dotshowto111

Table of Contents

Table of Contents

Database Opensource Test Suite
User's Guide

Version: 1.1.1
Owner: David Barrera
IBM Linux Technology Center
11501 Burnet Road
Austin, TX 78758

1. Overview

1.1 DOTS Overview

Database Opensource Test Suite (DOTS) is a set of test cases designed for the purpose of stress testing and long run testing on database systems to measure database performance and reliability. It has two kinds of test cases − Basic Cases and Advanced Cases. The primary goal of Basic Cases is stress and long run database testing; the secondary goal is 100% JDBC API coverage. There are 8 test cases written in Java to cover JDBC API under the Basic Cases category. The goal of the Advanced Cases is modeling *real−world* business logic, stress and long run testing on database systems. There are 2 test cases written in Java under the Advanced Cases category.

1.2 Environment

1.2.1 Software Requirements

	Server	Client
Operating System	Linux 2.4.4 Kernel	Linux 2.4.4 Kernel
Database System	IBM DB2 UDB for Linux Enterprise Edition 7.2	JDBC Driver: db2jdbc.zip
	Oracle 8i for Linux Enterprise Edition 8.1.7	JDBC Driver: classes12.zip
	Sybase Adaptive Server for Linux Enterprise Evaluation Edition 12.5	JDBC Driver: jconn2.jar
Other Softwares	J2SE 1.3.1 for Linux	J2SE 1.3.1 for Linux

1.2.2 Hardware Requirements

Category	Requirements
CPU	700MHz or higher
Memory	512MB or higher
Disk Capacity	2GB or higher

Notes:

1. DOTS was designed to run on any hardware platform supported by any Linux distribution. It was developed and tested on a 32 bit Intel platform running Linux.
2. The number of DOTS test cases you can simultaneously run on a test client, will depend on the physical memory available.

2. Architecture

2.1 Component Overview

There are two components that make up DOTS. The first component is the DOTS Client that runs on the test client machine. The second component is the Performance Monitor that runs on the test server machine. The DOTS client uses Sockets and JDBC calls to communicate with the Performance Monitor and the database server respectively. Performance Monitor gets system information from the server, such as CPU usage, memory usage, disk I/O, and sends it back to the test client. JDBC calls are used to perform database transactions on the server.

Neither DOTS Client nor the Performance Monitor has any GUI support in the current release. Users can start running test cases from the command line. DOTS Client can run any of the ten test cases. The test environment can be customized by modifying the configuration file. The format of the configuration file can be found in Appendix B.

2.2 DOTS Client Components

DotsClient:

DotsClient is the user interface to the DOTS application. When DotsClient is invoked, it instantiates the following objects: DotsConfig, DotsLogging, Keyboard Listener, Performance Client, Summary Writer and Testcase Threads.

DotsConfig:

DotsConfig stores all the global static variables which are used by most modules.

DotsLogging:

DotsLogging serves as the gateway to all the logging activities. It writes to the regular log, error log and summary log. Most modules of DotsClient use DotsLogging to write their output. For more details on the log file and log format, please refer to Appendix D.

Keyboard Listener:

Keyboard Listener reads keyboard input and process it. If the user enters "STOP" from the keyboard, this will set the termination flag in DotsConfig to "true". At this point, the testcase and all its threads will be terminated.

Performance Client:

Performance Client interacts with the Performance Monitor on the database server. It sends requests to the server, gets the response from the server and saves the data it gets to DotsConfig.

Summary Writer:

Summary Writer writes a test summary to the summary file at a specified time interval. The interval time between two writes can be set in the configuration file.

Test Case Threads:

Test Case Thread is an instance of a test case. Each instance of a thread has its own connection to the database server. Several threads may be started to impose enough workload on a given database server. The number of threads started are determined by the combination of the CURRENT_CONNECTIONS, AUTO_MODE and the CPU_TARGET variable in the configuration file. For the detail, please refer to Appendix B.

2.3 Performance Monitor Components

Performance Monitor:

Performance Monitor runs on the database server and communicates with the Dots client via sockets. It collects system information and sends it back to the client every five seconds. When Performance Monitor is invoked, it first reads command line parameters and instantiates Performance config, Performance Reader and Connection Sockets. Then it creates a socket and listens to a specified port.

Performance Config:

Performance Config stores performance data which is provided by Performance Reader.

Performance Reader:

Performance Reader reads performance data from the database server and saves the data into Performance Config.

Connection Sockets:

Connection Socket instance is a thread which interact with one specific client. It gets client request and sends the performance data from Performance Config to the client

3. Installation and Configuration

3.1 DOTS Packaging

The DOTS package comes in a gzipped tar file and contains five parts: source files (.java suffixes), script files for creating database and tables, sample script files for tuning system and database parameter, configuration file and make file.

After uncompressing the gzipped tar file, a new directory named Dots will be created in the current directory. The structures are as follows:

−Dots/src/: contains the source files.

Dots/src/dots/basecase/: contains source files for the eight basic cases.
Dots/src/dots/advcase/: contains source files for the two advanced cases.
Dots/src/dots/framework/: contains source files for the framework.
Dots/src/dots/perfmon/: contains source files for the performance monitor.

− Dots/scripts/: contains scripts for database operations that are used in the test server machine.

Dots/scripts/createdb_db2: used to create database "testdb" if database server is IBM DB2.
Dots/scripts/createtable_db2: used to create tables if database server is IBM DB2.
Dots/scripts/createproc_db2: used to create stored procedures if database server is IBM DB2.
Dots/scripts/createtable_ora: used to create tables if database server is Oracle.
Dots/scripts/createproc_ora: used to create stored procedures if database server is Oracle.
Dots/scripts/createdb_syb: used to creates database "testdb" if database server is Sybase
Dots/scripts/createtable_syb: used to creates tables if database server is Sybase.
Dots/scripts/createproc_syb: used to creates stored procedures if database server is Sybase.

