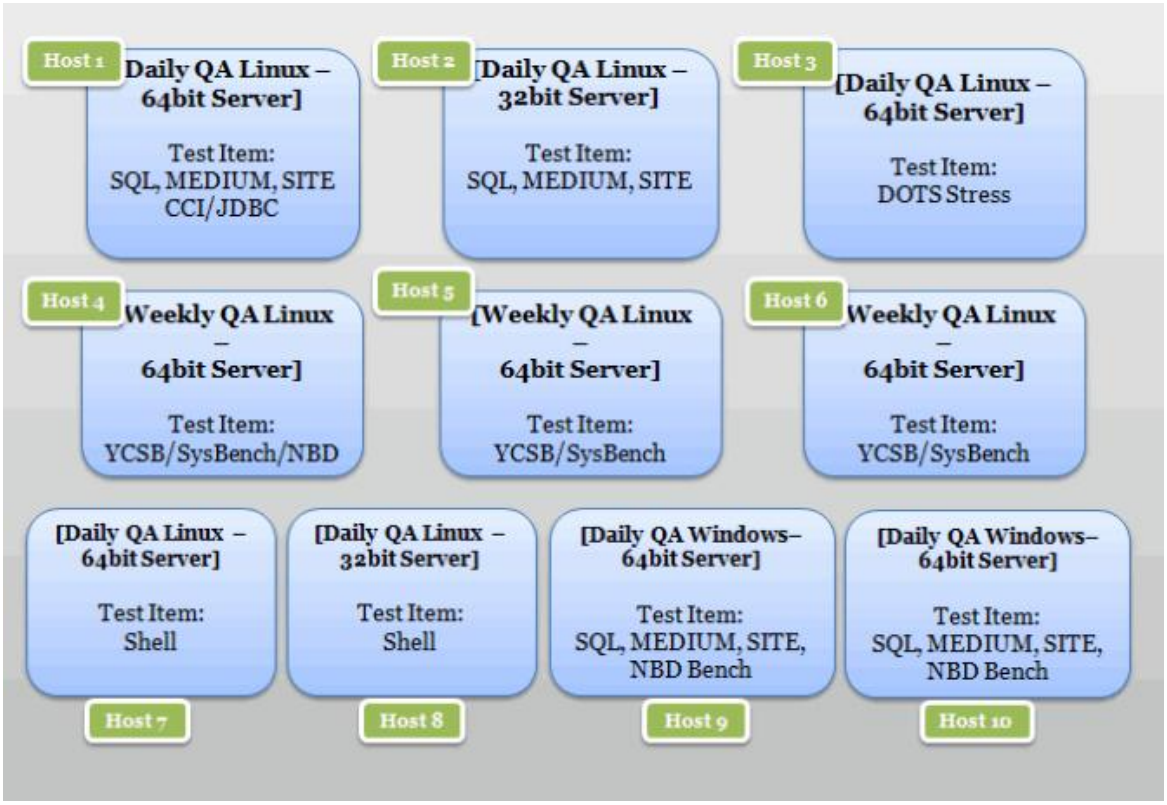


Figure 2. System Diagram for Basic Test

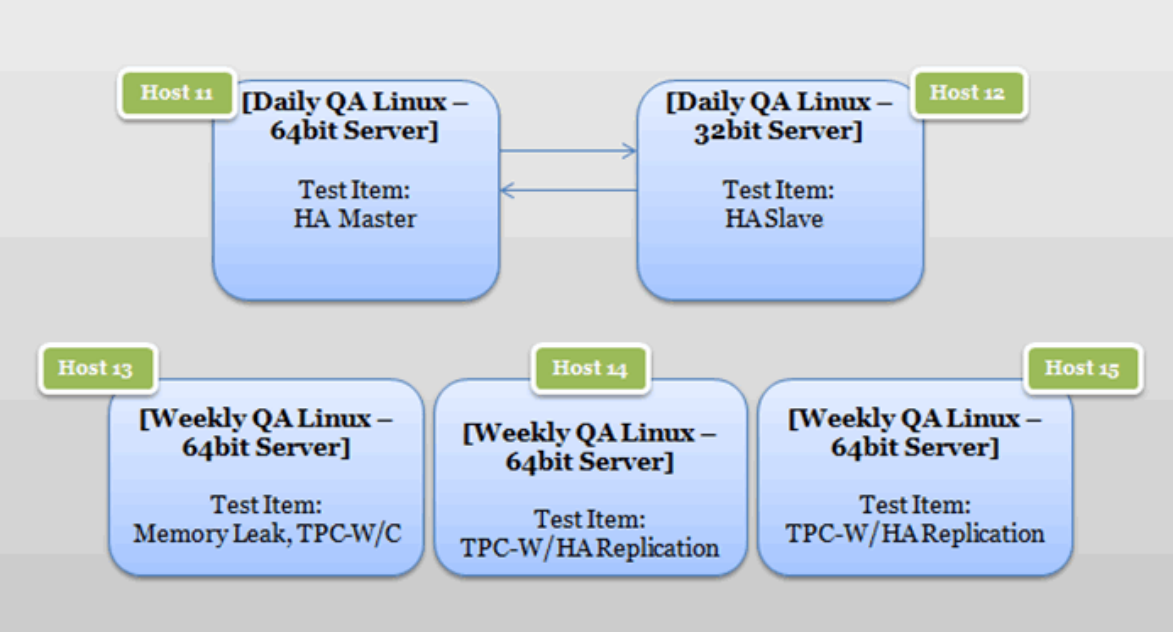


Figure 3. System Diagram for HA Test

1.2.2 Hardware Test Environment

Servers for the CUBRID test and their usage are listed in the table below.

Name	OS	CPU	MEMORY	DISK
Host 1	Cent OS 5.3 (64-bit)	Xeon(R) 2.4 GHz (12 cores) * 1	24 GB	SAS 600G * 3 (Raid5)
Host 2	Cent OS 5.3 (64-bit)	Xeon(R) 2.4 GHz (12 cores) * 1	24 GB	SAS 600G * 3 (Raid5)
Host 3	Cent OS 5.3 (64-bit)	Xeon(R) 2.4 GHz (12 cores) * 1	24 GB	SAS 600G * 3 (Raid5)
Host 4	Cent OS 5.3 (64-bit)	Xeon(R) 2.4 GHz (12 cores) * 1	24 GB	SAS 600G * 3 (Raid5)
Host 5	Cent OS 5.6 (64-bit)	Xeon(R) 2.4 GHz (12 cores) * 1	24 GB	SAS 600G * 3 (Raid5)
Host 6	Cent OS 5.6 (64-bit)	Xeon(R) 2.4 GHz (12 cores) * 1	24 GB	SAS 600G * 3 (Raid5)
Host 7	Cent OS 5.6 (64-bit)	Xeon(R) 2.4 GHz (12 cores) * 1	24 GB	SAS 600G * 3 (Raid5)
Host 8	Cent OS 5.3 (64-bit)	Xeon(R) 2.4 GHz (12 cores) * 1	32 GB	SAS 600G * 3 (Raid5)
Host 9	Windows 2003 (64-bit)	Xeon 2.33 GHz (quad cores) * 2	8 GB	SATA 500G * 2 (No Raid)
Host 10	Windows 2003 (32-bit)	Xeon 2.0 GHz (quad cores) * 2	8 GB	SATA 500G * 2 (No Raid)
Host 11	Cent OS 4.7 (64-bit)	Xeon 2.00 GHz (8 cores) * 2	8 GB	SATA 500G * 2 (No Raid)
Host 12	Cent OS 4.7 (64-bit)	Xeon 2.00 GHz (8 cores) * 2	8 GB	SATA 500G * 2 (No Raid)
Host 13	Cent OS 4.7 (64-bit)	Xeon 2.00 GHz (8 cores) * 2	8 GB	SATA 500G * 2 (No Raid)
Host 14	Cent OS 4.7 (64-bit)	Xeon 2.00 GHz (8 cores) * 2	8 GB	SATA 500G * 2 (No Raid)
Host 15	Cent OS 4.7 (64-bit)	Xeon 2.00 GHz (8 cores) * 2	8 GB	SATA 500G * 2 (No Raid)

1.3 Test Category

The following tests were performed to determine whether CUBRID 9.1 meets the criteria of release. The details of each test are described in the appendix of this report.

- Functionality tests
  - SQL query test
  - MEDIUM query test
  - SITE query test
  - Utility (Shell) test
  - HA Feature test
  - HA Replication test
  - CCI Interface test
  - JDBC Interface test
  - CAS4MySQL/Oracle
- Performance tests
  - Basic Performance Test
  - YCSB Benchmark
  - SysBench
  - NBD Benchmark
  - Data Replication Test on HA
  - TPC-C Benchmark
- Stability tests
  - DOTS stress test
  - TPC-W on HA test
- Compatibility tests
  - JDBC compatibility test
  - CCI compatibility test
- Live Service Simulation test
- Installation tests
- Other tests
  - Test for checking CUBRID 9.1 functionalities/bug fixes
  - Memory check
    - Execute SQL/MEDIUM with Valgrind
    - Execute SysBench with Valgrind
    - Execute TPC-C with Valgrind
    - Execute SQL with pmap to monitor cub_server

2.Test Results

2.1 Functionality Test Results

2.1.1 Basic Query Tests

This test was performed to verify the basic DBMS functionalities using SQL statements. SQL statements stored in 14,274 files have been executed to verify DBMS conformity. We have executed the stored SQL statements in a JDBC-based application and compared the results with the stored reference files for verification.

Table 1. Result of Basic Query Tests

Test Category	Number of Scenario Files	Number of Scenario Files passed	Pass Rate
SQL query test	12,091	12,091	100%
MEDIUM query test	970	970	100%
SITE query test	1,213	1,213	100%

2.1.2 Basic Utility and Other Scenario Tests

This test was performed to verify the basic DBMS functionalities using shell scripts. In particular, this test was also performed to verify CUBRID utilities that could not be tested by SQL statements. Scenarios of 1,560 shell scripts have been executed to verify DBMS conformity.

Table 2. Result of Basic Utility and Other Scenario Tests

Test Category	Number of Scenario Files	Number of Scenario Files passed	Pass Rate
Utility	225	225	100%
Bug regression	836	836	100%
Environment variable	7	7	100%
Other	492	492	100%

2.1.3 HA Feature Tests

Scenarios of 293 shell scripts have been executed to verify HA features and the regressions.

Table 3. Result of HA Feature Tests

Test Category	Number of Scenario Files	Number of Scenario Files passed	Pass Rate
Data replication test	5	5	100%
Bug regression	141	141	100%
Node fault test	16	16	100%
Process fault test	8	8	100%
Broker fault test	8	8	100%
Run replication test scenarios	115	115	100%

2.1.4 HA Replication Tests

HA Replication Test is a QA tool which runs SQL test cases on HA Master, and then verifies data consistency between Master and Slave. Scenarios of 12,182 SQL files have been executed to verify data consistency between Master and Slave.

Table 4. Result of HA Replication Tests

Test Category	Number of Scenario Files	Number of Scenario Files passed	Pass Rate
Test Cases migrated from SQL suite	12,091	12,091	100%
Bug regression	91	91	100%

2.1.5 CCI Interface Tests

CCI Interface Test is to verify if all the CCI APIs of CUBRID can work well as described in the CUBRID manual. Scenarios of 250 shell scripts have been executed to verify all the CCI APIs and the regressions.

Table 5. Result of CCI Interface Tests

Test Category	Number of Scenario Files	Number of Scenario Files passed	Pass Rate
Basic features	207	207	100%
Bug regression	43	43	100%

2.1.6 JDBC Interface Tests

Scenarios of 1,529 shell scripts have been executed to verify all the JDBC APIs and the regressions.

Table 6. Result of JDBC Interface Tests

Test Category	Number of Scenario Files	Number of Scenario Files passed	Pass Rate
Features test	1,529	1,529	100%

2.1.7 CAS4MySQL/Oracle Tests

Scenarios of 64 shell scripts have been executed to verify the features of CAS4MySQL and CAS4Oracle.

Table 7. Result of CAS4MySQL/Oracle Tests

Test Category	Number of Scenario Files	Number of Scenario Files passed	Pass Rate
CAS4MySQL	30	30	100%
CAS4Oracle	34	34	100%

2.2 Performance Test Results

2.2.1 CUBRID Basic Performance Test

This test was performed to check the performance of the CUBRID DBMS basic operations, which are select, insert, update and delete. For more information about test scenarios, see the appendix II. For all the configuration variables, except for SQL_LOG=OFF in cubrid_broker.conf, default configuration values were used. As shown in the table below, we can find that the performance of basic performance test is almost same as the results of 9.0 Beta. Based on the results we can say that 9.1 is a quite stable version.

A. Linux: Performance Comparison between 9.0 Beta and 9.1 (64-bit)

We can find that the performance of INSERT, UPDATE, SELECT and DELETE operations are almost same as that of 9.0 Beta, which means that 9.1 is a quite stable version.

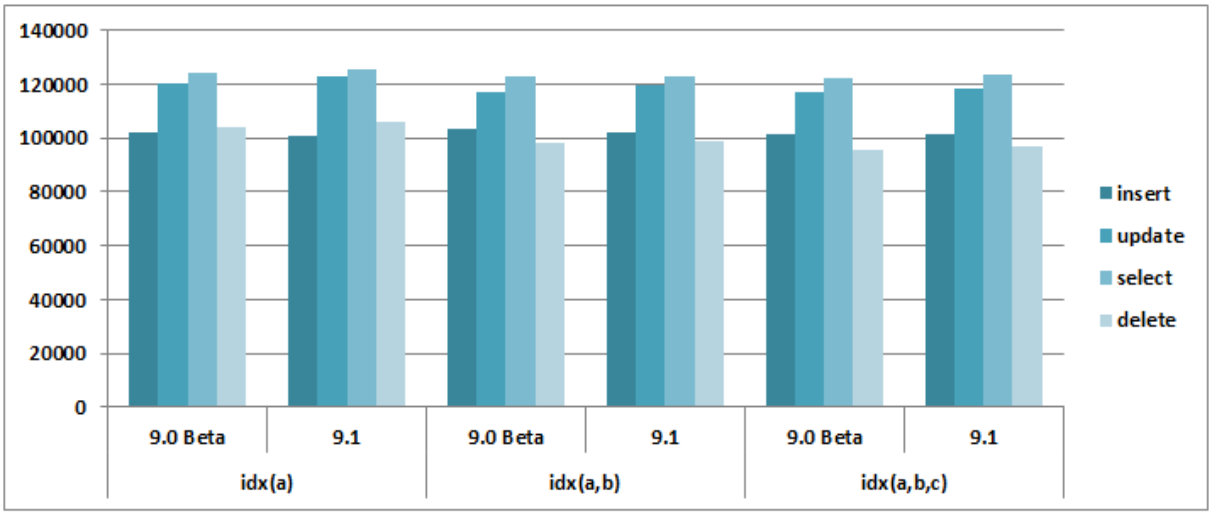


Figure 4. Performance Comparison between 9.0 Beta and 9.1 (Linux 64-bit)

Table 8. Performance Comparison between 9.0 Beta and 9.1 (Linux 64-bit)

	idx(a)			idx(a,b)			idx(a,b,c)		
	9.0 Beta	**9.1**	**Ratio**	**9.0 Beta**	**9.1**	**Ratio**	**9.0 Beta**	**9.1**	**Ratio**
Insert	101,849	100,692	99%	103,715	102,328	99%	101,529	101,237	100%
Update	120,147	122,786	102%	117,345	119,770	102%	116,842	118,623	102%
Select	124,448	125,633	101%	122,744	122,971	100%	122,331	123,411	101%
Delete	103,827	106,302	102%	98,011	99,102	101%	95,799	96,944	101%
Total	450,271	455,413	101%	441,815	444,171	101%	436,501	440,215	101%

(Unit: TPS)

B. Linux: Performance Comparison between 9.0 Beta (32-bit) and 9.1 (32-bit)

We can find that the performances of all operations are same as that of 9.0 Beta.

