Using DOTS as Apache Derby System Test

Date: 02/16/2005

Author: Ramandeep Kaur
ramank@yngvi.org

Table of Contents

1 Introduction

Database Open Source Test Suite (DOTS) is a test suite designed for stress testing on database systems. Apache Derby was tested with DOTS to investigate its usefulness as a system test. After several iterations of the test, it was found that DOTS may provide some value to Derby as a system test. This document is intended to provide information to the Derby community regarding the use of DOTS as a system test suite.

The document initially provides a brief introduction to DOTS and then provides the information on setting up DOTS to run with Derby, and details of the issues/problems that were found during Derby system testing with DOTS.

1.1 Notes

This document contains references to urls. The urls may change anytime. If there is any problem with any url, post it to the Derby mailing list (derby-dev@db.apache.org).

2 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. [2]

2.1 DOTS Test Cases

DOTS suite has 10 test cases that provide robust and complex version of testing by doing the database testing close to the production use of the system.

DOTS basic cases mainly gets database meta data, uses single and multiple SQL commands to execute database operations (insert, update, select and delete), uses PreparedStatement and CallableStatement to execute database operations, manipulates SQL3 data types CLOB (Character Large Object) and BLOB (Binary Large Object), sends CLOBs and BLOBs to the database, and accesses SQL CLOB and BLOB values. [2]

DOTS advanced cases tests real-world business scenario.
DOTS advanced test case 1 simulates the database actions of new user registration, updating existing user information and user authentication. This test case does the following 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 [2]

DOTS advanced test case 2 simulates the database actions of an online auction scenario. This test case does the following 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) [2]

For DOTS test cases, initially a database is created with multiple tables. DOTS test cases continuously inserts and updates the data into the database for duration of time specified in DOTS configuration file making the database to grow in size. In case of Network Server, the test cases continuously creates concurrent database connections as per specifications in DOTS configuration file. The duration of test cases, number of concurrent connections, and other parameters can be specified by user in the DOTS configuration files.

2.2 DOTS Database Schema

DOTS test cases uses 9 tables with the following data types:
- CHAR
- VARCHAR
- FLOAT
- INTEGER
- TIME
- TIMESTAMP
- DATE
- CLOB
- BLOB

More information about DOTS, DOTS test cases, Database schema, DOTS configuration file etc. can be found at:
http://ltp.sourceforge.net/dotshowto.php

3 Running Dots

To run Dots with Derby, main steps are as follows:

1. Install Dots, Derby and required softwares.

2. Create a Derby database to be used for test cases.

3. Set up configuration file needed during test run with various parameters. The brief description of each parameter is as follows:

Parameter	**Description**	**Default value**
DURATION	Duration a Dots test case will run in hours. The format is hh:mm	24:00
LOG_DIR	Directory where a DOTS test case stores its output, such as log files, error files, and summary files	*/usr/local/DOTS/Output*
CONCURRENT_CONNECTIONS	Number of concurrent database access connections that will be created by DOTS test case. This parameter is applicable only if paramater AUTO_MODE is set to “no”.	20
CPU_TARGET	Target level of CPU Utilization of Database Server while DOTS is running	90
AUTO_MODE	Parameter 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.	
SUMMARY_INTERVAL	Interval time between two writes of summary report to test summary file	30
UserID	User ID for database connection	
Password	Password for database connection	
DriverClass	DriverClass for database connection	
URL	URL for database connection	
SERVER_IP	Database server IP address	
SERVER_PORT	Port that performance monitor uses	
MAX_ROWS	Maximum rows a table can have	10,000
MAX_LOGFILESIZE	Maximum file size a log file can occupy	100M
CREATIONINTERVAL	Thread creation interval	3

During Dots testing, the configuration file was set up with the following parameters:
DURATION = 120:00
LOG_DIR = /home/derby//Dots/results_120hrs
CONCURRENT_CONNECTIONS= 30
CPU_TARGET = 95
AUTO_MODE = yes
SUMMARY_INTERVAL = 10
UserID = u1
Password = p1
SERVER_IP = 12.34.225.456
SERVER_PORT = 8001
MAX_ROWS= 500000
MAX_LOGFILESIZE = 10485760

CREATIONINTERVAL = 1
For Network server, the parameters DriverClass and URL were set as follows:
DriverClass = com.ibm.db2.jcc.DB2Driver
URL = jdbc:derby:net://12.34.225.456/TESTDB

For embedded database, the parameters DriverClass and URL were set as follows:
DriverClass = org.apache.derby.jdbc.EmbeddedDriver
URL = jdbc:derby:TESTDB

Note:
A dummy ip address is been used for parameters SERVER_IP and URL (Network Server).