− Dots/sample/: contains scripts used to tune Linux system parameter and tune database parameter.

Dots/sample/syscfg: a sample used to tune Linux system parameter.
Dots/sample/db2cfg: a sample used to tune IBM DB2 database parameter.

− Dots/makefile: make file used to compile the source code.

− Dots/config.ini: configuration file.

3.2 Installation

The example below assumes that Dots is installed in the /usr/tmp directory on both client and server machines. Dots can be installed in any directory.

1. Uncompress and untar the Dots.tar.gz file on the client test machine.

$ tar zxvf Dots.tar.gz

Note: Keep the directory structure intact. Compile the source code using JDK 1.3.1 or higher. Make sure that the path to JDK is in the $PATH variable. If not, add it in.

2. Run make to compile the source. Enter the following commands:

$cd /usr/tmp/Dots

$make

There will be two jar files − Dots.jar and Perfmon.jar and a directory named classes which contains the class files.

3. Copy the Perfmon.jar, scripts/createdb_<dbase>, scripts/createtable_<dbase>, and scripts/createproc_<dbase> to the server test system.

Note: <dbase> is either db2(DB2), ora(Oracle) or syb(Sybase).

3.3 Configuration

3.3.1 Server Configuration

3.3.1.1 DB2 Server Configuration

The following instructions are used for configuring DOTS with IBM DB2:

1. Download and install IBM DB2 UDB for Linux on the server machine.

2. Log in as DB2 user, start the database service.

$db2start

3. Run the script to create the database and tables.

$cd /usr/tmp/Dots/scripts

$./createdb_db2
$./createtable_db2
$./createproc_db2

Make sure scripts have execute permission.

4. Set JDBC 2.0 environment variables.

$cd $DB2_HOME/sqllib/java12

$. usejdbc2

Make sure the above commands execute successfully. If not, execute these commands manually:

$export CLASSPATH=$DB2_HOME/sqllib/java12/db2java.zip:/usr/tmp/Dots/Perfmon.jar:$CLASSPATH

$export PATH=$DB2_HOME/sqllib/java12:$PATH
$export LD_LIBRARY_PATH =$DB2_HOME/sqllib/java12:$LD_LIBRARY_PATH

$DB2_HOME

represents the home directory of DB2.

5. Start db2 jdbc service.

$db2jstrt 8083

Note: It is imperative that usejdbc2 gets sourced prior to starting the jdbc service. Port 8083 can be changed if it is taken by other applications. In this case, change the URL in config.ini to the same port number.

6. Start the Performance Monitor on the server machine.

$export CLASSPATH=/usr/tmp/Dots/Perfmon.jar:$CLASSPATH

$export PATH=/usr/<JDK version>/bin:$PATH
$java dots.perfmon.PerfMon −port <port number>

Note: The recommended port number is 8001. If it is taken, choose another and reflect the changes in the configuration file. The Performance Monitor requires a dedicated session. If for any reason the session is terminated, all test cases running are terminated.

You can retrieve usage with following command:

$java dots.perfmon.PerfMon −help or $java dots.perfmon.PerfMon −?

3.3.1.2 Oracle Server Configuration

The following instructions are used for configuring DOTS with Oracle.

1. Download and install ORACLE 8i for Linux on the server.

2. Log in as Oracle system manager and start the Database service

− Open Oracle server:

$svrmgrl

svrmgr>connect internal
svrmgr>startup
svrmgr>exit

− Open NET8'listener:

$lsnrctl

lsn>start
lsn>exit

3. Create Database

Log in as Oracle system manager and use *Oracle Database Configuration Assistant Tool* to create, change or delete a database.

$startx
$cd $ORACLE_HOME/bin

$dbassist
$ORACLE_HOME represents the home directory of Oracle.

Then the Oracle Database Configuration Assistant window will pop up. You can create database following the instructions. The database name is "testdb".

4. Create tables and procedures:

$cd /usr/tmp/Dots/scripts

$./createtable_ora
$./createproc_ora

Make sure scripts have execute permissions.

5. Start the Performance Monitor on the server machine.

$export CLASSPATH=/usr/tmp/Dots/Perfmon.jar:$CLASSPATH

$export PATH=/usr/<JDK version>/bin:$PATH
$java dots.perfmon.PerfMon −port <port number>

Note: The recommended port number is 8001. If it is taken, choose another number and reflect the changes in the configuration file. The Performance Monitor requires a dedicated session. If for any reason the session is terminated, all test cases running are terminated.

You can retrieve usage with the following command:

$java dots.perfmon.PerfMon −help or $java dots.perfmon.PerfMon −?

3.3.1.3 Sybase Server Configuration

The following instructions are used for configuring DOTS with Sybase.

1. Download and install Sybase Adaptive Server Enterprise Evaluation Version 12.5 for Linux on the server.

2. Log in as Sybase system manager and start the Database service

$cd $SYBASE_HOME/ASE/install

$startserver

$SYBASE_HOME

represents the home directory of Oracle.

3. Run the script to create the database and tables.

$cd /usr/tmp/Dots/scripts

$./createdb_syb # create database
$./createtable_syb # create tables
$./createproc_syb # create stored procedures

Make sure scripts have execute permission.

4. Start the Performance Monitor on the server machine.

$export CLASSPATH=/usr/tmp/Dots/Perfmon.jar:$CLASSPATH

$export PATH=/usr/<JDK version>/bin:$PATH
$java dots.perfmon.PerfMon −port <port number>

Note: The recommended port number is 8001. If it is taken, choose another number and reflect the changes in the configuration file. The Performance Monitor requires a dedicated session. If for any reason, the session is terminated, all the test cases running gets terminated.