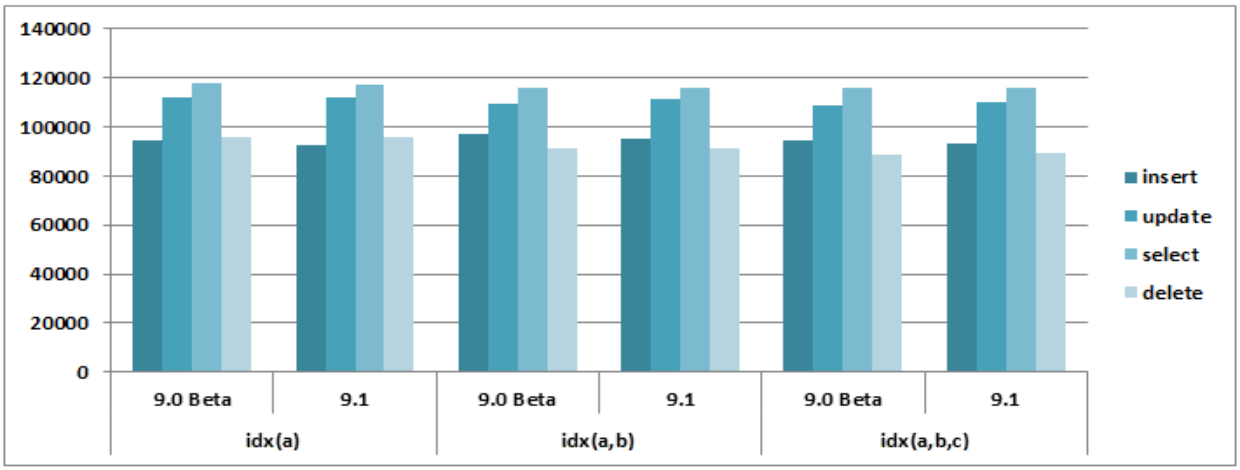


Figure 5. Performance Comparison between 9.0 Beta and 9.1 (Linux 32-bit)

Table 9. Performance Comparison between 9.0 Beta and 9.1 (Linux 32-bit)

	idx(a)			idx(a,b)			idx(a,b,c)		
	9.0 Beta	**9.1**	**Ratio**	**9.0 Beta**	**9.1**	**Ratio**	**9.0 Beta**	**9.1**	**Ratio**
Insert	94,571	92,718	98%	97,073	95,408	98%	94,711	93,105	98%
Update	112,309	112,089	100%	109,549	111,148	101%	108,643	110,331	102%
Select	118,142	117,236	99%	116,189	115,815	100%	115,766	116,079	100%
Delete	96,056	95,641	100%	91,145	91,219	100%	88,509	89,212	101%
Total	421,078	417,684	99%	413,956	413,590	100%	407,629	408,727	100%

(Unit: TPS)

C. Windows: Performance Comparison between 9.0 Beta (64-bit) and 9.1 (64-bit)

Performances of all operations have shown little change from 9.0 Beta.

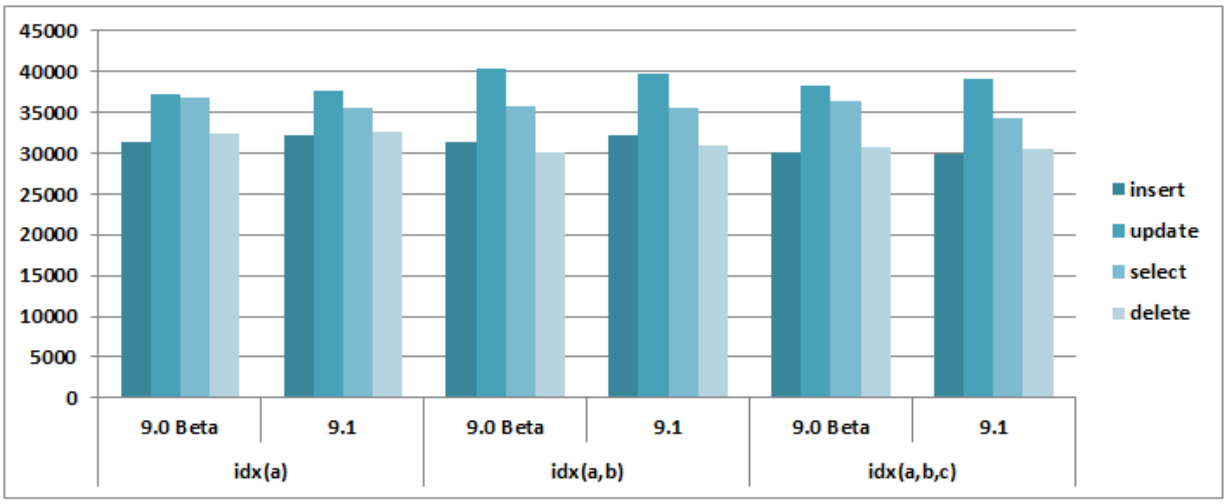


Figure 6. Performance Comparison between 9.0 Beta and 9.1 (Windows 64-bit)

Table 10. Performance Comparison between 9.0 Beta and 9.1 (Windows 64-bit)

	idx(a)			idx(a,b)			idx(a,b,c)		
	9.0 Beta	**9.1**	**Ratio**	**9.0 Beta**	**9.1**	**Ratio**	**9.0 Beta**	**9.1**	**Ratio**
Insert	31,257	32,091	103%	31,356	32,095	102%	30,136	29,997	100%
Update	37,229	37,583	101%	40,359	39,672	98%	38,361	39,016	102%
Select	36,776	35,473	96%	35,648	35,542	100%	36,280	34,270	94%
Delete	32,470	32,531	100%	30,204	30,843	102%	30,700	30,422	99%
Total	137,732	137,678	100%	137,567	138,152	100%	135,477	133,705	99%

(Unit: TPS)

D. Windows: Performance Comparison between 9.0 Beta (32-bit) and 9.1 (32-bit)

According to the test result, we can say that there is no significant change of performance on Windows 32-bit OS.

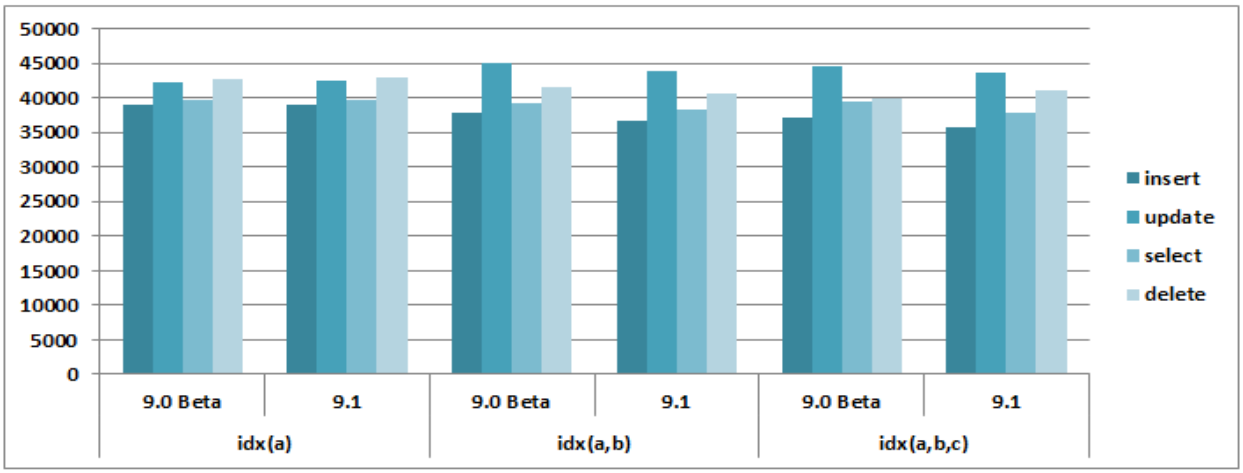


Figure 7. Performance Comparison between 9.0 Beta and 9.1 (Windows 32-bit)

Table 11. Performance Comparison between 9.0 Beta and 9.1 (Windows 32-bit)

	idx(a)			idx(a,b)			idx(a,b,c)		
	9.0 Beta	**9.1**	**Ratio**	**9.0 Beta**	**9.1**	**Ratio**	**9.0 Beta**	**9.1**	**Ratio**
Insert	39,119	38,928	100%	37,978	36,642	96%	37,153	35,775	96%
Update	42,173	42,544	101%	45,029	43,914	98%	44,509	43,765	98%
Select	39,681	39,826	100%	39,340	38,327	97%	39,390	37,814	96%
Delete	42,705	43,089	101%	41,610	40,713	98%	40,003	41,187	103%
Total	163,678	164,387	100%	16,3957	159,596	97%	161,055	158,541	98%

(Unit: TPS)

2.2.2 YCSB Performance Test

YCSB as a framework for benchmarking system is popular in the world (see also https://github.com/brianfrankcooper/YCSB/wiki). This test was performed to verify CUBRID performance of not only basic operations but also compositive operations, which are insert, select, scan, update and the mix of them. For more information about test scenarios, see the appendix II. As shown in the results below, the performance of SELECT operation has improved nearly 20%, and the performance of the other operations is almost same as that of 9.0 Beta.

A. Master Server Configuration: Performance Comparison between 9.0 Beta (64-bit) and 9.1 (64-bit)

Table 12. Result of YCSB Benchmark (Master Server)

	Throughput(OPS)			Average Latency(ms)		95th Percentile Latency(ms)	
Operations	**9.0 Beta**	**9.1**	**Ratio**	**9.0 Beta**	**9.1**	**9.0 Beta**	**9.1**
Insert	15,883	15,854	100%	18	18	31	32
Select	28,697	34,086	119%	10	8	26	27
Scan	4,481	4,407	98%	60	61	244	245
Update	13,948	13,802	99%	20	21	18	17
Mix	14,233	14,274	100%	N/A	N/A	N/A	N/A

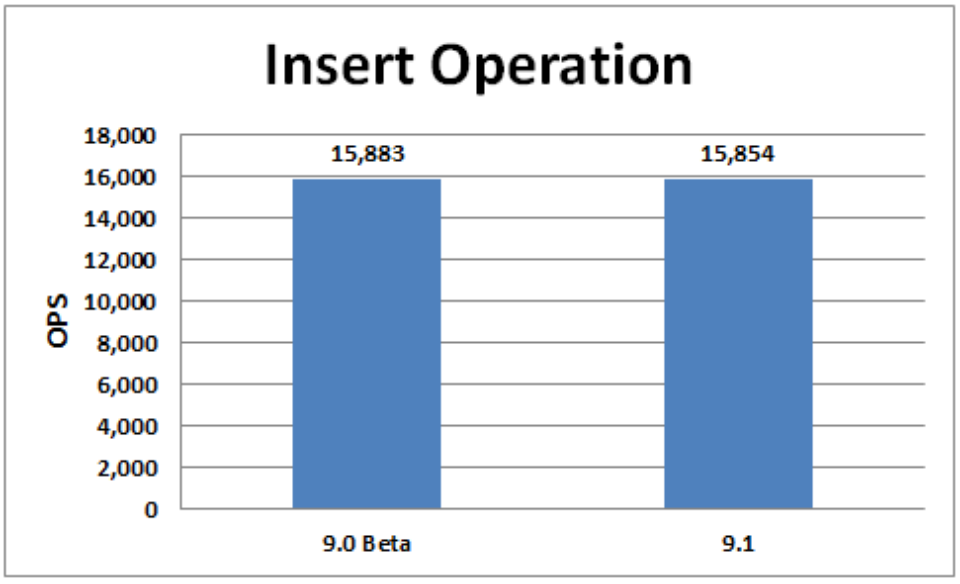


Figure 8. Result of Insert Operation of YCSB Benchmark (Master Server)

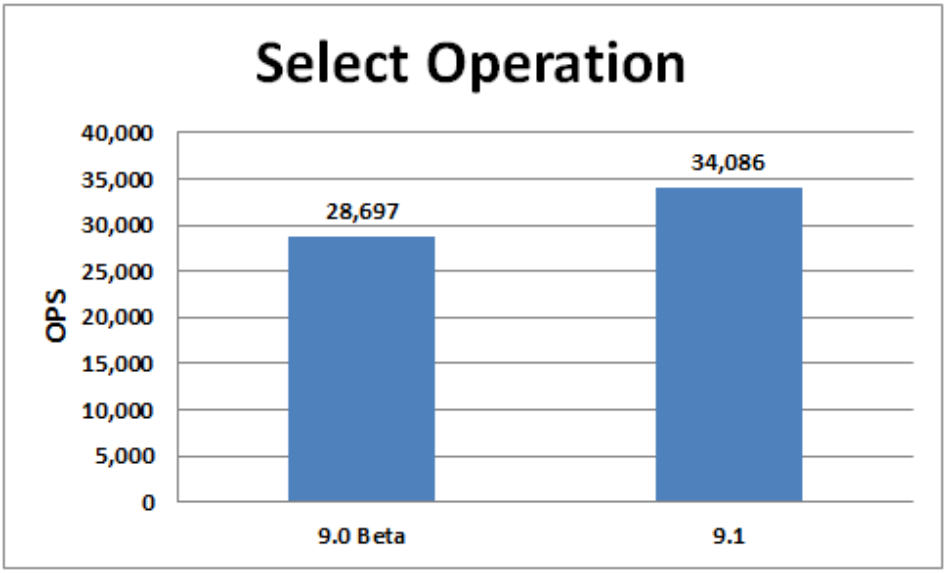


Figure 9. Result of Select Operation of YCSB Benchmark (Master Server)

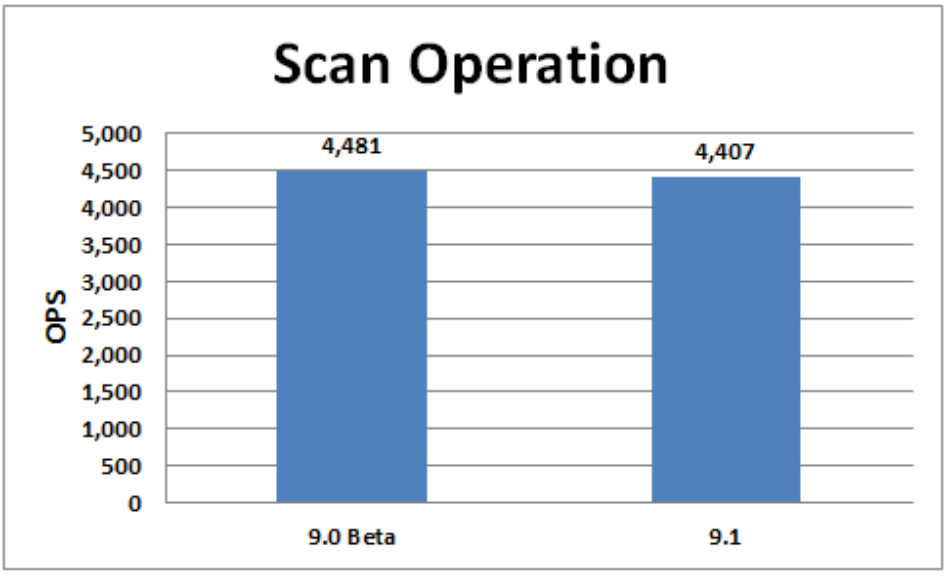


Figure 10. Result of Scan Operation of YCSB Benchmark (Master Server)

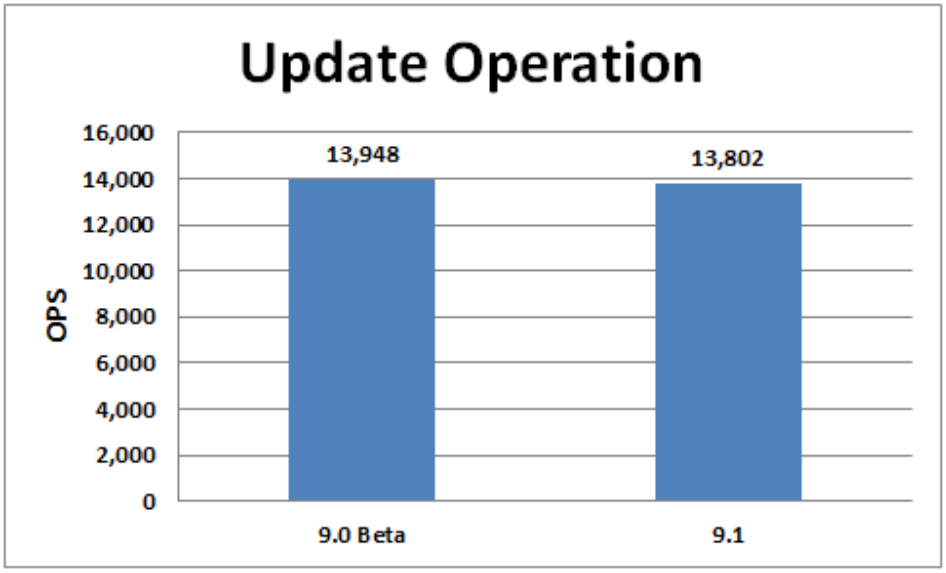


Figure 11. Result of Update Operation of YCSB Benchmark (Master Server)