4. Run Dots performance monitor and Dots test case as following:
   >java dots.perfmon.PerfMon -port *portnumber*
   >java dots.framework.Dots -config *config_filename* -case *testcase_name*

5. Review the results in log directory specified in configuration file.

More detailed information about Dots environment, installing Dots and Derby, creating Derby database, setting up Dots configuration file, and executing DOTS test cases, and reviewing the results can be found at:
http://incubator.apache.org/derby/DOTS_Derby.html

4 Issues/Tips/Hints

As part of initial testing, DOTS test cases were executed with Derby. Each test case or combination of test cases were executed for about 5 days. While running DOTS, some issues such as DOTS incompatibility with linux kernel, out of memory errors, and exceptions were found. This sections provides information about and/or solutions to those issues.

4.1 DOTS Code

DOTS is currently supported on Linux platform. As DOTS reports system level information such as CPU usage and memory usage, it accesses the system level files (/proc/meminfo, /proc/stat etc.) and gets CPU and memory usage information by parsing the system level files. As the format of system level files may vary depending on kernel version, the information in system files is not parsed properly by DOTS if linux kernel on your machine does not match with kernel version supported by DOTS code. Therefore the reported information such as CPU usage and memory usage is incorrect or is not reported at all.

I ran DOTS test cases on Linux kernel version 2.6.5-7.111.5-bigsmp. During testing, I found that summary file was printing memory usage as 0. After debugging the problem, I found that the format of file “/proc/meminfo” on my machine was not as expected by DOTS source code. To support format of file “/proc/meminfo” from my machine, I modified DOTS source code (src/dots/perfmon/PerfReader.java) and re-ran build to create new DOTS jar files with my changes. The original and modified code is as follows:

Original code:

```
public static int getMemory() {
  String memString;
  long totalMemory,usedMemory;

  try {
    /* Get memory usage */
    BufferedReader memReader = new BufferedReader(new FileReader(MEMINFO));
    memString = memReader.readLine();
    while (memString != null) {
      if (memString.startsWith("Mem:")) {
          StringTokenizer stk = new StringTokenizer(memString);
          if (stk.hasMoreTokens()) {
              memString = stk.nextToken();
              totalMemory = Long.parseLong(stk.nextToken());
              usedMemory = Long.parseLong(stk.nextToken());
              usedMemory = usedMemory/(1024*1024);
              return (int)usedMemory;
           }
        }
        memString = memReader.readLine();
     }
     memReader.close();
  } catch (IOException e) {
       System.out.println("PerfMon.PerfReader:" + e);
  }
  return 0;
}
```

Modified code:

```
public static int getMemory() {
   String memString;
   long totalMemory=0;
   long freeMemory=0;
   long usedMemory=0;

   try {
     /* Get memory usage */
     BufferedReader memReader = new BufferedReader(new FileReader(MEMINFO));
     memString = memReader.readLine();
     while (memString != null) {
       if (memString.startsWith("MemTotal:")) {
          StringTokenizer stk = new StringTokenizer(memString);
          if (stk.hasMoreTokens()) {
             memString = stk.nextToken();
             totalMemory = Long.parseLong(stk.nextToken());
          }
       }
       if (memString.startsWith("MemFree:")) {
           StringTokenizer stk1 = new StringTokenizer(memString);
           if (stk1.hasMoreTokens()) {
              memString = stk1.nextToken();
              freeMemory = Long.parseLong(stk1.nextToken());
           }
       }
       memString = memReader.readLine();
     }
     usedMemory = totalMemory-freeMemory;
     usedMemory = usedMemory/1024;
     memReader.close();
     return (int)usedMemory;
  } catch (IOException e) {
     System.out.println("PerfMon.PerfReader:" + e);
  }
  return 0;
}
```

The highlighted lines in original code were modified to highlighted lines in modified code. After deploying modified jar files, I was able to get correct memory informaiton.