You can get usage by using following command:

$java dots.perfmon.PerfMon −help or $java dots.perfmon.PerfMon −?

3.3.1.4 MySQL Server Configuration

The following instructions are used for configuring DOTS with MySQL.

1. Download and install MySQL 3.23.36 on the server.

$

rpm −Uvh mysql−3.23.36−1.i386.rpm
$rpm −Uvh mysql−server−3.23.36−1.i386.rpm

2. Log in as root and start the Database service

$/etc/rc.d/init.d/mysqld start

3. Change to the directory which contains the script and run the script to create the database and tables.

$mysql < createdb_mysql

#create database and grant the user "dots" the privilege to access the database remotely.
$mysql <createtable_mysql # create tables

Make sure scripts have execute permission.

4. Start the Performance Monitor on the server machine.

$export CLASSPATH=/usr/tmp/Dots/Perfmon.jar:$CLASSPATH

$export PATH=/usr/<JDK version>/bin:$PATH
$java dots.perfmon.PerfMon −port <port number>

Note: The recommended port number is 8001. If it is taken, choose another number and reflect the changes in the configuration file. The Performance Monitor requires a dedicated session. If for any reason, the session is terminated, all the test cases running gets terminated.

You can get usage by using following command:

$java dots.perfmon.PerfMon −help or $java dots.perfmon.PerfMon −?

3.3.1.4 PostgreSQL Server Configuration

The following instructions are used for configuring DOTS with PostgreSQL.

1. Create a linux user account i.e., postgres to own and manage the PostgreSQL database files.

```
useradd postgres
```

2. Download and install the following PostgreSQL packages in the given order:

```
a) postgresql−7.2.2−1PGDG.i382.rpm
b) postgresql−jdbc−7.2.2−1PGDG.i386.rpm
c) postgresql−libs−7.2.2−1PGDG.i386.rpm
d) postgresql−odbc−7.2.2−1PGDG.i386.rpm
e) postgresql−perl−7.2.2−1PGDG.i386.rpm
f) postgresql−python−7.2.2−1PGDG.i386.rpm
g) postgresql−server−7.2.2−1PGDG.i386.rpm
```

3. Change the current directory to the home directory of the user postgres (/var/lib/pgsql).

4. Copy the .bash_profile and .bashrc from /etc/skel/ directory.

6. Modify the .bash_profile file: (a) Modify the PATH to

PATH=$PATH:$HOME/bin (b) Add the following lines to .bash_profile file:

```
PATH=$PATH:/usr/lib/pgsql
export PATH
PGLIB=/usr/lib/pgsql/:/var/lib/pgsql/data
PGDATA=/var/lib/pgsql/data
LD_LIBRARY_PATH=/usr/lib/pgsql
ENV=$HOME/.bashrc
export ENV PGLIB PGDATA LD_LIBRARY_PATH
Also you might have to add the following:
  export PATH="bin path of your JDK/SDK":$PATH
  export CLASSPATH="path of Dots"/Perfmon.jar:$CLASSPATH
```

7. Execute the command: chown −Rf postgres.postgres /var/lib/pgsql

8. Logout and login again as root

9. su − postgres

10. Execute the command: initdb

11. Edit the pg_hba.conf file in /var/lib/pgsql/data Add the client's ipaddress to connect to database "TESTDB", use the format:

```
TYPE DATABASE IP_ADDRESS MASK  AUTH_TYPE
```

For example to allow users from 9.3 network, to connect to any database:

```
host   all     9.3.0.0 255.255.0.0     trust
```

12. Start the server by using the commands: /usr/bin/postmaster −i −D /var/lib/pgsql/data −l logfile start
To stop the server: /etc/rc.d/init.d/postgresql stop

13. Make sure the following scripts have the execute permission and execute the scripts to create the database and its objects: (i) cd to the Dots/scripts directory (ii) ./createdb_pg (iii) ./createtable_pg (iv) ./createproc_pg

14. Start the Performance Monitor on the server machine:
java dots.perfmon.PerfMon −port
You can get usage by using following command:
java dots.perfmon.PerfMon −help or java dots.perfmon.PerfMon −?
Note: The recommended port number is 8001. If it is taken, choose another number and reflect the changes in the configuration file. The Performance Monitor requires a dedicated session. If for any reason, the session is terminated, all the test cases running gets terminated.

3.3.2 Client Configuration

3.3.2.1 Set environment

Download the JDBC driver from the database vendor's website and update the CLASSPATH variable

accordingly.

Install JDK 1.3.1 or higher on the client machine. Export the path for the java runtime executable.

$export PATH=/path_to_java/bin:$PATH

1. For DB2

DB2 JDBC driver file is db2java.zip. The driver is in the directory *$DB2_HOME/sqllib/java12*. Copy the db2java.zip file from the test server machine into the test client's DOTS local directory. Update the CLASSPATH variable with DB2 JDBC driver.

$export CLASSPATH=/usr/tmp/db2java.zip:$CLASSPATH

You should also add DOTS client jar files to CLASSPATH

$export CLASSPATH=/usr/tmp/Dots/Dots.jar:$CLASSPATH

2. For Oracle:

Oracle JDBC driver file is classes12.zip (on Linux), and can be download it from the Oracle web site. Update the CLASSPATH variable with the Oracle JDBC driver.

$export CLASSPATH=usr/tmp/classes12.zip:$CLASSPATH

3. For Sybase:

Sybase JDBC driver file is jconn2.jar. The driver is in the directory *$SYBASE_HOME/sqllib/java12*. Copy the jconn2.jar file from the test server machine into the test client's DOTS local directory. Update the CLASSPATH variable with Sybase JDBC driver.