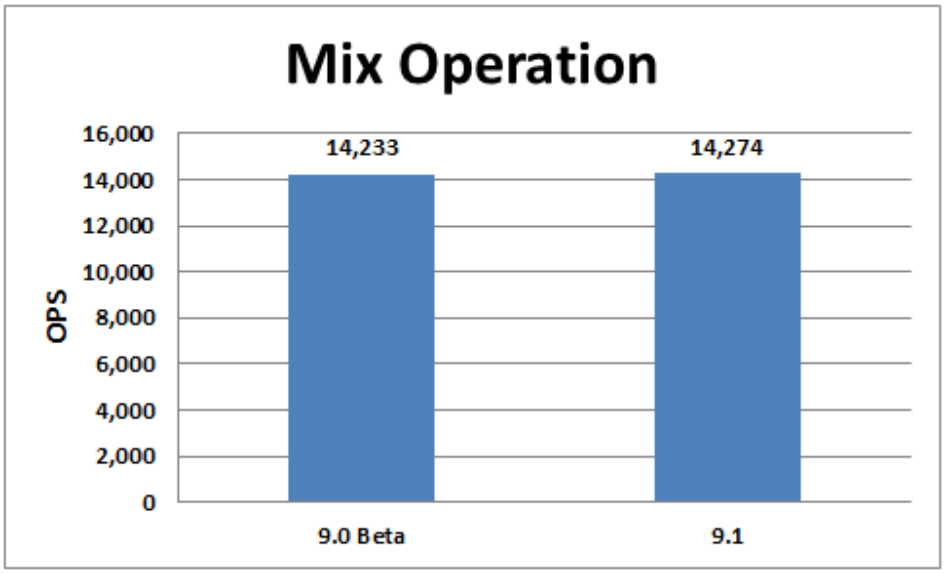


Figure 12. Result of Mix Operation of YCSB Benchmark (Master Server)

B. Slave Server Configuration: Performance Comparison between 9.0 Beta (64-bit) and 9.1 (64-bit)

Table 13. Result of YCSB Benchmark (Slave Server)

	Throughput(OPS)			Average Latency(ms)		95thPercentile Latency(ms)	
Operations	**9.0 Beta**	**9.1**	**Ratio**	**9.0 Beta**	**9.1**	**9.0 Beta**	**9.1**
Insert	16,351	16,459	101%	18	18	34	33
Select	25,008	30,924	124%	11	10	28	28
Scan	4,321	4,248	98%	63	63	250	246
Update	14,392	14,488	101%	20	20	14	14
Mix	14,404	14,432	100%	N/A	N/A	N/A	N/A

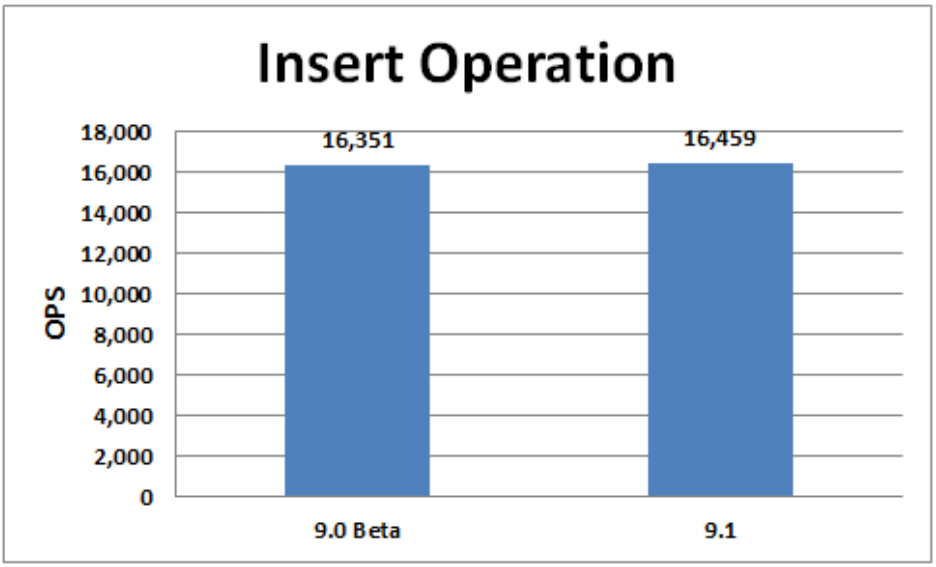


Figure 13. Result of Insert Operation of YCSB Benchmark (Slave Server)

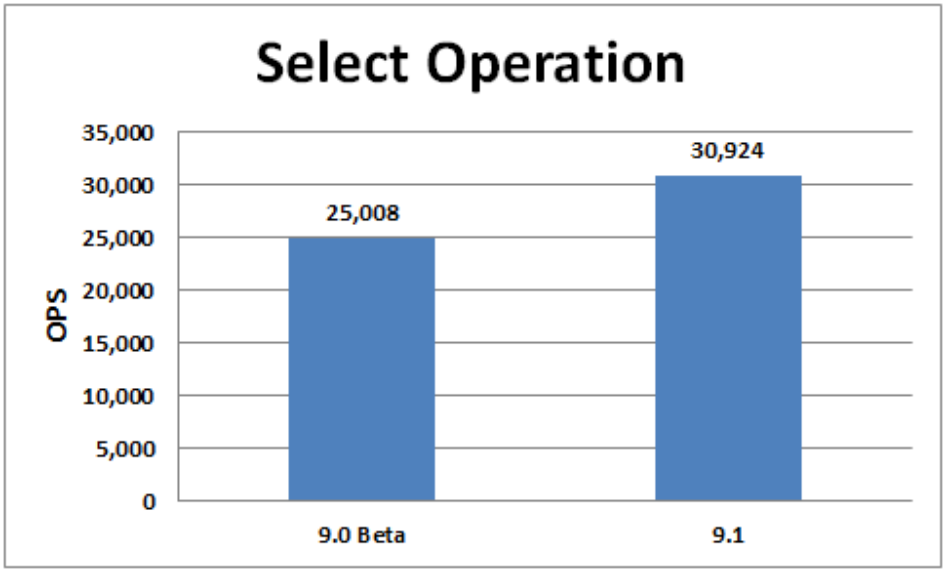


Figure 14. Result of Select Operation of YCSB Benchmark (Slave Server)

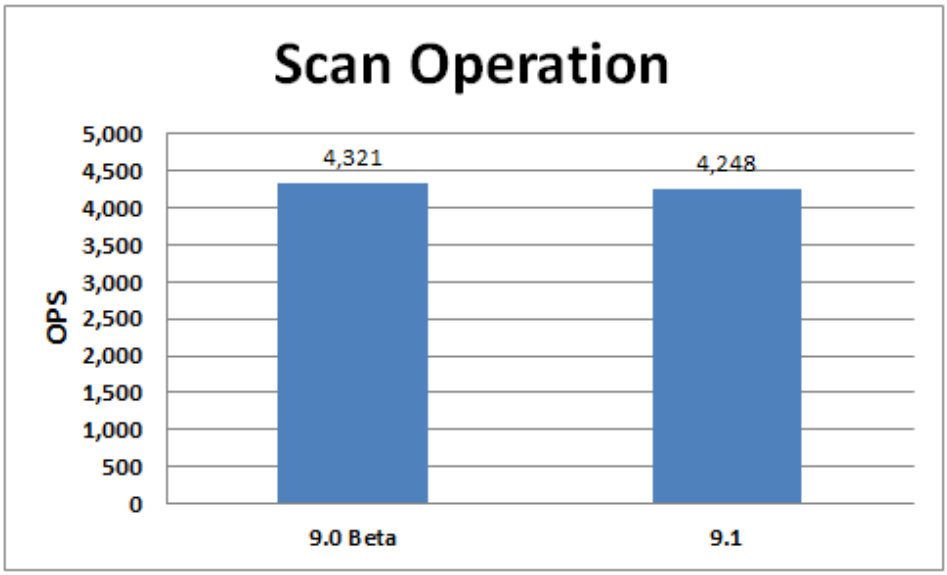


Figure 15. Result of Scan Operation of YCSB Benchmark (Slave Server)

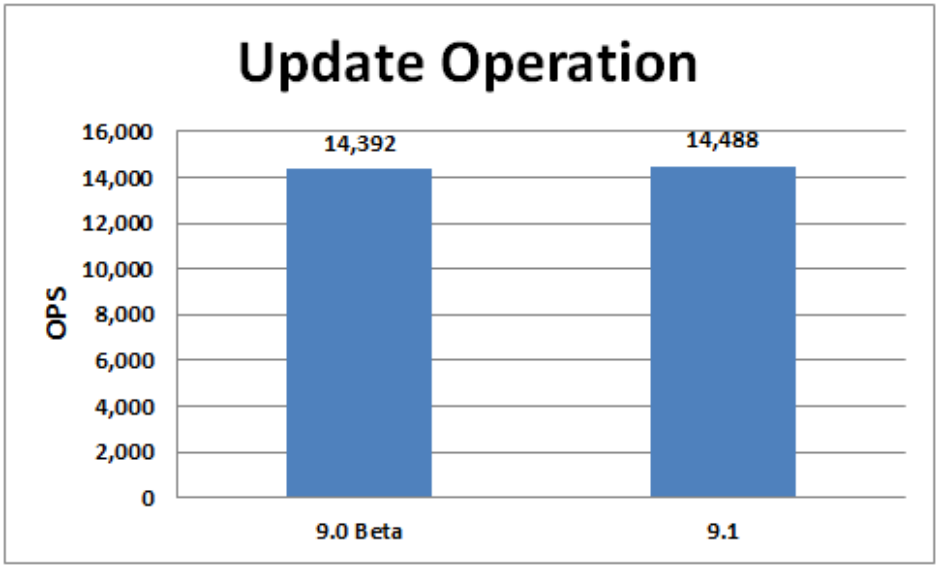


Figure 16. Result of Update Operation of YCSB Benchmark (Slave Server)

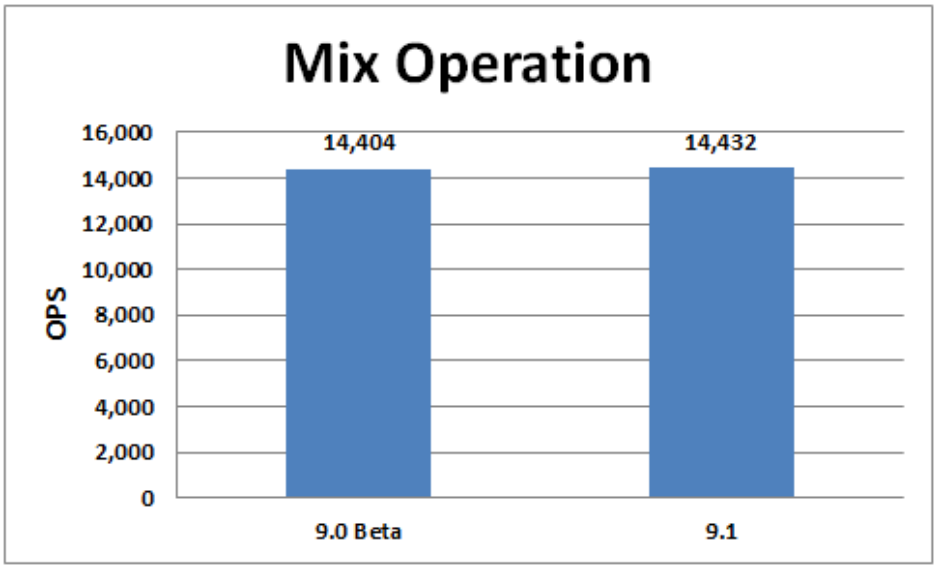


Figure 17. Result of Mix Operation of YCSB Benchmark (Slave Server)

2.2.3 SysBench Performance Test

SysBench is a modular, cross-platform and multi-threaded benchmark tool for evaluating OS parameters that are important for a system running a database under intensive load (see also http://sysbench.sourceforge.net/). SysBench runs a specified number of threads and they all execute requests in parallel. The actual workloads produced by requests depend on the specified test mode. You can limit either the total number of requests or the total time for the benchmark, or both. Available test modes are implemented by compiled-in modules, and SysBench was designed to make adding new test modes an easy task. Each test mode may have additional (or workload-specific) options. For more information about test scenarios, see the appendix II.

As shown in the results below, there is just small difference on the performance of SysBench between 9.0 Beta and 9.1.

A. SysBench performance comparison between 9.0 Beta (64-bit) and 9.1 (64-bit)

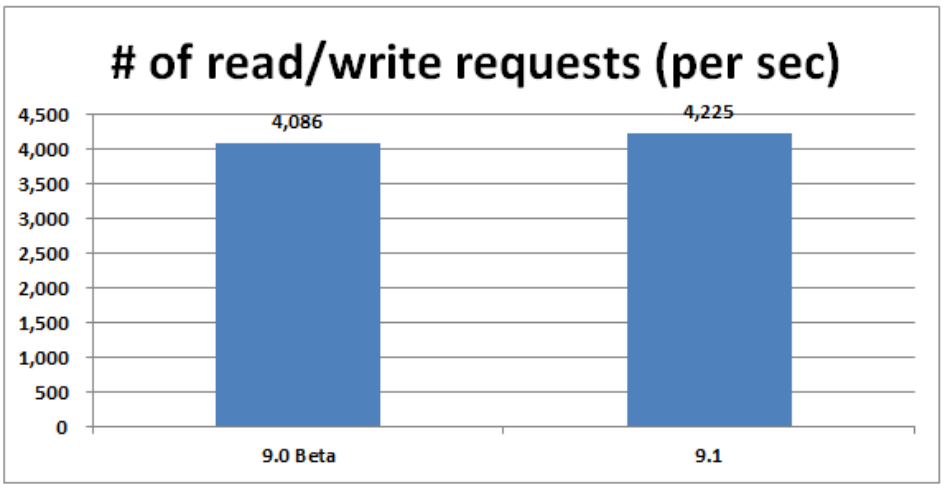


Figure 18. The number of read/write requests per second of SysBench benchmark

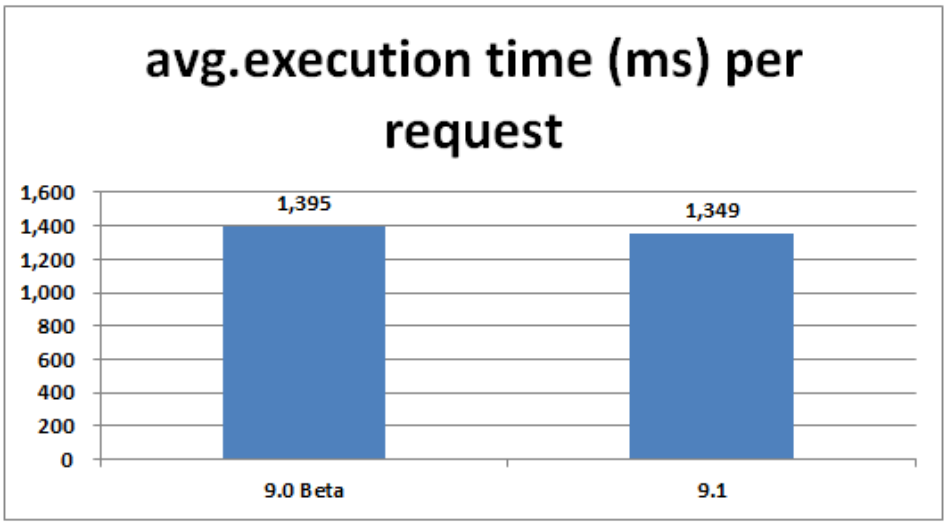


Figure 19. The average execution time per request of SysBench benchmark

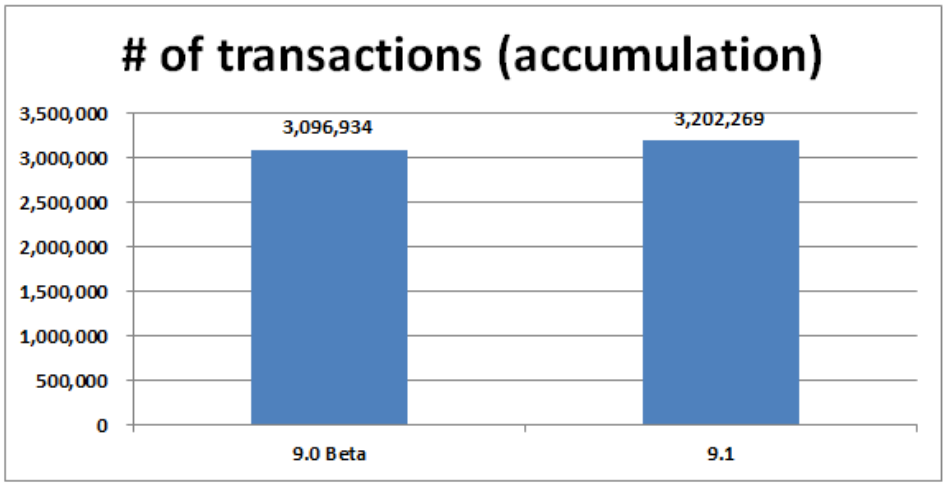


Figure 20. The accumulated number of transactions of SysBench benchmark

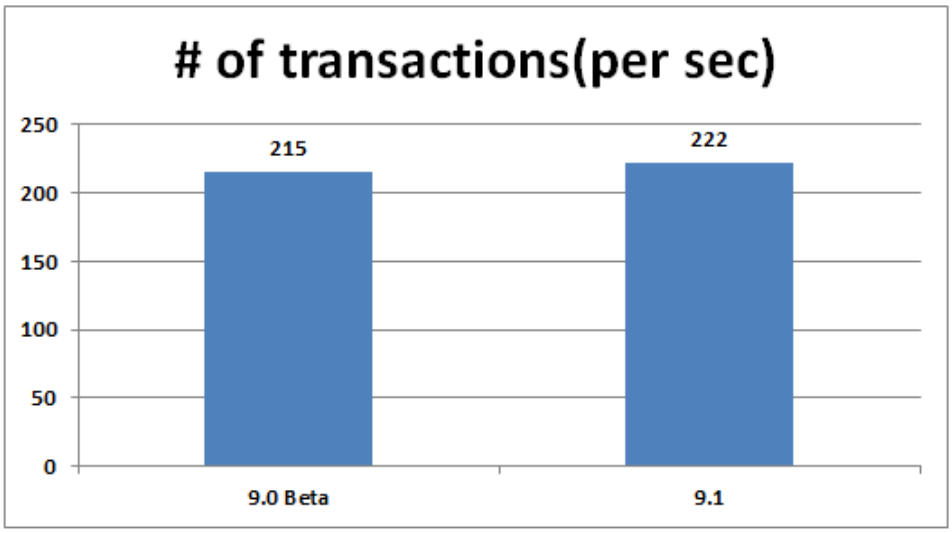


Figure 21. The number of transactions per second of SysBench benchmark

2.2.4 NBD Benchmark Performance Test

This test was performed to verify CUBRID performance with the NBD Benchmark tool, which has been developed to verify the performance of the general bulletin board application framework. The scalability of the test DB was Level 1. The number of Page Views of 9.1 is almost same as that of 9.0 Beta.