4.2 Test Cases Duration

The duration of DOTS test cases can be specified in DOTS configuration file with parameter -DURATION. The limitation on duration of DOTS test cases is mainly machine memory. DOTS test cases keep on inserting data into the database which makes database keep on getting bigger during the test cases execution. If the memory on the machine is not enough to store the size of growing database, the test cases will fail with an out of memory error.

4.3 Java OutOfMemory Error

While running DOTS test cases with Derby, some instances of java.lang.OutOfMemoryError were found in the log file for DOTS Client**.** This is due to the fact that the jvm default memory was not big enough for the given test case. This problem was fixed by increasing the jvm memory by changing jvm configuration.

Example:
java -Xms400m -Xmx400m dots.framework.Dots -config config.ini -case BTCJ1

The same problems were found on network server side when test cases were executed with Derby network server.

Tip:
The jvm memory size will depend on how long the test cases will be executed for.
For example, to run DOTS basic and advanced test cases with Derby network server for 120 hours, jvm memory for DOTS client was increased to 400M and for Derby network server was increased to 720M. I gave memory of 400M and 720M to ensure that there is no memory problem after running it for few days. It is possible that the problem could be solved by setting jvm memory lower than 400M for client and 720M for Network Server.

4.4 Expected Exceptions

While running DOTS with Derby, the following exceptions were caught which after investigation were found to be expected exceptions. If you find any of the following exceptions while doing DOTS testing, you may ignore them.

The exceptions information is as follows:

4.4.1 Duplicate Key Exception with Embeded Database

Exception Output: *"SQL Exception: The statement was aborted because it would have caused a duplicate key value in a unique or primary key constraint or unique index*' "

This exception was caught while running DOTS with embedded derby database. This exception was found when DOTS test case was trying to write the same entry to the table of database where one of the datatype was declared as primary key.

4.4.2 Duplicate Key Exception with Network Server

Exception Output: *"com.ibm.db2.jcc.c.SqlException: DB2 SQL error: SQLCODE: -1, SQLSTATE: 23505. The statement was aborted because it would have caused a duplicate key value in a unique or primary key constraint or unique index*".

This exception was caught while running DOTS with derby database Network Server. This exception was found when DOTS test case was trying to write the same entry to the table of database where one of the datatype was declared as primary key.

4.4.3 Lock Timeout Exception

Exception Output: *"com.ibm.db2.jcc.c.SqlException: DB2 SQL error: SQLCODE: -1, SQLSTATE: 40XL1, SQLERRMC: 40XL1. A lock could not be obtained within the time requested".*

This exception was caught because Dots test case could not obtain lock within the time requested. This exception was thrown when cases (basic) were running at a time.

4.4.4 Deadlock Exception

Exception Output: "*com.ibm.db2.jcc.c.SqlException: DB2 SQL error: SQLCODE: -1, SQLSTATE: 40001, SQLERRMC: Lock : ROW. A lock could not be obtained due to a deadlock, cycle of locks".*

This exception was caught because DOTS test case could not obtain lock due to a deadlock, cycle of locks & waiters. This exception was thrown when multiple cases (basic) were running at time.

4.4.5 Batch Failure Exception

Exception Output: “*com.ibm.db2.jcc.c.vd: Non-atomic batch failure. The batch was submitted, but at least one exception occurred on an individual member of the batch. Use getNextException() to retrieve the exceptions for specific batched elements.*”

This exception was caught when Dots test case sent SQL statements as a batch/unit. One or more of the statement was with duplicate primary key.

5 Summary

In this document, a brief overview of DOTS has been given to get started with DOTS. Most of the information can be found from the DOTS web site (http://ltp.sourceforge.net/dotshowto.php#SEC11) and DOT and Derby paper (http://incubator.apache.org/derby/DOTS_Derby.html). This paper mainly described some of the issues/problems that were found during initial system testing of Derby with DOTS.

6 References

[1] Sunitha Kambhampati, *Running DOTS with Derby*
http://incubator.apache.org/derby/DOTS_Derby.html

[2] David Barrera, *Database Opensource Test Suite User's Guide*
http://ltp.sourceforge.net/dotshowto.php#SEC11