$export CLASSPATH=/usr/tmp/jconn2.jar:$CLASSPATH

You should also add DOTS client jar files to CLASSPATH

$export CLASSPATH=/usr/tmp/Dots/Dots.jar:$CLASSPATH

4. For MySQL:

MySQL JDBC driver file is mm.mysql−2.0.14. This driver can be downloaded at the web site: https://sourceforge.net/project/showfiles.php?group_id=15923. Unjar the file mm.mysql−2.0.14−you−must−unjar−me.jar and copy the directory mm.mysql−2.0.14 from the test server machine into the test client's DOTS local directory. Update the CLASSPATH variable with MySQL JDBC driver.

$export CLASSPATH=/usr/tmp/mm.mysql−2.0.14/:$CLASSPATH

Because MySQL 3.23 do not support Stored Procedure, The basic case BTCJ8 cannot run against MySQL.

You should also add DOTS client jar files to CLASSPATH

$export CLASSPATH=/usr/tmp/Dots/Dots.jar:$CLASSPATH

5. For PostgreSQL: Download and install the package: postgresql−jdbc−7.2.2−1PGDG.i386.rpm. Copy the following files from /usr/share/pgsql to the Dots/pgsql directory:

```
jdbc7.1−1.1.jar
jdbc7.1−1.2.jar
jdbc7.2dev−1.1.jar
```

Update the CLASSPATH variable with the postgresql JDBC driver.

```
export CLASSPATH=$CLASSPATH:/dots/Dots/pgsql/jdbc7.1−1.2.jar:/usr/share/pgsql/jdbc7.1−1.2.
   jdbc7.2dev−1.2.jar
```

You should also add DOTS client jar files to
CLASSPATH

$export CLASSPATH=/usr/tmp/Dots/Dots.jar:$CLASSPATH

3.3.2.2 Customize Configuration File

DOTS client is configured through a text file called the DOTS Configuration File. This file may have any name you desire, but the default is config.ini. User can edit configuration file to customize working environments for DOTS. Various parameters are provided for user to control how test cases in DOTS are running against database systems. Please refer to Appendix B for details.

3.4 Running DOTS

Prior to running any of the DOTS test cases, it is imperative that the Performance Monitor and the Database Services have been started in the test server machine.

3.4.1 Run DOTS client

1. Log in as the user that has execution authority for java.

2. Execute any of the 10 test cases.

$java dots.framework.Dots [−config <config file name>] −case <test case name>

−config: Allows you to specify the config file. The default is config.ini in the current path.

<config file name> : If file name is specified without a path, the current path is assumed; otherwise specify full name.

−case: Specifies which case to run.

<test case name>: The 8 basic test cases are named BTCJ1 −− BTCJ8; the 2 advanced cases are named ATCJ1 and ATCJ2.

You can retrieve usage with the following command:

$java dots.framework.Dots −help or $java dots.framework.Dots −?

3. When running more than one test case, open another session and follow steps 1 and 2. The number of test cases and concurrent connections to the database is influenced on the amount of memory available on both the client and server machines. On a client system with 512 Megabytes of physical memory, a user can run 4 test cases with 25 concurrent connections. There are a couple of test cases, BTCJ7 and ATCJ2 that require a lot of memory. Run only one of these on a single client system.

4. Function Description

There are 10 test cases (8 Basic Cases and 2 Advance Cases) in DOTS and they are all written in JAVA. Every test case is independent from each other.

Note: For a function description of Performance Monitor, refer to section 2.3.

4.1 Basic Cases

There are 8 basic cases in DOTS. The primary goal of these test cases are stress testing and/or long run database testing; the secondary goal is 100% test coverage of the JDBC APIs.

4.1.1 BTCJ1

The main function of this test case is to get database meta data. Database meta data is the comprehensive information about the database, which includes driver name, driver version, database product name and version, transaction isolation level, columns in the table and so on. The case also retrieves data from table BASIC1, BASIC2, BASIC3. For details about the tables, refer to Appendix A.

4.1.2 BTCJ2

This test case mainly uses SQL commands to execute database operations such as insert, update, select and delete. This test case uses tables BASIC1, BASIC2, BASIC3. For details about the tables, refer to Appendix A.

4.1.3 BTCJ3

This test case works like BTCJ2, but sends multiple SQL statements to the database as a unit, or batch. It uses tables BASIC1, BASIC2, BASIC3. For details about the tables, refer to Appendix A.

4.1.4 BTCJ4

This test case uses PreparedStatement to execute database operations such as insert, update, select and delete. The operations are based on a single table. It uses table BASIC4. For details about the tables, refer to Appendix A.

4.1.5 BTCJ5

This test case uses PreparedStatement to execute database operations such as insert, update, select and delete. It uses tables BASIC1, BASIC2, BASIC3. For details about the tables, refer to Appendix A.

4.1.6 BTCJ6

This test case mainly manipulates SQL3 data type CLOB (Character Large Object). It sends CLOBs to the database and accesses SQL CLOB values. This test case uses table BASIC5. For details about the tables, refer to Appendix A.

4.1.7 BTCJ7

This test case manipulates SQL3 data type BLOB (Binary Large Object). It sends BLOB to the database and accesses an SQL BLOB values. This test case uses table BASIC6. For details about the tables, refer to Appendix A.

4.1.8 BTCJ8

This test case uses CallableStatement to execute database operations such as insert, update, select and delete. This test case uses tables BASIC1, BASIC2, BASIC3. For details about the tables, refer to Appendix A.

4.2 Advance Case

There are 2 advanced test cases, which are modeling *real−world* business logic.

4.2.1 ATCJ1 User Registration/Authentication

4.2.1.1 Business Scenario

This test case simulates the database actions of new user registration, updating existing user information and user authentication.

4.2.1.2 Database Actions

- Continuously insert new user registration information into the user information table
- Continuously check userid/password to validate user login
- Continuously update user information table for record update

4.2.1.3 Database Schema

Table Registry. Details refer to Appendix A.