A. NBD performance comparison between 9.0 Beta (64-bit) and 9.1 (64-bit)

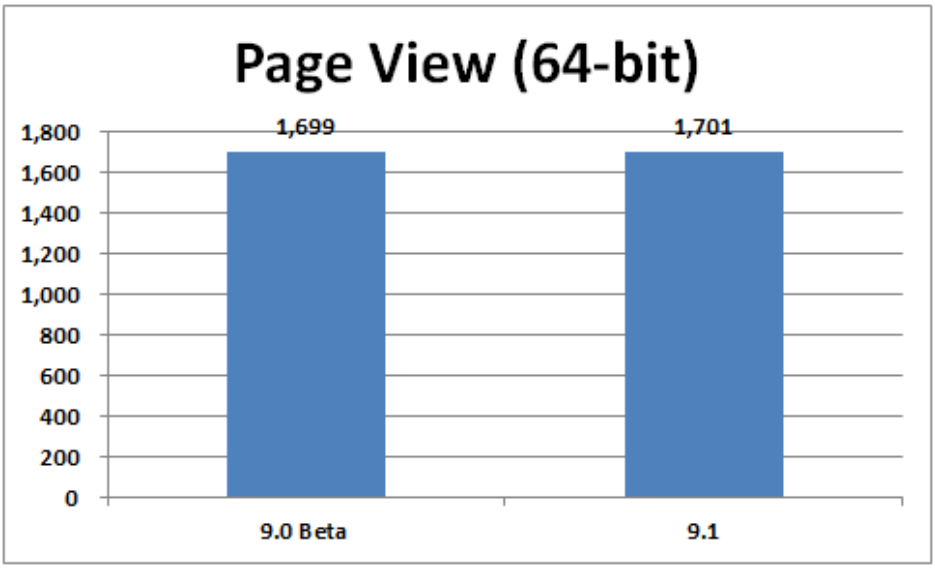


Figure 22. NBD performance comparison (64-bit)

B. NBD performance comparison between 9.0 Beta (32-bit) and 9.1 (32-bit)

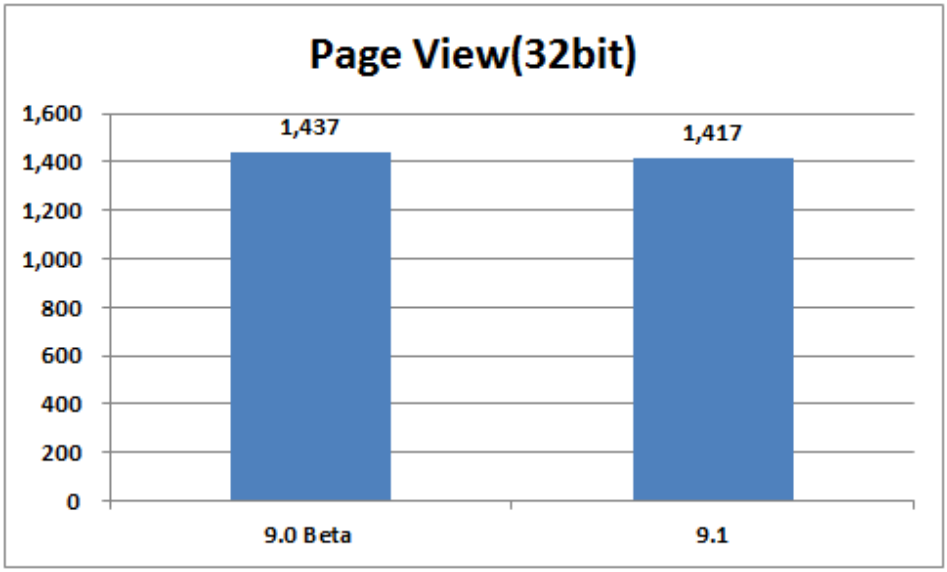


Figure 23. NBD performance comparison (32-bit)

The following graphs represent the usage rate of each resource while processing the NBD benchmark test on Linux 64-bit.

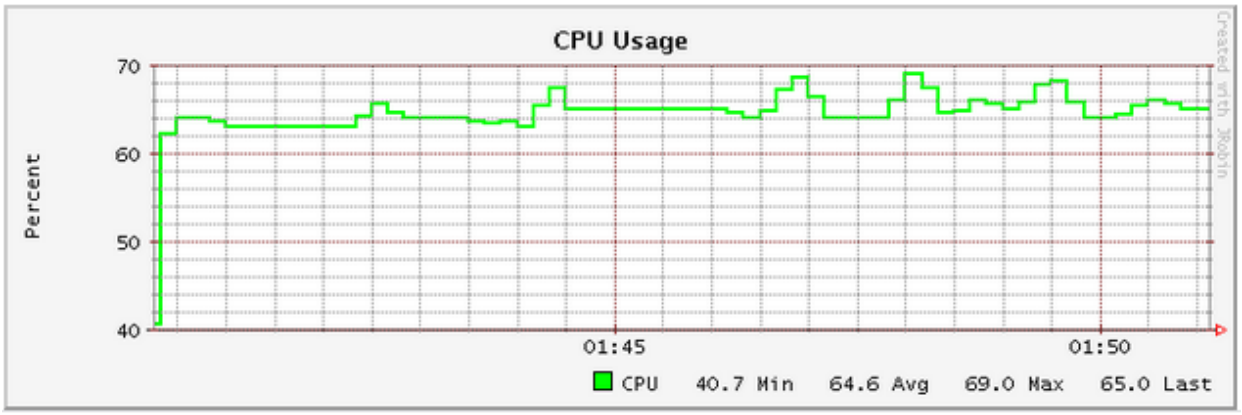


Figure 24. CPU Usage for NBD Benchmark

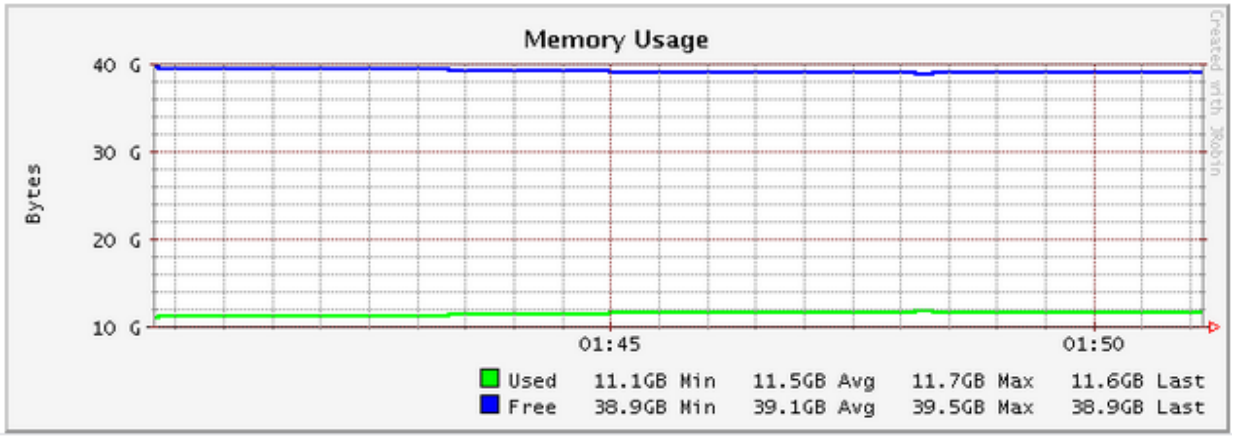


Figure 25. Memory Usage for NBD Benchmark

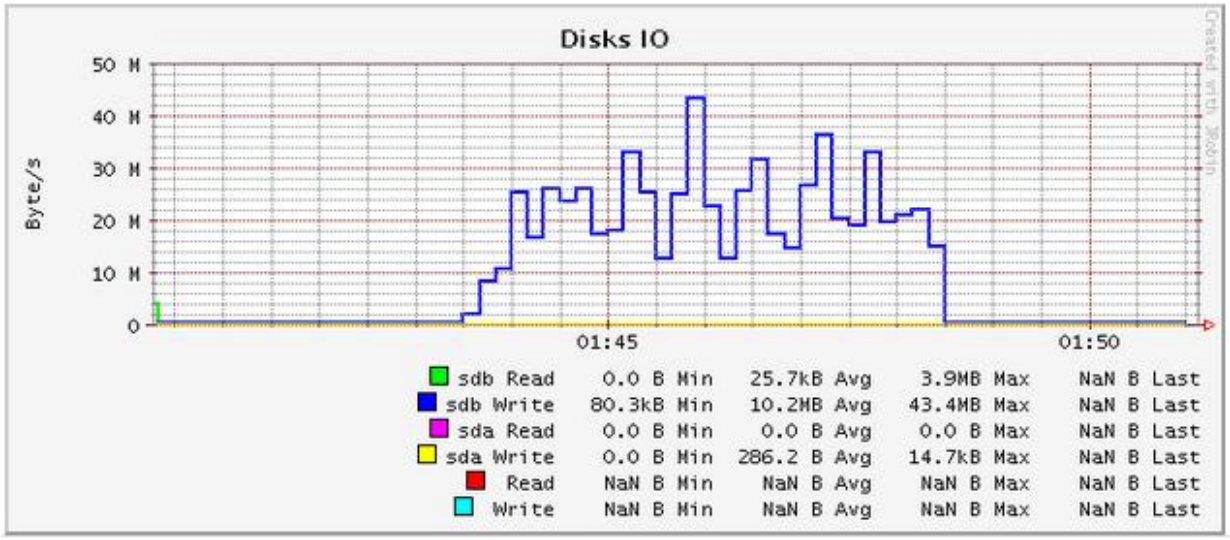


Figure 26. Disks IO status for NBD Benchmark

2.2.5 Data Replication Test on HA

This test was performed to evaluate the performance of data replication under HA environment, by using YCSB to execute Insert, Update and Delete operations on Master server with the related configurations, and check the delay time of data replication on Slave by CUBRID SQL statement. For more details, please refer to appendix II. As shown in the table below, the performance of data synchronization on 9.1 has been significantly improved.

Table 14. Data replication performance comparison

Version	Delay Time (sec.)
9.0 Beta	2238.73
9.1	1.18

2.2.6 TPC-C Performance Test

TPC Benchmark C, approved in July of 1992, is an on-line transaction processing (OLTP) benchmark. TPC-C (see also http://www.tpc.org/tpcc/) is more complex than previous OLTP benchmarks such as TPC-A because of its multiple transaction types, more complex database and overall execution structure. TPC-C involves a mix of five concurrent transactions of different types and complexity either executed on-line or queued for deferred execution. The database is comprised of nine types of tables with a wide range of record and population sizes. TPC-C is measured in transactions per minute (tpmC).
As shown in the results below, the performance of 9.1 on TPC-C is practically same as that of 9.0 Beta.

TPC-C performance comparison between 9.0 Beta (64-bit) and 9.1 (64-bit)

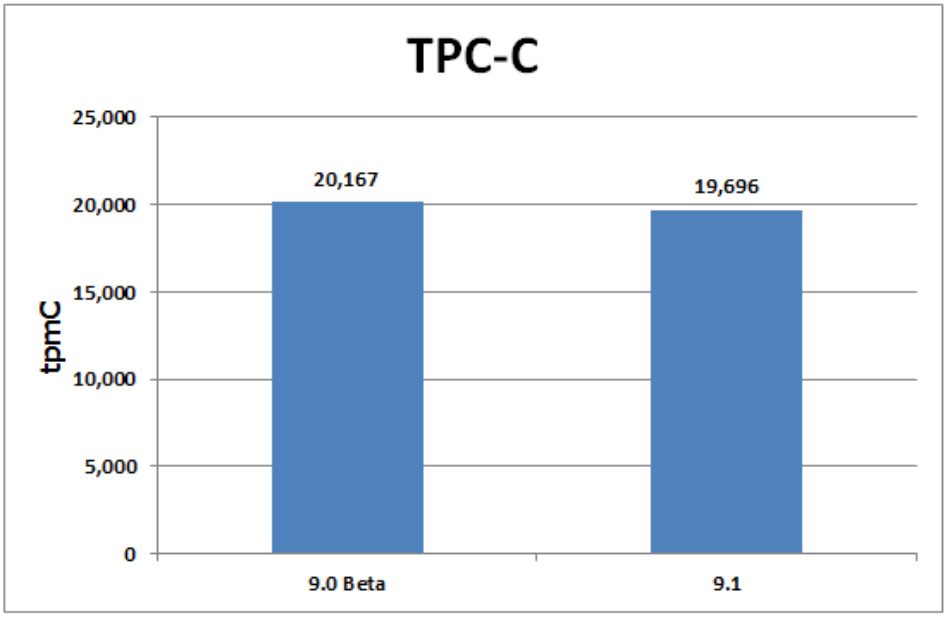


Figure 27. tpmC comparison of TPC-C benchmark

2.3 Stability Test Results

DOTS, a sub-project of an open project called "Linux Test Project", is an open test tool for testing the DBMS. For more information about DOTS, see the appendix III. As shown in the test results below, the system operated stably without any abnormalities during 65 hours. You can ignore the failures because they are unique violations due to the modification of duplicated data.

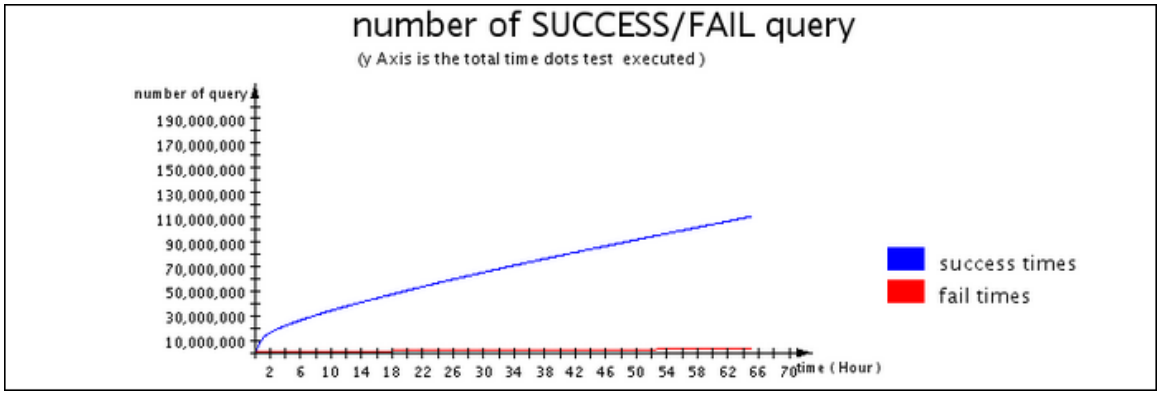


Figure 28. The number of SUCCESS/FAIL Queries of DOTS Test

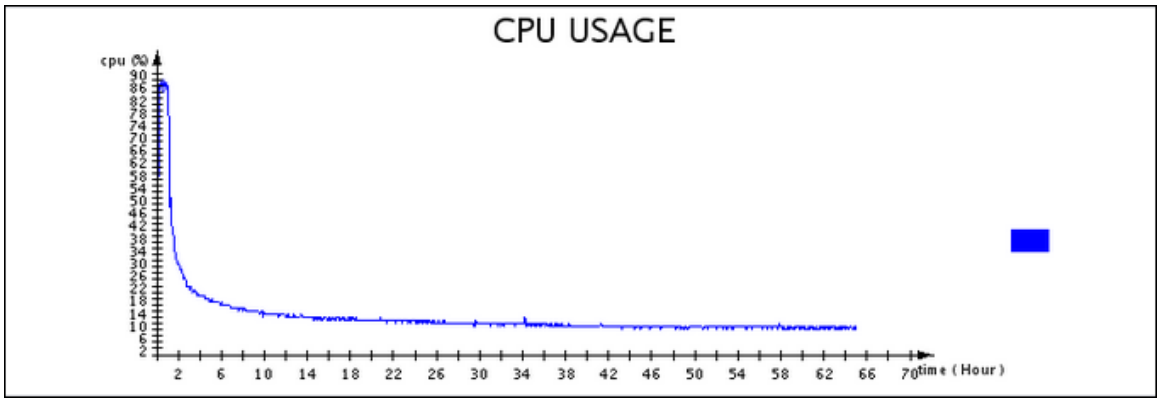


Figure 29. CPU Usage of DOTS Test

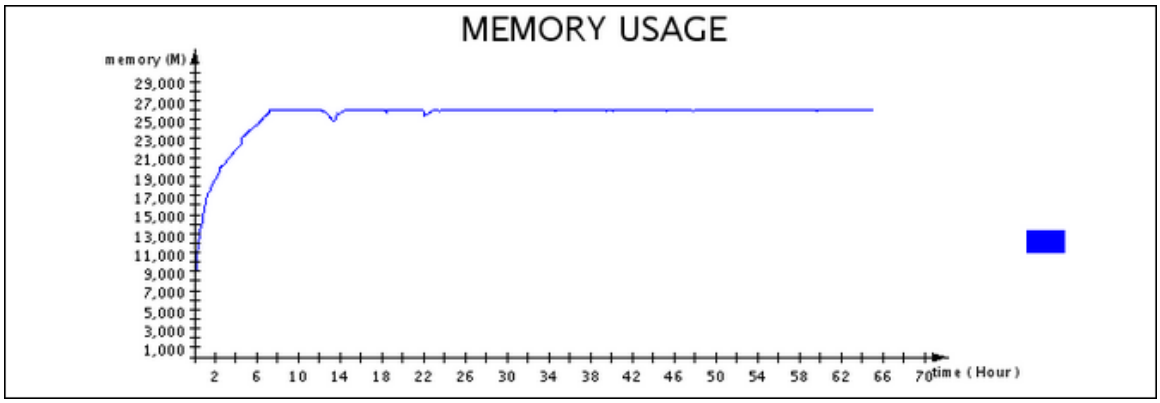


Figure 30. Memory Usage of DOTS Test

2.4 Live Service Simulation Test Result

Live Service Simulation Test is a QA test suite to simulate the real business environment based on CUBRID database, which generates a lot of URLs according to the user id and the keywords users want to search, and the Live Service clients execute queries on CUBRID database by URLs. TPS, CPU usage and Memory usage have also been collected during the test to evaluate the performance and stability of CUBRID.

Table 15. The check points of Live Service Simulation Test

check points	12 hours
	No crash
	No special error messages in server/broker log
	No memory leak
	Continuous service, the total number of requests reaches around 2 billion

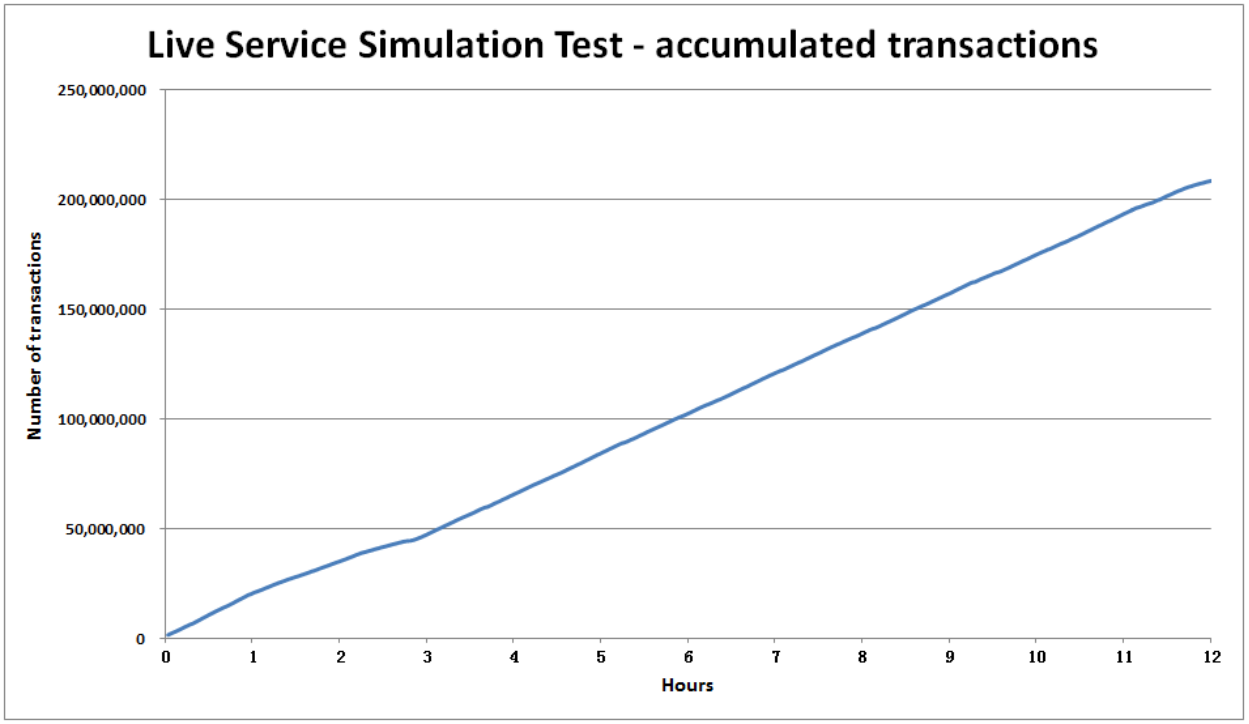


Figure 31. Transactions of Live Service Simulation Test

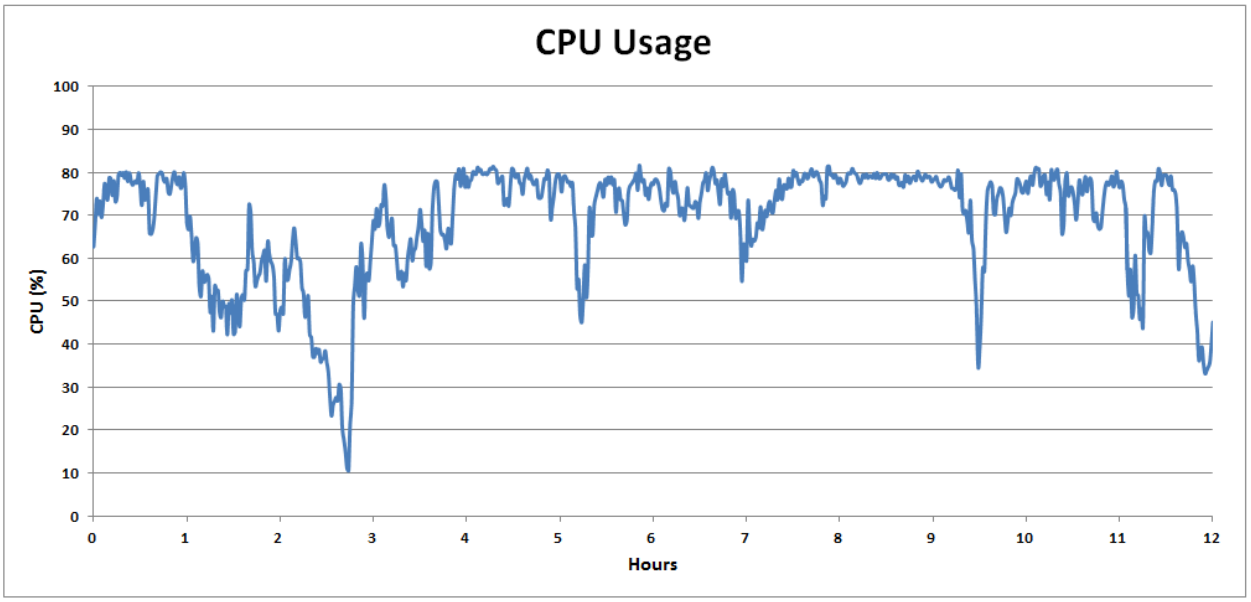


Figure 32. CPU Usage of Live Service Simulation Test

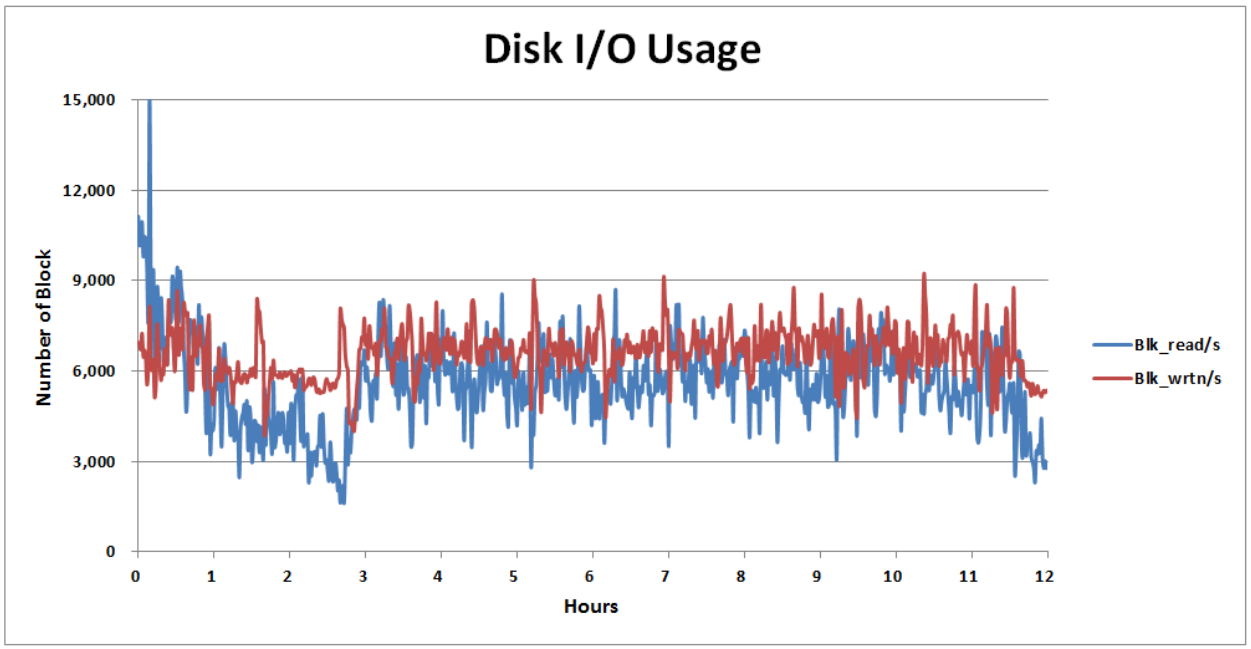


Figure 33. Disk I/O Usage of Live Service Simulation Test

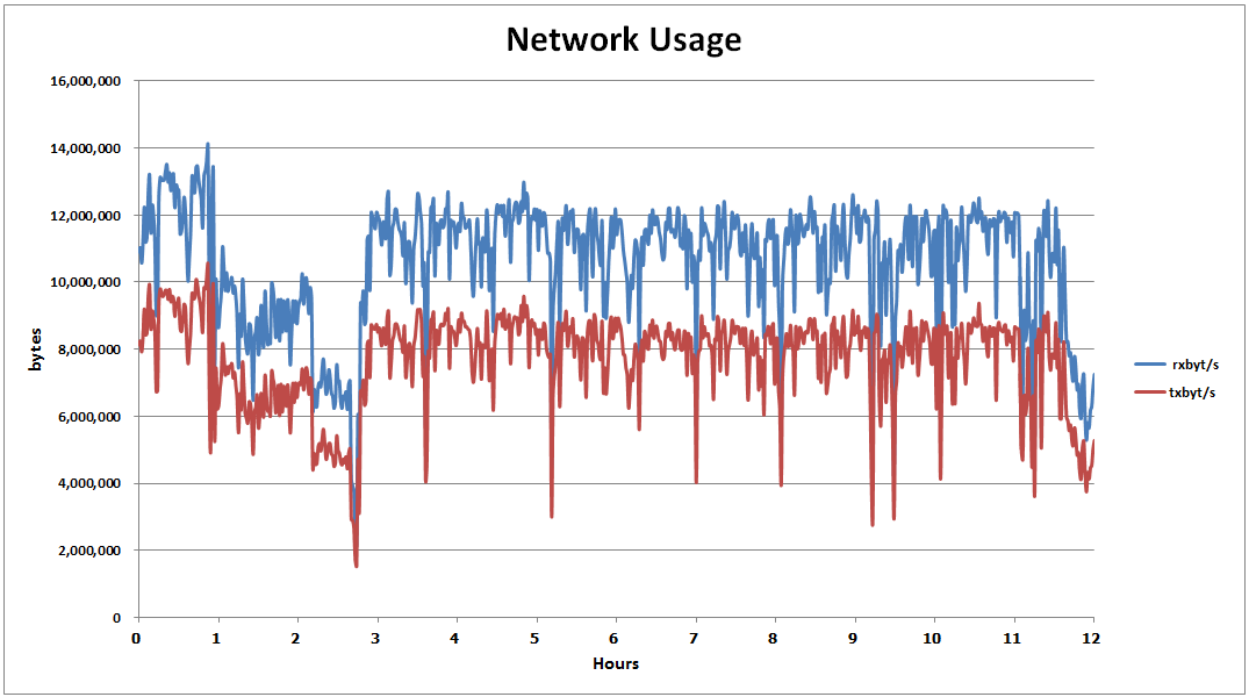


Figure 34. Network Usage of Live Service Simulation Test

2.5 Compatibility Test Results

This test was performed to verify the JDBC and CCI compatibility between R4.1, R4.3, 9.0 Beta and 9.1. SQL, MEDIUM and JDBC Unit Tests were executed to verify JDBC compatibility. Shell test cases for CCI were executed to verify CCI compatibility.

Table 16. Scenario of JDBC Compatibility Tests

Test Category	Scenario(Branch)
9.1 JDBC → 9.0 Beta Server	- SQL/MEDIUM (in 9.0 Beta) - JDBC Test Suite (in 9.1)
9.1 JDBC → R4.3 Server	- SQL/MEDIUM (in R4.3) - JDBC Test Suite (in 9.1)
9.1 JDBC → R4.1 Server	- SQL/MEDIUM (in R4.1) - JDBC Test Suite (in 9.1)
9.0 Beta JDBC → 9.1 Server	- SQL/MEDIUM (in 9.1) - JDBC Test Suite (in 9.0 Beta)
R4.3 JDBC → 9.1 Server	- SQL/MEDIUM (in 9.1) - JDBC Test Suite (in R4.3)
R4.1 JDBC → 9.1 Server	- SQL/MEDIUM (in 9.1) - JDBC Test Suite (in R4.1)

Table 17. Result of JDBC Compatibility Tests

Test Category	# of Scenario Files	# of Scenario Files passed	Pass Rate
9.1 JDBC → 9.0 Beta Server	14,142	14,142	100%
9.1 JDBC → R4.3 Server	11,289	11,289	100%
9.1 JDBC → R4.1 Server	11,264	11,264	100%
9.0 Beta JDBC → 9.1 Server	14,159	14,159	100%
R4.3 JDBC → 9.1 Server	14,537	14,537	100%
R4.1 JDBC → 9.1 Server	14,539	14,539	100%

Table 18. Scenario of CCI Compatibility Tests

Test Category	Scenario(Branch)
9.1 CCI → 9.0 Beta Server	- SQL (in 9.0 Beta) - CCI Shell (in 9.0 Beta)
9.1 CCI → R4.3 Server	- CCI Shell (in R4.3)
9.1 CCI → R4.1 Server	- CCI Shell (in R4.1)
9.0 Beta CCI → 9.1 Server	- SQL (in 9.1) - CCI Shell (in 9.1)
R4.3 CCI → 9.1 Server	- CCI Shell (in 9.1)
R4.1 CCI → 9.1 Server	- CCI Shell (in 9.1)

Table 19. Result of CCI Compatibility Tests

Test Category	# of Scenario Files	# of Scenario Files passed	Pass Rate
9.1 CCI → 9.0 Beta Server	11,943	11,943	100%
9.1 CCI → R4.3 Server	204	204	100%
9.1 CCI → R4.1 Server	204	204	100%
9.0 Beta CCI → 9.1 Server	12,362	12,362	100%
R4.3 CCI → 9.1 Server	12,362	12,362	100%
R4.1 CCI → 9.1 Server	204	204	100%

2.6 Installation Test Results

Installation test was performed based on the following basic scenarios:

- Install and uninstall package
- Start and stop service/server/broker and manager
- Create and delete database

- Execute a simple query in csql
- Make locale

Table 20. Result of Installation Test

Package Type	Test OS	Result
RPM/SH/TAR.GZ	Linux CentOS on 32-bit and 64-bit	PASS
SH	Ubuntu 11 on 64-bit SULinux on 64-bit Fedora 15 64-bit	PASS
EXE/ZIP	Windows Server 2008/2003 on 32-bit and 64-bit	PASS
EXE/ZIP	Windows 7 on 32-bit and 64-bit Windows XP on 32-bit	PASS

2.7 Other Test Results

The entire bug and issue fixes for 9.1 have been confirmed.