4.2.2 ATCJ2 On−line Auction

4.2.2.1 Business Scenario

This test case simulates the database actions of an online auction scenario. Buyers can search auction items for detailed information, and bid on the items (only for registered users). Sellers (registered users) will add/update items. Everyone can view bid history of one item.

4.2.2.2 Database Actions

- Continuously search auction items for detail information (Buyer)
- Continuously update auction prices for the items to bid (Buyer)
- Continuously insert new auction items for sale (Seller)
- Continuously search all the auction items on which one particular user has transaction with. (Buyer and Seller)

4.2.2.3 Database Schema

This test case uses tables REGISTRY, ITEM and BID. For details about the tables, refer to Appendix A.

5. Q&A

Q1:

Why do I get " Database connection failed" when trying to run one of the test cases?

A1:

Check the following settings:

1. Check if the database and its respective services have been started successfully.

2. Check if JDBC drivers were added to the CLASSPATH variable.

3. Check if the environment variables are correct.

4. For DB2, make sure client and server machine both use the db2 jdbc 2.0 driver. Check such parameters as DriverClass, URL, UserID and Password in configuration file.

Refer to the error file for detail description.

Q2:

When running DOTS in a DB2 environment, there are Java.lang.AbstractMethodErrors. How can this be corrected?

A2:

Both client and server should use the db2 jdbc 2.0 driver.

For the Server side problem, one possible reason is db2 JDBC 2.0 driver file should be in the CLASSPATH. Another possible reason, the JDBC 1.0 driver file is ahead of the JDBC 2.0 in the CLASSPATH variable.

DOTS requires the JDBC 2.0 driver file db2java.zip at the beginning of the CLASSPATH. This rule applies to both test client and test server.

Q3:

There are a lot of DB2 Exceptions in the error files. Some of these exception are:
COM.ibm.db2.jdbc.DB2Exception: [IBM][CLI Driver][DB2/LINUX] SQL0803N One or more values in the INSERT statement, UPDATE statement, or foreign key update caused by a DELETE statement are not valid because the primary key, unique constraint or unique index identified by "1" constrains table "DB2INST1.REGISTRY" from having duplicate rows for those columns. SQLSTATE=23505. Why?

A3:

This problem is not a functional error but a data feed error. The advance cases are trying to simulate user registration and online auction. The key data that gets generated for insert into the database table follows a special rule. However, in a multithread environment, duplicate keys are a normal occurrence. The primary key mechanism in the target table does not allow duplicate keys. Thus, an exception gets generated. For example,

in eBay or Yahoo, if you register as "MIKE" and a user MIKE already exists, it will return an error saying MIKE user id is already in use. This error is similar to this type of exception. However, the exceptions are not so numerous unless two or more instances of the same case run against the same database at the same time.

Q4:

When running DOTS for several minutes in a DB2 environment, there are instances of java.lang.OutOfMemoryError in the log file. Why and how to solve?

A4:

This error can be fixed by changing JVM configuration as follows:

java −Xms30m −Xmx500m −Xss9m dots.framework.Dots −config config.ini −case <case name>
These three parameters are used to modify JVM default setting.
−Xms30m is to set initial Java heap size to 30M
−Xmx500m is to set maximum Java heap size to 500M
−Xss9m is to set thread stack size to 3M .

You can change the number according to client machine's configuration.

Q5:

When running DOTS for several hours in an ORACLE environment, the error, ORA−04031: unable to allocate 4200 bytes of shared memory ("shared pool"," ", "sga heap", "state objects") appears in the error file. How can I fix this problem?

A5:

It is caused by lack of system resource. You can tune the OS parameters and Oracle parameters.
To tune Linux system parameters, you can log in as root user and run the script in directory */usr/tmp/Dots/sample*:

$cd /usr/tmp/Dots/sample

*$./*syscfg

Notes: The file "syscfg" is only a sample. System paremeters depend on the hardware configuration. You can edit "syscfg" to modify the parameters.

To tune Oracle configuration parameters, you should modify the init*.ora file(* represents database name):

Open_cursor = 800
Db_block_buffers = 45056
Shared_poor_size = 314572800
Large_pool_size = 6144000
Processes = 800
Log_buffer = 1638400

Notes: Above parameters are only a sample. The parameters depend on the hardware configuration and Linux system parameters.

Q6:

After running DOTS for several minutes, DB2 crashed. How can I prevent DB2 from crashing?

A6:

DOTS was designed for stress testing and/or long run testing on any given database systems to measure database performance and reliability. The disk space and memory allocated by the database manager are not sufficient to meet DOTS needs. You should tune the DB2 parameters and Linux parameters to achieve maximum performance.

To tune Linux system parameters, log in as root user and enter the commands:
$cd /usr/tmp/Dots/sample
*$./*syscfg

To tune the following DB2 parameters, log in as DB2 user and enter the commands:
$cd /usr/tmp/Dots/sample
*$./*db2cfg

Notes: The tuning in "syscfg" and "db2cfg" is for reference. The parameters of Linux system depend on the hardware configuration. The parameters of DB2 depend on both hardware configuration and Linux system parameters.

Q7:

When running test case BTCJ3 in an Sybase environment, the error:
"com.sybase.jdbc2.jdbc.SybBatchUpdateException: JZ0BE: BatchUpdateException: Error occurred while executing batch statement: Your server command (family id #0, process id #165) encountered a deadlock situation. Please re−run your command " appears in the error file. Why?

A7:

This exception is not a functional error. The occurence of this exception depends on how the database driver executes SQL commands. This exception occurs when Sybase responds to many SQL commands from DOTS client. Oracle and DB2 can handle these situation better.