2.8 Quality Index

The standard quality index of 9.1 is listed below.

Table 21. Quality Index of 9.1

Quality Index Name	Project Quality Standard	Approved Quality Index during Implementation	Measurement Target	
Coding Standards Compliance Rate	100%	100%	Number of coding conventions observed in a project	56
			Number of coding conventions applied to each team	56
Code Review Execution Rate	100%	100%	Number of source code lines for which code review is performed.	1,335,804 LOC
			Total number of source code lines in the changed files	1,335,804 LOC
QA Scenario Code Coverage	76%	75.8%	Number of tested statements	213,437
			Total number of statements	281,505
Fault Density Detected by Static Analysis	4 /KLOC	4.2 /KLOC	Number of faults detected by static analysis (Level 1)	307
			Number of faults detected by static analysis (Level 2)	11
			Number of faults detected by static analysis (Level 3)	697
			Number of faults detected by static analysis (Level 4)	0
			Total number of source code lines	938,642 LOC
Cyclomatic Code Complexity	3.3%	3.03%	Number of modules whose complexity is over 30	675
			Total number of modules in a project	22,265
	12%	16.05%	Number of modules whose complexity is over 10	3,574
			Total number of modules in a project	22,265

3.Conclusions

As described in Chapter 1 and 2, all the test cases for functions have been regressed and the scenarios for performance, stability, compatibility, installation and other tests have also been successfully executed before the release of 9.1. The tests have been performed on Linux 32-bit, Linux 64-bit, Windows 32-bit and Windows 64-bit environments. The related defects have been logged into the issue tracker.

Based on the results obtained from the basic performance test, we have found that the performance of INSERT, DELETE, SELECT and UPDATE are almost same as that of 9.0 Beta.

For YCSB, the performance for SELECT operation has improved nearly 20%, and the performance of the other operations has not shown significant changes. Meanwhile, according to the results of SysBench and TPC-C tests, the performance results of 9.1 are almost same as that of 9.0 Beta.

For stability test with DOTS, according to the resource usage and logs within CUBRID, there are no notable issues.

According to the result of data replication test on HA mode, the performance of data synchronization has significantly improved from the previous versions.

From the result of compatibility test, we can reach the conclusion that JDBC and CCI of 9.1 is compatible with 9.0 Beta server and R4.3 server. JDBC and CCI of 9.0 Beta and R4.3 are also compatible with 9.1 server except some known issues.

As a conclusion, CUBRID 9.1 is a very stable version and it meets the criteria of release.

Appendix

I. Functionality Test Scenarios

This test was performed to verify the basic DBMS functionalities using SQL statements. SQL statements stored in files have been executed to verify DBMS conformity. We executed the stored SQL statements in a JDBC-based application, and compared the results to the stored reference file for verification. The scenario files included in the basic functionality test are stored in the SQL and MEDIUM directories of the CUBRID QA tool.

- SQL Query Test

Total: 12,091		
Case Name	**Path**	**Description**
object	sql/_01_object	Performs functionality tests of objects supported by CUBRID, and has the largest number of scenarios (3,332 scenarios).
user_authorization	sql/_02_user_authorization	Performs functionality tests of user and authorization management.
object_oriented	sql/_03_object_oriented	Performs tests for the object-oriented concept. CUBRID is an object-relational database management system (DBMS).
operator_function	sql/_04_operator_function	Performs functionality tests of basic functions and operators supported by CUBRID.
manipulation	sql/_06_manipulation	Performs tests of the insert, update, delete, and select statements, which are the most commonly used SQL statements in DML. Basic statements, subqueries and various join queries are tested.
misc	sql/_07_misc	Performs functionality tests of DCL (Data Control Language), including statistics update or other functionalities.
javasp	sql/_08_javasp	Performs functionality tests of Java stored procedures.
64-bit	sql/_09_64bit	Performs basic functionality test scenarios of the bigint and datetime types
Connect_by	sql/_10_connect_by	Performs a test of the hierarchical query feature
Code coverage	sql/_11_codecoverage	Performs a test of uncovered codes based on the code coverage results.
Syntax Extension	sql/_12_mysql_compatibility	Performs a test of the syntax extension.
BTS issues	sql/_13_issues	Performs a test of known issues, which comes from issue management system.
MySQL compatibility	sql/ _14_mysql_compatibility_2	Performs a unit test of the syntax extension 2.
FBO	sql/ _15_fbo	Performs a test of the FBO feature.
Index enhancement	sql/ _16_index_enhancement	Performs a unit test of the index enhancement.
SQL Extension	sql/ _17_sql_extension2	Performs a test of the syntax extension 2. Includes a test of syntax enhancements, system parameters, show statements, date/time functions, string functions, aggregate functions, other functions.

Index enhancement	sql/ _18_index_enhancement_qa	Performs a test of the index enhancement. Includes a test of limit optimizing, using index clause enhancement, descending index scan, covering index, ordering index, optimizing group by clause, Index scan with like predicate, next key locking, etc.
MySQL compatibility for NEWS service	sql/_22_news_service_mysql_compatibility	Performs a test of several functions, regular expression and hint rewriting.
SQL Extension 3 Index Enhancement Internationalization (CUBRID 9.0 Beta unit test)	sql/_19_apricot	Performs a unit test of syntax extension 3, performance and internationalization features. Includes multi-table UPDATE/DELETE, pseudo column, analytic functions, MERGE statements, ENUM type, filtered index, function based index, index skip scan, partition and collation.
SQL Extension 3 Index Enhancement Internationalization (CUBRID 9.0 Beta QA scenario)	sql/_23_apricot_qa	Performs a test of syntax extension 3, performance and internationalization features, Test of syntax extension 3 includes multi-table UPDATE/DELETE, pseudo column, analytic functions, MERGE statements, ENUM type, and other functions, Test of performance includes filtered index, function based index, index skip scan and partition enhancement. Test of internationalization includes tests of 11 languages.
SQL Extension 3 Internationalization (CUBRID 9.1 QA scenario)	sql/_24_aprium_qa	Performs a test of syntax extension, internationalization features. Test of syntax extension includes TRUNC, WIDTH_BUCKEY, ROUND, NTILE functions, LEAD analytic function, and direct access to partitions in INSERT/UPDATE statements. Test of internationalization includes collation per table, SHOW COLLATION, COLLATE modifier applied to expressions, etc.

- MEDIUM Query Test

Total: 970		
Case Name	**Path**	**Description**
01_fixed	medium/_01_fixed	Performs regression test scenarios for bug fixes that have been implemented since the initial version.
02_xtests	medium /_02_xtests	Performs test scenarios for functionalities supported by CUBRID, but not by other DBMSs.
03_full_mdb	medium /_03_full_mdb	Performs test scenarios for sequential/index scan queries with an index.
04_full	medium /_04_full	Performs test scenarios that include testing queries for limit values of CUBRID.
05_err_x	medium /_05_err_x	Performs negative test scenarios for functionalities that are supported by CUBRID, but not by other DBMSs.
06_fulltests	medium /_06_fulltests	Performs test scenarios for search queries with OIDs.

07_mc_dep	medium /_07_mc_dep	Includes a query that gives various conditions to a WHERE clause in the SELECT query, and tests whether or not a correct result has been selected.
08_mc_ind	medium/_08_mc_ind	Includes scenarios that test queries performing schema change.

- SITE Query Test

Total: 1,213		
Case Name	**Path**	**Description**
k_count_q	site/k_count_q	Retrieves count (*) results of a query that is included in the kcc_q query.
k_merge_q	site/k_merge_q	Forces to give a hint to the kcc_q queries allowing merge joins.
k_q	site/k_q	Performs tests for OID reference, collection type, and path expression that are part of the object-oriented concept supported by CUBRID with different scalabilities. In addition, it performs functionality tests while increasing the number of join participating tables.
n_q	site/n_q	Performs tests for a complex query in which subqueries, outer/inner joins or group-by queries are combined, and checks whether correct results are retrieved.

- Utility (Shell) Test

  This test was performed to verify the basic DBMS functionalities using shell scripts. In particular, this test was also performed to verify CUBRID utilities that cannot be tested by SQL statements. Scenarios of shell scripts are executed to verify DBMS conformity.

Total: 1,439		
Case Name	**Path**	**Description**
utility	shell/_01_utility	Includes a script that tests the database management commands supported by CUBRID.
sqlx_init	shell/_02_sqlx_init	Includes scenarios that change the configuration of CUBRID DBMS parameters, and checks whether they are working correctly.
itrack	shell/_03_itrack	Includes scenarios that verify there is no regression by checking the bug fixes in CUBRID, and stores scenarios that cannot be tested by SQL.
misc	shell/_04_misc	Includes miscellaneous scenarios, such as index, query cache test, jdbc c ache and async_commit.
addition	Shell/_05_addition	Includes scenarios added to improve code coverage and mainly tests the options of CUBRID utilities.
BTS issues	shell/_06_issues	Includes scenarios that verify there is no regression by checking the bug fixes in CUBRID, and stores scenarios that cannot be tested by SQL.
Index enhancement	shell/_07_index_enhance ment	Includes scenarios that verify next key lock and change the configuration of CUBRID DBMS related to index enhancement, which has been added in CUBRID 2008 R4.0 Beta.
64bit scenario	shell/_09_64bit	Includes file size on linux 64 bit
xa datasource	shell/_21_xa	Includes scenarios to cover xa DataSource features
MySQL service compatibility	shell/_22_news_service_ mysql_compatibility	Includes scenarios of CUBRID compatibility with MySQL service

MySQL compatibility	shell/_23_mysql_compatibility	Includes scenarios that verify syntax extension, which has been added in CUBRID 2008 R3.1.
CUBRID 9.0 Beta QA	shell/_24_apricot	Includes scenarios that verify CUBRID 9.0 Beta functions such as i18n, enum, etc.
unstable	shell/_25_ unstable	Includes scenarios that are not very stable
CUBRID 9.0 Beta QA	shell/_26_apricot_qa	Includes scenarios that added by QA to verify CUBRID 9.0 Beta functions such as i18n, cursor holdability, etc.
CUBRID 9.1 QA	shell/_27_aprium_qa	Includes scenarios that added by QA to verify prefix key, enum, collation of CUBRID 9.1 i18n function.
manual shell	Manually/*	All manual test cases which can't be automated or need long time to regress

- HA Feature Test

Total: 293		
Case Name	**Path**	**Description**
Data replication test	execp/UsualCase	Includes scenarios that check whether HA replication is properly performed in a normal state with no fault.
Node fault test	execp/UsualCase	Includes scenarios that check whether HA replication is properly performed when a node fault occurs during insert/update/delete operations.
Process fault test	execp/UsualCase	Includes scenarios that check whether HA replication is properly performed when a process fault occurs that causes the database process to stop during insert/update/delete operations.
Broker fault test	execp/UsualCase	Includes scenarios that check whether HA replication is properly performed when a broker fault occurs during insert/update/delete operations.
Replication scenario	scripts/sql	Includes scenarios that test whether HA is working properly for each CUBRID transaction type, and has two sub directories: random_case and special_case
Bug regression	HA/shell/	Includes scenarios that verify there is no regression by checking the HA bug fixes in CUBRID

- HA Replication test

Total: 12,182		
Case Name	**Path**	**Description**
Test Cases migrated from SQL suite	N/A	Migrated existing SQL suite into HA environment. Execute them on master node, then check whether be replicated to slave or not.
Bug Regression	HA/shell/_24_functional_repl/	Includes scenarios that verify there is no regression by checking the HA bug fixes in CUBRID

- CCI Interface test

Total: 250		
Case Name	**Path**	**Description**
Features test	Interface/shell/_20_cci	Which contains CCI all APIs, each APIs are mentioned in manual are tested in shell scripts
Bug Regression	Interface/shell/_20_cci/_12_issue	Includes shell scripts which are written when verify CCI bts issues

- JDBC Interface test

Total: 1,529		
Case Name	**Path**	**Description**
Features test	N/A	Which include unit test for jdbc, jdbc spec 3.0 test, and other open source databases jdbc case migration

- CAS4MySQL/Oracle test

Total: 64		
Case Name	**Path**	**Description**
CAS4MySQL	N/A	Cas4MySQL test and CAS4MySQL bts issues automation scripts
CAS4Oracle	N/A	Cas4Oracle test and Cas4Oracle bts issues automation scripts

II. Performance Test Scenarios

■ CUBRID Basic Performance Test

To evaluate the basic performance of DBMS, the following 5 variables were used. Database Server, Broker, and Load Generator were run on a single server.

■ Number of data (or number of program loops)

✧ Total number of data: 900,000 items

✧ Number of program loops: 100,000 loops/program (900,000 items)

- COMMIT Interval
  - After every execution
  - After 100 executions
  - After 1,000 executions
- Number of concurrent users
  - 5 users
  - 10 users
- Number of index attributes
  - create index idx1 on xoo(a)
  - create index idx2 on xoo(a,b)
  - create index idx3 on xoo(a,b,e)
- Interface
  - JDBC (Dynamic SQL): Prepared statements were used.

■ Test data

✧ Test schema

```
CREATE TABLE xoo (
        a       int,
        b       int,
        c       int,
        d       int,
        e       char(10),
        f       char(20),
        g       char(30)
)
```

```
CREATE INDEX idx1 on xoo(a);
CREATE INDEX idx2 on xoo(a,b);
CREATE INDEX idx3 on xoo(a,b,e);
```

✧ Test data

Enter data from 1 to 450,000; total number of data is 900,000.

✧ How to perform a test

- Insert/update/select/delete data from a specific number.
- For concurrent user tests, the start and end numbers are defined to prevent data from overlapping, in order to ensure that there is no competition between the concurrent clients.
- For concurrent user test programs, a JDBC test program is tested with a multi-threaded program, and a C program is tested with a multi-process program.
- If the number of loops is 10,000, a user repeats execution 10,000 times in the case of the 1-user test, and each user repeats execution 2,000 times in the case of the 5-user test. Similarly, if the number of loops is 100,000, a user repeats execution 100,000 times in the case of the 1-user test, and each user repeats execution 20,000 times in the case of the 5-user test.

✧ How to measure test results

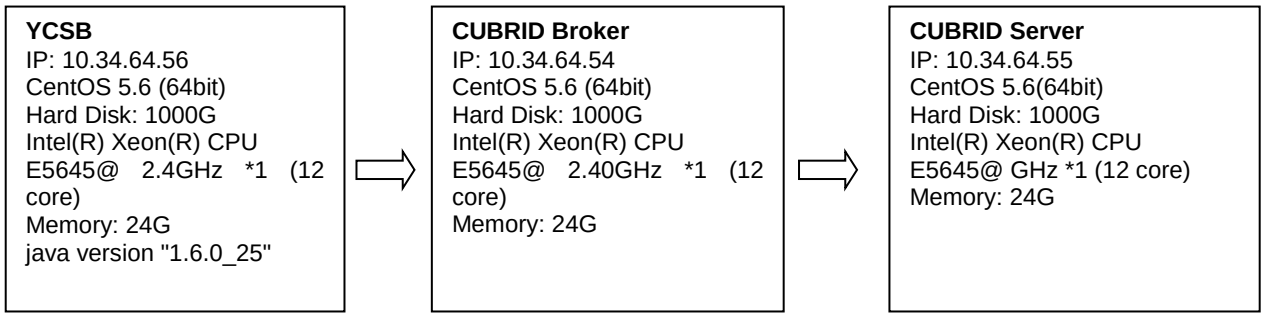
- Measure the number of loops per second.
- For concurrent user tests, add the execution times of all users.