Q8:

When running DOTS with Sybase, the log: "server 15 task(s) are sleeping waiting for space to become available in the log segment for database tempdb" appears in the Sybase log. How to avoid this?

A8:

You can avoid this problem by enlarging the size of tempdb and dump log automatically. To do this, you should create a separate device and move tempdb to that device following below steps:
#move tempdb to the device testdb_dev(assume testdb_dev is the device you have created.)
>use master
>go

```
>alter database tempdb on testdb_dev
>go
>use tempdb
>go

#Add a new segment and extend the segment
>sp_addsegment tempdb_seg,tempdb,testdb_dev
>go
>sp_extendsegment tempdb_seg,tempdb,testdb_dev
>go

#Dump log automatically
>create procedure sp_thresholdtempdb as dump transaction tempdb with no_log
>go
>sp_addthreshold tempdb,tempdb_seg,150,sp_thresholdtempdb
>go
```

But if we do so, we have not found way to delete that device later when we re−run those scripts.

Q9:

Is it necessary to use disk init to initialize a database device if I want to create a very large database for Sybase?

A9:

Yes, it is necessary to use disk init to initialize a database device if you want to create a large database.

Q10:

When running test case BTCJ6 or BTCJ7 on Sybase, we have experienced the error: "[Fri Oct 12 17:08:11 CDT 2001] BTCJ6.populateTable(): com.sybase.jdbc2.jdbc.SybSQLException: The name 'DOC' is illegal in this context. Only constants, constant expressions, or variables allowed here. Column names are illegal." or the error: "[Fri Oct 12 17:08:48 CDT 2001] BTCJ7.populateTable2(): com.sybase.jdbc2.jdbc.SybSQLException: The name 'PHOTO' is illegal in this context. Only constants, constant expressions, or variables allowed here. Column names are illegal." And server's CPU utilization is low.Why?

A10:

BTCJ6 and BTCJ7 are designed to manipulate CLOB and BLOB respectively. Different databases use different implementations to leverage their unique features and strength, these two test cases can run against DB2 and Oracle but not for Sybase.

Q11:

How about the performance of each case?

A11:

Each case's performance depend on the configuration of both client machine and server machine. Below table is for reference:

DOTS Server − IBM eServer x200−52x/PIII 866/1280MB + IBM DB2 UDB for Linux Enterprise Edition 7.2

DOTS Client − IBM eServer x200−52x/PIII 866/640MB

DOTS	Measure Parameters	BTCJ1	BTCJ2	BTCJ3	BTCJ4	BTCJ5	BTCJ6	BTCJ7	BTCJ8	ATCJ1	ATCJ2
Client resource	No. of Threads reach CPU target ~ after stable	19	4	4	3	3	4	14	5	28	23
	Total Elapse Time reach CPU Target (min)	54	9	9	10	6	9	39	12	81	66
	Used ~ Total Memory after system stable (MB)	103	100	101	150	101	103	108	102	102	104
	Used CPU after system stable (%)	35	10	10	10	10	10	94	20	30	90
Server resource	Used ~ Total Memory after system stable (MB)	533	625	638	568	640	657	635	605	761	750
	Used CPU after system stable (%)	80	96	89	89	96	88	88	97	60	43
Remarks	CPU Utilization Target (%)	85	85	85	85	85	85	85	85	85	85
	Thread Creation Interval (min)	3	3	3	3	3	3	3	3	3	3

Before running the case, the resources are as belows:

	Used Memory − Idle	Used CPU − Idle
DOTS Client Resource	56M	1%
DOTS Server Resource	176M	1%

6. Warranty Disclaimer Information

This document is provided on an "AS IS" basis, with no warranties of any kind, including, but not limited to, the implied warranties of merchantibility or fitness for particular purpose. The information contained in this document is subject to change without notice. It shall be the user's responsibility to take all appropriate fail−safe, backup, and other measures to ensure the safe use of the application. The author disclaims any liability for any damages caused by use of the DOTS program or this documentation. .

Appendix A. Database Schema

TABLE1: BASIC1

Column Name	Column Type	Column Description
ID_1	CHAR(10) NOT NULL	PRIMARY KEY
RND_CHAR	VARCHAR(50)	
RND_FLOAT	FLOAT	

TABLE2: BASIC2

Column Name	Column Type	Column Description
ID_2	CHAR(10) NOT NULL	PRIMARY KEY
RND_INTEGER	INTERGER	
RND_TIME	TIME	
RND_TIMESTAMP	TIMESTAMP	

TABLE3: BASIC3

Column Name	Column Type	Column Description
ID_1	CHAR(10) NOT NULL	PRIMARY KEY
ID_2	CHAR(10) NOT NULL	PRIMARY KEY
RND_DATE	DATE	
RND_INT	INTEGER	

TABLE4: BASIC4

Column Name	Column Type	Column Description
ID_4	CHAR(15) NOT NULL	PRIMARY KEY
NAME	VARCHAR(30)	
AGE	INTEGER	
SALARY	FLOAT	
DEPTNO	INTEGER	

TABLE5: BASIC5

Column Name	Column Type	Column Description
ID_5	CHAR(15) NOT NULL	PRIMARY KEY
NAME	VARCHAR(30)	
AGE	INTEGER	
SALARY	FLOAT	
DOC	CLOB(1000 K)	

TABLE6: BASIC6

Column Name	Column Type	Column Description
ID_6	CHAR(15) NOT NULL	PRIMARY KEY
NAME	VARCHAR(30)	
AGE	INTEGER	
SALARY	FLOAT	
PHOTO	BLOB(1000 K)	

TABLE7: REGISTRY

Column Name	Column Type	Column Description
UID	CHAR(15) NOT NULL	PRIMARY KEY
PASSWD	CHAR(15)	
EMAIL	CHAR(40)	
ADDRESS	CHAR(200)	
PHONE	CHAR(15)	

TABLE8: BID

Column Name	Column Type	Column Description
ItemID	CHAR(15) NOT NULL	FOREIGN KEY
BIDERID	CHAR(15)	
BID_PRICE	FLOAT	
BID_TIME	DATE	

TABLE9: ITEM

Column Name	Column Type	Column Description
ITEM_ID	CHAR(15) NOT NULL	PRIMARY KEY
SELLERID	CHAR(15) NOT NULL	
DESCRIPTION	VARCHAR	
START_TIME	DATE	
END_TIME	DATE	
BID_PRICE	FLOAT	
BID_COUNT	INTEGER	

Appendix B. Configuration File

Users can use the configuration file to customize the working environment for DOTS. Various parameters are provided for users to control how test cases are running against a database server.