■ YCSB Benchmark

This test was performed to verify CUBRID performance of not only basic operations but also compositive operations, which are insert, select, scan, update and mix of them.

■ Common Test Environment

✧ Test Servers



✧ CUBRID database volume configuration

```
cubrid createdb ycsb
cubrid addvoldb -p data --db-volume-size=2G ycsb -S
cubrid addvoldb -p data --db-volume-size=2G ycsb -S
```

```
cubrid  addvoldb -p index --db-volume-size=2G ycsb -S
cubrid  addvoldb -p index --db-volume-size=2G ycsb -S
cubrid  addvoldb -p temp --db-volume-size=2G ycsb –S
```

- ✧ Configuration for CUBRID
  - ⬩ cubrid_broker.conf:

```
SERVICE                 =ON
BROKER_PORT               =33000
MIN_NUM_APPL_SERVER       =5
MAX_NUM_APPL_SERVER       =300
APPL_SERVER_SHM_ID        =33000
LOG_DIR                 =log/broker/sql_log
ERROR_LOG_DIR             =log/broker/error_log
SQL_LOG                 =OFF
TIME_TO_KILL              =120
SESSION_TIMEOUT           =300
KEEP_CONNECTION           =AUTO
CCI_DEFAULT_AUTOCOMMIT    =ON
```

  - ⬩ cubrid.conf:

```
data_buffer_size=4G
sort_buffer_size=2M
cubrid_port_id=1523
max_clients=500
db_volume_size=512M
log_volume_size=512M
```

- ✧ Workload configuration on YCSB
  - ⬩ Insert operation (load)

```
recordcount=10000000
operationcount=10000000
workload=com.yahoo.ycsb.workloads.CoreWorkload
readallfields=true
readproportion=0
updateproportion=0
scanproportion=0
insertproportion=1
requestdistribution=zipfian
threads=300
fieldlength=10
```

  - ⬩ Select operation

```
recordcount=10000000
operationcount=10000000
workload=com.yahoo.ycsb.workloads.CoreWorkload
readallfields=true
readproportion=1
```

```
updateproportion=0
scanproportion=0
insertproportion=0
requestdistribution=zipfian
threads=300
fieldlength=10
table=usertable
```

- Scan operation

```
recordcount=10000000
operationcount=10000000
workload=com.yahoo.ycsb.workloads.CoreWorkload
readallfields=true
readproportion=0
updateproportion=0
scanproportion=1
insertproportion=0
requestdistribution=zipfian
fieldlength=10
table=usertable
maxscanlength=200
threads=300
```

- Update operation

```
recordcount=10000000
operationcount=10000000
workload=com.yahoo.ycsb.workloads.CoreWorkload
readallfields=true
readproportion=0
updateproportion=1
scanproportion=0
insertproportion=0
requestdistribution=zipfian
fieldlength=10
table=usertable
threads=300
```

- Mix operation

```
recordcount=10000000
operationcount=10000000
workload=com.yahoo.ycsb.workloads.CoreWorkload
readallfields=true
readproportion=0.3
updateproportion=0.3
scanproportion=0.1
insertproportion=0.3
requestdistribution=zipfian
fieldlength=10
table=usertable
maxscanlength=200
```

```
threads=300
```

✧ Test schema

```
Create table usertable (
userkey                          CHARACTER VARYING(100) PRIMARY KEY,
field1              CHARACTER VARYING(100),
field2              CHARACTER VARYING(100),
field3              CHARACTER VARYING(100),
field4              CHARACTER VARYING(100),
field5              CHARACTER VARYING(100),
field6              CHARACTER VARYING(100),
field7              CHARACTER VARYING(100),
field8              CHARACTER VARYING(100),
field9              CHARACTER VARYING(100),
field10             CHARACTER VARYING(100)
)
```

■ Test data on master server configuration

✧ CUBRID server configuration

- async_commit=no
- group_commit_interval_in_msecs=0

■ Test data on slave server configuration

✧ CUBRID server configuration

- async_commit=yes
- group_commit_interval_in_msecs=1000

■ Statements to be tested

✧ Insert operation

```
INSERT INTO usertable(userkey, field1, field2, field3, field4, field5, field6, field7, field8, field9, field10)
VALUES (?, ?, ?, ?, ?, ?,?, ?, ?, ?, ?);
```

✧ Select operation

```
SELECT * FROM usertable WHERE userkey= ?;
```

✧ Scan operation

```
SELECT * FROM usertable WHERE userkey>= ?LIMIT ?;
```

✧ Update operation

```
UPDATE usertable set field1=?, field2=?, field3=?, field4=?, field5=?, field6=?, field7=?, field8=?, field9=?, field10=? WHERE
```

```
userkey = ?;
```

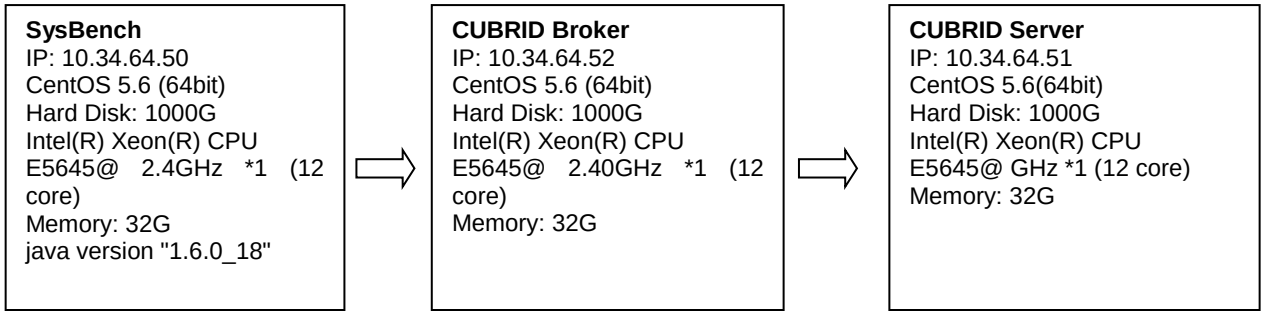
- ✧ Mix operation
  - Select operation: 30%
  - Update operation: 30%
  - Scan operation: 10%
  - Insert operation: 30%

■ SysBench Benchmark

This test was performed to verify CUBRID performance based on OLTP business.

■ Test Environment

✧ Test Servers



✧ CUBRID database volume configuration

```
cubrid createdb sysbench
cubrid addvoldb -p data --db-volume-size=2G sysbench -S
cubrid addvoldb -p data --db-volume-size=2G sysbench -S
cubrid addvoldb -p index --db-volume-size=2G sysbench -S
cubrid addvoldb -p temp --db-volume-size=2G sysbench -S
```

✧ Configuration for CUBRID

- cubrid_broker.conf:

```
SERVICE                 =ON
BROKER_PORT              =33000
MIN_NUM_APPL_SERVER      =350
MAX_NUM_APPL_SERVER      =350
APPL_SERVER_SHM_ID       =33000
LOG_DIR                 =log/broker/sql_log
ERROR_LOG_DIR            =log/broker/error_log
SQL_LOG                 =OFF
TIME_TO_KILL            =120
SESSION_TIMEOUT          =300
KEEP_CONNECTION          =AUTO
```

```
CCI_DEFAULT_AUTOCOMMIT   =ON
```

- cubrid.conf:

```
data_buffer_size=4G
log_buffer_size=4M
sort_buffer_size=2M
max_clients=500
cubrid_port_id=1523
db_volume_size=512M
log_volume_size=512M
async_commit=no
group_commit_interval_in_msecs=0
```

✧ Test schema

```
create table sbtest(
      id        INTEGER AUTO_INCREMENT PRIMARY KEY,
      k         INTEGER DEFAULT 0 NOT NULL,
      c         CHAR(120) NOT NULL DEFAULT '',
      pad       CHAR(60) NOT NULL DEFAULT '',
)
```

✧ Configuration to start SysBench

```
./sysbench --test=oltp ☐
           --db-driver=cubrid ☐
           --cubrid-host=10.34.64.52 ☐
           --cubrid-port=33000 ☐
           --cubrid-db=sysbench ☐
           --num-threads=300 ☐
           --max-requests=0 ☐
           --max-time=14400 ☐
           --oltp-skip-trx=off ☐
           --oltp-read-only=off ☐
           --oltp-table-size=1000000 ☐
run
```

■ NBD Benchmark

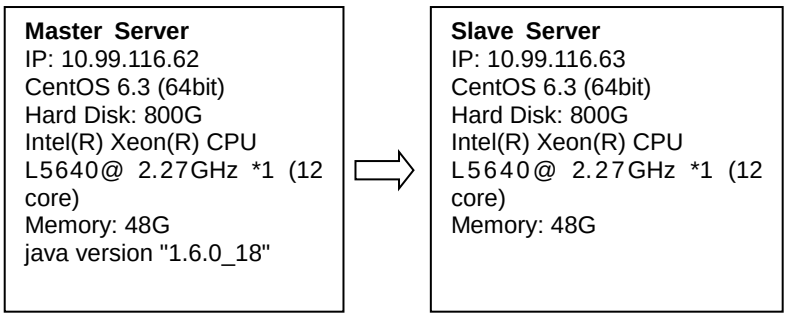
This test was performed to verify CUBRID performance using the NBD Benchmark tool, which has been developed to verify the performance of the general bulletin board application framework. For more information about NBD Benchmark, see separate documents.

■ Data Replication Test on HA

This test was performed to e

valuate the performance of data replication on HA environment, by using YCSB to execute Insert, Update and Delete operati ons on Master server with the related configurations, and check the delay time of data replication on Slave by CUBRID SQL st atement.

✧ Test Servers



✧ Table scheme

```
csql> ;sc usertable
=== <Help: Schema of a Class> ===
 <Class Name>
      usertable
 <Attributes>
      userkey                 CHARACTER VARYING(100) NOT NULL
      field1                  CHARACTER VARYING(100)
      field2                  CHARACTER VARYING(100)
      field3                  CHARACTER VARYING(100)
      field4                  CHARACTER VARYING(100)
      field5                  CHARACTER VARYING(100)
      field6                  CHARACTER VARYING(100)
      field7                  CHARACTER VARYING(100)
      field8                  CHARACTER VARYING(100)
      field9                  CHARACTER VARYING(100)
      field10                 CHARACTER VARYING(100)
 <Constraints>
      PRIMARY KEY pk_usertable_userkey ON usertable (userkey)
```

✧ Configuration for CUBRID

- cubrid_broker.conf:

```
SERVICE                     =ON
BROKER_PORT                   =33000
MIN_NUM_APPL_SERVER         =5
MAX_NUM_APPL_SERVER         =200
APPL_SERVER_SHM_ID          =33000
LOG_DIR                     =log/broker/sql_log
ERROR_LOG_DIR                 =log/broker/error_log
SQL_LOG                     =OFF
TIME_TO_KILL                =120
SESSION_TIMEOUT               =300
KEEP_CONNECTION              =AUTO
CCI_DEFAULT_AUTOCOMMIT    =ON
```

- cubrid.conf:

```
data_buffer_size=5G
max_clients=100
ha_copy_sync_mode=sync:sync
```

✧ YCSB configurations

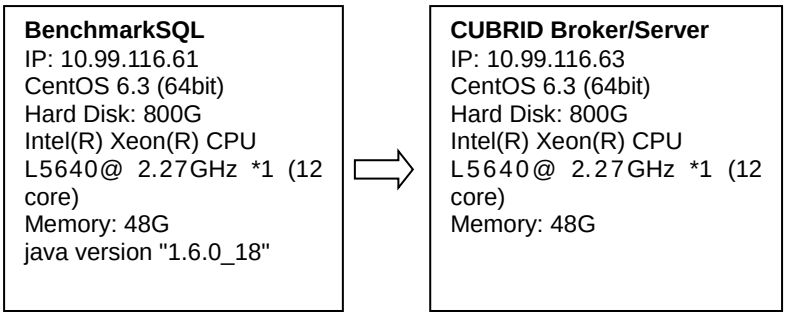
```
recordcount=20000000
operationcount=10000000
insertproportion=0.6/updateproportion=0.3/deleteproportion=0.1
threads=50
```

■ TPC-C Benchmark

BenchmarkSQL is a implementation of TPC-C standard. We can get more information in website http://sourceforge.net/projects/benchmarksql/. For this performance test, we just use this BenchmarkSQL tool to execute on CUBRID. In order to support CUBRID very well, we made some corrections.

■ Test Environment

✧ Test Servers



✧ CUBRID database volume configuration

```
cubrid createdb tpcdb10
cubrid addvoldb -p data --db-volume-size=2G tpcdb10 -S
cubrid addvoldb -p data --db-volume-size=2G tpcdb10- S
cubrid addvoldb -p index --db-volume-size=2G tpcdb10 -S
cubrid addvoldb -p temp --db-volume-size=2G tpcdb10 -S
```

✧ Configuration for CUBRID

• cubrid_broker.conf:

```
SERVICE                 =ON
BROKER_PORT               =33000
MIN_NUM_APPL_SERVER       =5
MAX_NUM_APPL_SERVER       =200
APPL_SERVER_SHM_ID        =33000
LOG_DIR                 =log/broker/sql_log
ERROR_LOG_DIR             =log/broker/error_log
SQL_LOG                 =OFF
TIME_TO_KILL              =120
SESSION_TIMEOUT           =300
KEEP_CONNECTION           =AUTO
CCI_DEFAULT_AUTOCOMMIT    =ON
```

• cubrid.conf:

```
data_buffer_size=4G
max_clients=300
```

✧ BenchmarkSQL configuration

```
Number of warehouses: 10
Number of Terminals: 100
Execute minutes: 30

Payment : 43%, Order-Status: 4%,  Delivery: 4% , Stock-Level: 4% ,New-Order:45%
```

III. Stability Test Scenarios

DOTS, a sub-project of an open project called "Linux Test Project", is an open test tool for testing the DBMS.

- Test Related Schema (the Number of Data in Each Table)

```
CREATE TABLE REGISTRY (
    USERID              CHAR(15) NOT NULL PRIMARY KEY,
    PASSWD              CHAR(10),
    ADDRESS             CHAR(200),
    EMAIL               CHAR(40),
    PHONE               CHAR(15)
);

CREATE TABLE ITEM (
    ITEMID              CHAR(15) NOT NULL PRIMARY KEY,
    SELLERID            CHAR(15) NOT NULL,
    DESCRIPTION         VARCHAR(250) ,
    BID_PRICE           FLOAT,
    START_TIME          DATE,
    END_TIME            DATE,
    BID_COUNT           INTEGER
);

CREATE TABLE BID (
    ITEMID              CHAR(15) NOT NULL PRIMARY KEY,
    BIDERID             CHAR(15) NOT NULL,
    BID_PRICE           FLOAT,
    BID_TIME            DATE
);
```

- CUBRID configuration
  - cubrid_broker.conf

```
MIN_NUM_APPL_SERVER=20
MAX_NUM_APPL_SERVER=100
APPL_SERVER_MAX_SIZE=100
```

  - cubrid.conf

```
log_max_archives=150
async_commit=yes
group_commit_interval_in_msecs=10
checkpoint_every_npages=100000
checkpoint_interval_in_mins=10
max_clients=200
data_buffer_size=1G
```

- DOTs configuration

```
DURATION=24:00
CONCURRENT_CONNECTIONS= 20
```

```
AUTO_MODE = no
SUMMARY_INTERVAL = 5
MAX_ROWS= 900000000
```

- Data Size and How to Create Data

The initial number of data when starting the test is 0. Enter 1000 of data in the REGISTRY table. Next, enter 100 of data in the ITEM table as well as in the bid table. Then, update 100 times.

- Transaction types
  - ✧ INSERT transaction 1

```
INSERT INTO ITEM (ITEMID,SELLERID,DESCRIPTION,BID_PRICE,START_TIME,END_TIME,BID_COUNT)
VALUES (?, ?, ? ,?, ?, ?, ?)
```

  - ✧ INSERT transaction 2

```
INSERT INTO BID (ITEMID,BIDERID,BID_PRICE,BID_TIME)
VALUES (?, ?, ?, ?)
```

  - ✧ SELECT transaction 1

```
SELECT SELLERID,DESCRIPTION,BID_PRICE,START_TIME,END_TIME,BID_COUNT
FROM ITEM WHERE ITEMID = ?
```

  - ✧ SELECT transaction 2

```
SELECT BIDERID, BID_PRICE, BID_TIME FROM BID WHERE ITEMID = ?
SELECT BIDERID, BID_PRICE, BID_TIME FROM BID WHERE ITEMID = ?
```

  - ✧ UPDATE transaction 1

```
SELECT SELLERID,DESCRIPTION,BID_PRICE,START_TIME,END_TIME,BID_COUNT
FROM ITEM WHERE ITEMID =
UPDATE ITEM SET DESCRIPTION = ?,BID_PRICE = ?,START_TIME = ?,END_TIME = ? WHERE ITEMID = ?
```

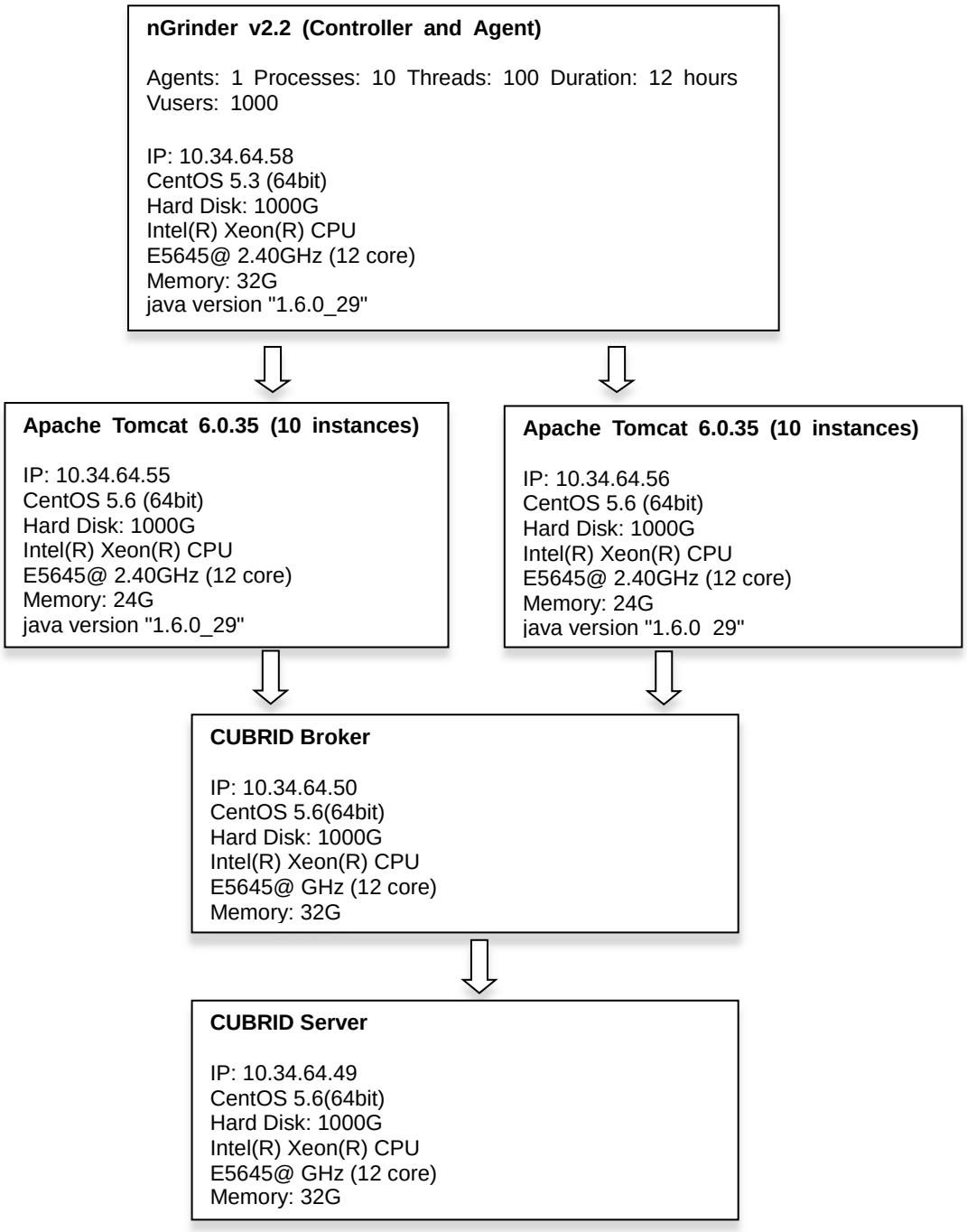
- How to Generate Load
  - ✧ How to generate load

Use two threads to generate the initial load. Each thread repeats the insert/select/update queries mentioned above. The DOTS program checks CPU usage every 5 minutes. If the Peak CPU usage does not exceed 100%, the test continues, by adding two more threads.

IV. Live Service Simulation Test Scenarios

Live Service Simulation Test is a new test suite developed by CUBRID QA. It is based on a real business application (simply called WB Service) and simulated workloads. WB Service is a web based application. It is powered by Apache Tomcat web server.

- Test Environment
  - ✧ Test Servers



Note: In order to increase stress on CUBRID, we adopted 20 Tomcat instances. It also can be reached by performance tuning on Tomcat.

- How to generate requests

WB Service uses an existing database. There are 34 tables and total 717,297,489 records existed before testing. In order to simulate workloads similar to effective user accesses, we generated totally 405,560,054 effective URLs according to the existing database data. So the testing will cover all the requests for the whole database data. WB Service supports user login and customization. It requires user authentication, but for the purpose of testing, we made some cracks to support automatic login according to a parameter in URL.

After getting URLs, we then split the whole URLs into 1000 sub collections. Each collection data is saved as one URLs' file. During the test, each URLs' file serves one v-user.

- CUBRID configuration
  - cubrid_broker.conf

```
[%BROKER1]
SERVICE                 =ON
BROKER_PORT              =33000
MIN_NUM_APPL_SERVER      =500
MAX_NUM_APPL_SERVER      =1000
APPL_SERVER_SHM_ID       =33000
LOG_DIR                 =log/broker/sql_log
ERROR_LOG_DIR            =log/broker/error_log
SQL_LOG                 =OFF
SLOW_LOG                 =OFF
TIME_TO_KILL            =120
SESSION_TIMEOUT          =300
KEEP_CONNECTION          =AUTO
CCI_DEFAULT_AUTOCOMMIT   =ON
```

  - cubrid.conf

```
[%BROKER1]
data_buffer_size=4G
log_buffer_size=4M
sort_buffer_size=2M
max_clients=1000
```

- Connection Pool in one Tomcat instance

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
        <properties>
                <property>
                        <driverClassName>cubrid.jdbc.driver.CUBRIDDriver</driverClassName>
                        <url>jdbc:cubrid:**.**.**.**:33000:wordbook:dba::?charset=UTF-8</url>
                        <username>********</username>
                        <password>********</password>
                        <maxWait>6000000</maxWait>
                        <initialSize>20</initialSize>
                        <maxActive>100</maxActive>
                        <maxIdle>5</maxIdle>
                        <minIdle>5</minIdle>
                        <Encrypt>false</Encrypt>
                        <useManagedObject>false</useManagedObject>
                        <UseStatementCache>true</UseStatementCache>
                        <StatementCacheSize>10</StatementCacheSize>
                        <UseCallableStatementCache>true</UseCallableStatementCache>
                        <UseConnectionWatcher>false</UseConnectionWatcher>
                        <UseLogger>false</UseLogger>
                        <SlowQueryTime>9999999</SlowQueryTime>
```

```
                        <QueryTimeout>9999999</QueryTimeout>
                        <validationQuery>select 1 from db_root</validationQuery>
                        <testOnBorrow>false</testOnBorrow>
                        <testWhileIdle>true</testWhileIdle>
                        <timeBetweenEvictionRunsMillis>30000</timeBetweenEvictionRunsMillis>
                        <LogAbandoned>true</LogAbandoned>
                        <RemoveAbandoned>true</RemoveAbandoned>
                        <RemoveAbandonedTimeout>50000</RemoveAbandonedTimeout>
                        <poolPreparedStatements>false</poolPreparedStatements>
                        <maxOpenPreparedStatements>20</maxOpenPreparedStatements>
                </property>
        </properties>
</configuration>
```

- Configuration in nGrinder

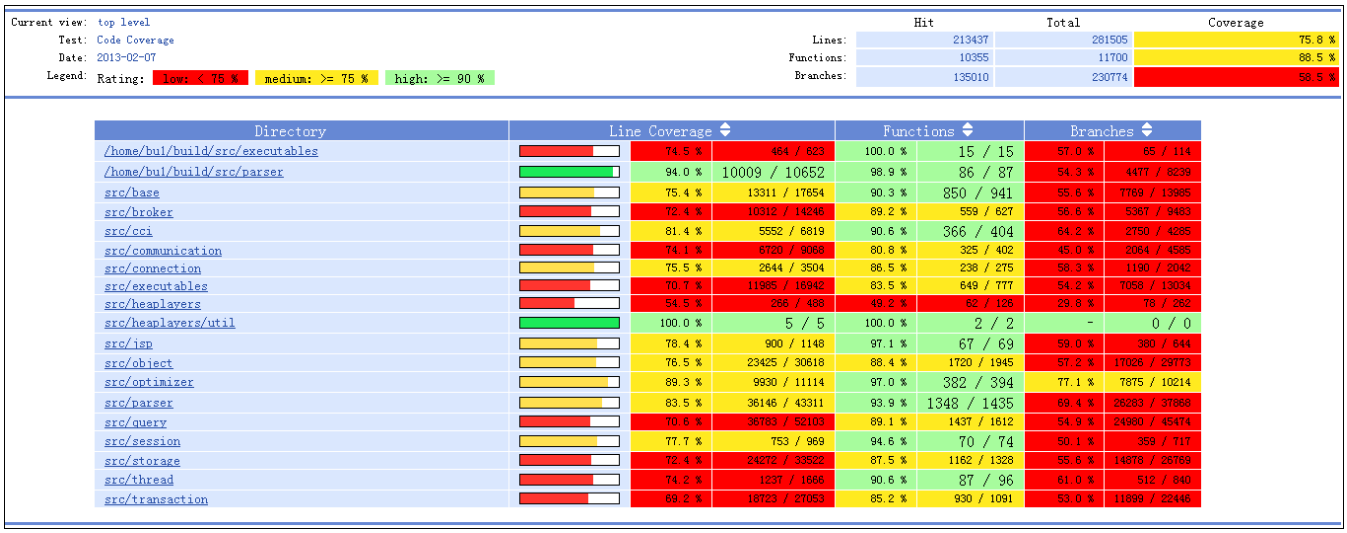
```
Agents: 1
Processes: 10
Threads: 100
V-users: 1000
Duration: 12 hours
```

Each v-user will access its URLs' file in WB_DATA directory. Once the URLs' file reaches the end line, it will go back to the head and continue executing.

```
from net.grinder.script.Grinder import grinder
from net.grinder.script import Test
from net.grinder.plugin.http import HTTPRequest

wordbook_test = Test(1, "CUBRID arpium Test")
request_00 = wordbook_test.wrap(HTTPRequest(url="http://tomcat55:19500"))
request_01 = wordbook_test.wrap(HTTPRequest(url="http://tomcat55:19501"))
request_02 = wordbook_test.wrap(HTTPRequest(url="http://tomcat55:19502"))
request_03 = wordbook_test.wrap(HTTPRequest(url="http://tomcat55:19503"))
request_04 = wordbook_test.wrap(HTTPRequest(url="http://tomcat55:19504"))
request_05 = wordbook_test.wrap(HTTPRequest(url="http://tomcat55:19505"))
request_06 = wordbook_test.wrap(HTTPRequest(url="http://tomcat55:19506"))
request_07 = wordbook_test.wrap(HTTPRequest(url="http://tomcat55:19507"))
request_08 = wordbook_test.wrap(HTTPRequest(url="http://tomcat55:19508"))
request_09 = wordbook_test.wrap(HTTPRequest(url="http://tomcat55:19509"))

request_10 = wordbook_test.wrap(HTTPRequest(url="http://tomcat56:19500"))
request_11 = wordbook_test.wrap(HTTPRequest(url="http://tomcat56:19501"))
request_12 = wordbook_test.wrap(HTTPRequest(url="http://tomcat56:19502"))
request_13 = wordbook_test.wrap(HTTPRequest(url="http://tomcat56:19503"))
request_14 = wordbook_test.wrap(HTTPRequest(url="http://tomcat56:19504"))
request_15 = wordbook_test.wrap(HTTPRequest(url="http://tomcat56:19505"))
request_16 = wordbook_test.wrap(HTTPRequest(url="http://tomcat56:19506"))
request_17 = wordbook_test.wrap(HTTPRequest(url="http://tomcat56:19507"))
request_18 = wordbook_test.wrap(HTTPRequest(url="http://tomcat56:19508"))
request_19 = wordbook_test.wrap(HTTPRequest(url="http://tomcat56:19509"))


requests = [request_00, request_01, request_02, request_03, request_04, request_05, request_06, request_07, request_08, req
uest_09,request_10, request_11, request_12, request_13, request_14, request_15, request_16, request_17, request_18, request
_19]

total_process=10

class TestRunner:
     def __init__( self ) :
          self.filename = "/home/wordbook_ngrinder/WB_DATA/WB_A" + str(grinder.processNumber%total_process+1).zfill(2) + "/
W" + str(grinder.threadNumber+1).zfill(3) + ".txt"

          self.index = grinder.threadNumber
          self.testfile = open (self.filename, "r")

     def __call__( self ) :
```

```
        line = self.testfile.readline()
        if line is None :
            self.testfile = open (self.filename, "r")
        curIndex = self.index % 20
        requests[curIndex].GET(line)
        self.index = self.index+1
    def __del__(self):
        self.testfile.close()
```

V. Scenario-based Code Coverage Results



Current view: top level
Test: Code Coverage
Date: 2013-02-07
Legend: Rating: low: < 75 % medium: >= 75 % high: >= 90 %

	Hit	Total	Coverage
Lines:	213437	281505	75.8 %
Functions:	10355	11700	88.5 %
Branches:	135010	230774	58.5 %

Directory	Line Coverage		Functions		Branches	
/home/bu1/build/src/executables	74.5 %	464 / 623	100.0 %	15 / 15	57.0 %	65 / 114
/home/bu1/build/src/parser	94.0 %	10009 / 10652	98.9 %	86 / 87	54.3 %	4477 / 8239
src/base	75.4 %	13311 / 17654	90.3 %	850 / 941	55.6 %	7769 / 13985
src/broker	72.4 %	10312 / 14246	89.2 %	559 / 627	56.6 %	5367 / 9483
src/cci	81.4 %	5552 / 6819	90.6 %	366 / 404	64.2 %	2750 / 4285
src/communication	74.1 %	6720 / 9068	80.8 %	325 / 402	45.0 %	2064 / 4585
src/connection	75.5 %	2644 / 3504	86.5 %	238 / 275	58.3 %	1190 / 2042
src/executables	70.7 %	11985 / 16942	83.5 %	649 / 777	54.2 %	7058 / 13034
src/heaplayers	54.5 %	268 / 488	49.2 %	62 / 126	29.8 %	78 / 262
src/heaplayers/util	100.0 %	5 / 5	100.0 %	2 / 2	-	0 / 0
src/isp	78.4 %	900 / 1148	97.1 %	67 / 69	59.0 %	380 / 644
src/object	76.5 %	23425 / 30618	88.4 %	1720 / 1945	57.2 %	17026 / 29773
src/optimizer	89.3 %	9930 / 11114	97.0 %	382 / 394	77.1 %	7875 / 10214
src/parser	83.5 %	36146 / 43311	93.9 %	1348 / 1435	69.4 %	26283 / 37868
src/query	70.6 %	36783 / 52103	89.1 %	1437 / 1612	54.9 %	24980 / 45474
src/session	77.7 %	753 / 969	94.6 %	70 / 74	50.1 %	359 / 717
src/storage	72.4 %	24272 / 33522	87.5 %	1162 / 1328	55.6 %	14878 / 26769
src/thread	74.2 %	1237 / 1666	90.6 %	87 / 96	61.0 %	512 / 840
src/transaction	69.2 %	18723 / 27053	85.2 %	930 / 1091	53.0 %	11899 / 22446

VI. JDBC Code Coverage Results

Package	# Classes	Line Coverage	Branch Coverage	Complexity
All Packages	98	81% 8415/10384	70% 2732/3902	3.098
cubrid.jdbc.driver	56	85% 5289/6192	74% 1496/2004	2.562
cubrid.jdbc.jci	36	73% 2763/3759	65% 1157/1779	4.783
cubrid.jdbc.log	2	92% 64/69	92% 12/13	1.393
cubrid.jdbc.net	1	52% 51/98	34% 13/38	12
cubrid.jdbc.util	1	96% 88/91	88% 23/26	2.231
cubrid.sql	2	91% 160/175	73% 31/42	2.826