Detailed explanations for configuration parameters and their values are as follows:

- Duration:

Specify how long a test case in DOTS will run, in hours. The value of Duration ranges from one minute to any hours specified. Default is 24h hours. The format is hh:mm.
Sample: *DURATION = 24:00*The above sample means a duration of 24 hours.

- Output Directory:

Specify where a test case in DOTS stores its output, such as log files, error reports and test summaries. Default Output Directory is */usr/local/DOTS/Output* while DOTS is installed in */usr/local/DOTS* by default.
Sample: *LOG_DIR = /usr/local/DOTS/Output* The above sample will notify DOTS to generate output files (such as log files, error reports and test summaries) in */usr/local/DOTS/Output* .

- Concurrent Connections:

Specify the concurrent database access connections will be created by a test case in DOTS if *AUTO_MODE = no*. The actual number of connections created will depend on the database settings.
Sample: *CONCURRENT_CONNECTIONS = 30* The above sample means at least 30 database access threads will be created to generate workload for a database system while a test case is running.

- CPU Utilization Target:

Specify the target level of CPU Utilization of Database Server while DOTS is running. If a test case cannot achieve this CPU Utilization target even when *ConcurrentConnections* of concurrent database connections are created, then more concurrent database access threads will be created to meet this target. Default is 75%. The value ranges from 75% to 100%.
Sample: *CPU_TARGET = 75* The above sample means the bottom level for CPU Utilization of Database Server should be 75%.

- Auto Mode:

Specify whether to run the test automatically to have enough work load to meet the CPU utilization target. If set to "yes", DOTS will automatically add database access workload trying to meet the CPU utilization target. If set to "no", then DOTS starts the specified number of Connections.
Sample: *AUTO_MODE = yes*

- Summary Interval:

Specify the interval time between two writes of summary report to test summary file. The default is 30 minutes.
Sample: *SUMMARY_INTERVAL = 30*

- Database Connection Parameters:

Specify parameters needed to customize the database connection environment for a test case in DOTS, such as user id, password, etc. The number and content of parameters will vary for different test cases and will be specified in the design for each test case in DOTS.
Sample for DB2:

UserID = db2inst1

− User ID for connection
Password = db2inst1 − Password for connection
DriverClass =Com.ibm.db2.jdbc.net.DB2Driver
URL = jdbc:db2://<IP Address>:8083/dotstest1 − URL for connection

Sample for PostgresSQL:

UserID = postgres

− User ID for connection
Password = "" − Password for connection
DriverClass = org.postgresql.Driver /* CASE SENSITIVE*/
URL = jdbc:postgresql://<IP Address>/TESTDB /* CASE SENSITIVE */ − URL for connection

- Server IP Address:

Specify the database server IP address.
Sample: *SERVER_IP = 10.10.10.1*

- Server Port:

Specify the port that performance monitor uses.
Sample: *SERVER_PORT = 8001*

- Max Rows:

Specify the maximum rows a table can have. The default is 10,000.
Sample: *MAX_ROWS= 10000.*

- Max Log File Size:

Specify the maximum file size a log file can occupy. The default is 100M.
Sample: *MAX_LOGFILESIZE= 104857600.*

- Thread Creation Interval:

Specify the thread creation interval. The Default is 3 minutes, the maximum is 5 minutes, minimum is 1 minutes.
Sample: *CREATIONINTERVAL = 3.*

Appendix C. Keyboard Control

Users can stop execution of a test case in DOTS by entering predefined key combinations (then pressing the *enter* key) at any time from the console window in which the test case is running. All active threads within this test case will be terminated. Test cases running in other console windows will not be affected.

Sample:

The default is *STOP*.

Appendix D. Output

Output of a test case in DOTS includes log file, error report, test summary and screen status update. Users can obtain the comprehensive information from output files (log file, error report and test summary) of a test case in DOTS.

D.1 Log File

Test cases in DOTS record database access activities and database response information in Log File.

The log file is named as "CASE−timestamp(yyyy−mm−dd−hh−mm−ss−mmm).log ".

For example: The log file name may be like BTCJ4−2001−9−29−11−8−25−745.log.

Sample Parts of Log File:

[Sat Sep 29 11:08:25 EDT 2001] Database Opensource Test Suite V1.0
[Sat Sep 29 11:08:25 EDT 2001] Start to run JDBC API Test Case − BTCJ4
[Sat Sep 29 11:08:25 EDT 2001] Initialization started
[Sat Sep 29 11:08:25 EDT 2001] Starting Performance Monitor client ...OK
[Sat Sep 29 11:08:25 EDT 2001] Client Socket: Socket[addr=<server machine's name>/<server machine's IP>,port=8001,localport=2592]
[Sat Sep 29 11:08:41 EDT 2001] Testing Database Connections ...OK
[Sat Sep 29 11:08:41 EDT 2001] Starting Summary Writer ... OK
[Sat Sep 29 11:08:41 EDT 2001] Starting Keyboard Thread ... OK
[Sat Sep 29 11:08:49 EDT 2001] Active Threads = 1 Average CPU Usage = 22%
[Sat Sep 29 11:09:49 EDT 2001] Active Threads = 1 Average CPU Usage = 10%
[Sat Sep 29 11:10:49 EDT 2001] Active Threads = 1 Average CPU Usage = 18%
[Sat Sep 29 11:11:45 EDT 2001] 3 Minutes Average CPU Usage = 13%
[Sat Sep 29 11:11:50 EDT 2001] Active Threads = 2 Average CPU Usage = 19%
[Sat Sep 29 11:12:50 EDT 2001] Active Threads = 2 Average CPU Usage = 33%
[Sat Sep 29 11:13:50 EDT 2001] Active Threads = 2 Average CPU Usage = 41%
[Sat Sep 29 11:14:45 EDT 2001] 3 Minutes Average CPU Usage = 35%
[Sat Sep 29 11:14:50 EDT 2001] Active Threads = 3 Average CPU Usage = 41%
[Sat Sep 29 11:15:51 EDT 2001] Active Threads = 3 Average CPU Usage = 57%
[Sat Sep 29 11:16:51 EDT 2001] Active Threads = 3 Average CPU Usage = 56%
[Sat Sep 29 11:17:46 EDT 2001] 3 Minutes Average CPU Usage = 56%
[Sat Sep 29 11:17:51 EDT 2001] Active Threads = 4 Average CPU Usage = 63%
[Sat Sep 29 11:18:51 EDT 2001] Active Threads = 4 Average CPU Usage = 72%
[Sat Sep 29 11:19:51 EDT 2001] Active Threads = 4 Average CPU Usage = 79%
[Sat Sep 29 11:20:47 EDT 2001] 3 Minutes Average CPU Usage = 76%
[Sat Sep 29 11:20:47 EDT 2001] CPU Target 75% is achieved now.
[Sat Sep 29 11:20:52 EDT 2001] Active Threads = 4 Average CPU Usage = 82%
[Sat Sep 29 11:21:52 EDT 2001] Active Threads = 4 Average CPU Usage = 88%
[Sat Sep 29 11:22:52 EDT 2001] Active Threads = 4 Average CPU Usage = 90%
[Sat Sep 29 11:23:52 EDT 2001] Active Threads = 4 Average CPU Usage = 91%
[Sat Sep 29 11:24:52 EDT 2001] Active Threads = 4 Average CPU Usage = 92%
[Sat Sep 29 11:25:47 EDT 2001] 5 Minutes Average CPU Usage = 86%
. . . . . .
. . . . . .

[Sat Sep 29 19:17:30 EDT 2001] Dots is Terminating ...
[Sat Sep 29 19:17:30 EDT 2001] Writing summary to Summary File...

D.2 Error Report

Test Cases in DOTS record error messages received from the Database in Error Report.

The error file is named as "CASE−timestamp(yyyy−mm−dd−hh−mm−ss−mmm).err ".

For example: The error file name may be like BTCJ3−2001−07−06−09−10−20−332.err.

Sample Parts of Error Report:

[Thu Aug 23 10:17:44 CST 2001] BTCJ3.populateTables() java.sql.BatchUpdateException: [IBM][CLI Driver][DB2/LINUX] SQL0803N One or more values in the INSERT statement, UPDATE statement, or foreign key update caused by a DELETE statement are not valid because the primary key, unique constraint or unique index identified by "1" constrains table "DB2INST1.BASIC1" from having duplicate rows for those columns. SQLSTATE=23505

D.3 Test Summary

Test Cases in DOTS record summary and statistic information of a test case execution in Test Summary.

The summary file is named as "CASE−timestamp(yyyy−mm−dd−hh−mm−ss−mmm).sum".

For example: The summary file name may be like BTCJ4−2001−9−29−11−8−25−745.sum.

Sample Parts of Test Summary:

[Sat Sep 29 11:08:25 EDT 2001] Start to run JDBC API Test Case − BTCJ4
[Sat Sep 29 11:08:25 EDT 2001] The Linux Kernel Version is 2.4.4−64GB−SMP
[Sat Sep 29 11:08:25 EDT 2001] The Database Server's CPU Count is 8
[Sat Sep 29 11:28:41 EDT 2001]
<Total Execution Time> 0 hours 20 minutes.

<Current Concurrent DB Connections> 4.

<JDBC APIs> Total number of QUERY 28523.
Total number of UPDATE 12655.
Total number of INSERT 34223.
Total number of DELETE 0.
Total number of FAILED 0.

<Average CPU> Peak of this Interval: 97%
Average of this Interval: 61%
Peak of all: 97%
Average of all: 61%

<CPU0> Peak of this Interval: 99%
Average of this Interval: 62%

Peak of all: 99%
Average of all: 62%

<CPU1> Peak of this Interval: 98%
Average of this Interval: 61%
Peak of all: 98%
Average of all: 61%
. . . . . .
. . . . . .
<CPU7> Peak of this Interval: 97%
Average of this Interval: 61%
Peak of all: 97%
Average of all: 61%

<Memory> Peak of this Interval: 752M
Average of this Interval: 675M
Peak of all: 752M
Average of all: 675M

<Disk IO> Peak of this Interval: 305/s
Average of this Interval: 207/s
Peak of all: 305/s
Average of all: 207/s

<Page In> Peak of this Interval: 124/s
Average of this Interval: 1/s
Peak of all: 124/s
Average of all: 1/s

<Page Out> Peak of this Interval: 20896/s
Average of this Interval: 1019/s
Peak of all: 20896/s
Average of all: 1019/s

. . . . . .

[Sat Sep 29 19:17:30 EDT 2001] Waiting for the active threads to exit .....
[Sat Sep 29 19:17:30 EDT 2001] All Database Access Threads exit.