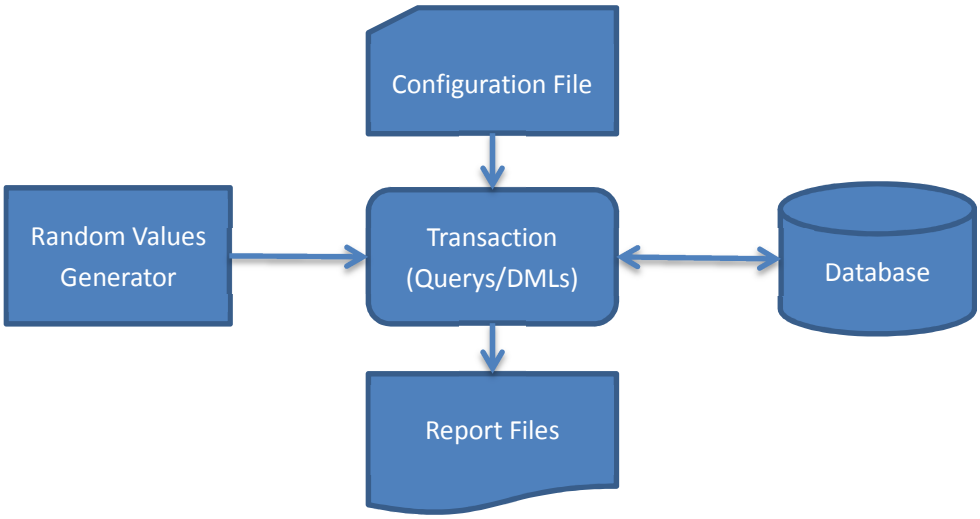
Database Response Time Testing Utility

In the past five years, I am managing the world's biggest database system for online payment service (AliPay of Alibaba Group), it handles 100 million trades on 2012/11/11, totally 4 billion database transaction, 28.5 billion SQL executions, 193 billion memory data block touches, 15 terabytes database log files generated at that day. So for every database (including Oracle and MySQL), I have to know how much business it can afford to avoid system crash at peak time, the peak time's load is about 6 times higher than normal time duration, I spend a lot of time on database testing for different hardware configuration, such as SDD or Fusion IO device. And I also write my own database testing utility because I cannot find a tool good enough for me in this case.

The most important thing is the abstraction of the business data model and testing it, and I want to know the response time curve at different database load situation. The test utility enable you create your own business tables, and generate the random data, and performing the business logic testing, and generate a detailed response time report for the business logic. According to the response time requirement we summarized from the application server performance log, we will know how many server we required exactly for the peak time, without obvious resource wastage.

The testing utility is mainly focus on the transaction process. Following is the design diagram I draw three years ago.



I will explain every section, after that you will be able to use the testing utility for performance testing and capacity evaluation of given database and hardware comination.

RANDOM VALUE GENERATOR

You can define external dynamic variables (compared to the database system), and define the data range for automatically value generation, you can reference the variable by name (prefix with a colon char) in SQL statement both for Oracle and MySQL, as following:

INSERT INTO <table name> (…) VALUES (:varname, :varname,…);

*SELECT * FROM <table name> WHERE … > :varname;*

There are 7 different variable types you can choose, the variable definition syntax listed as following.

varname VARTYPE minimum maximum

The valid variable types including “SEQ”, “INT”, “CHAR”, “FLOAT”, “DOUBLE”, “DATE”, “TIMESTAMP”. I will explain different types to you.

- SEQ

  Integer value, auto incremental, minimum is the start value, maximum is the end value, if the execution loops large than the range, it will return to the start value again, like a circle Oracle sequence.

- INT

  Integer value, auto generated value, minimum is the minimum value, maximum is the maximum value, the value auto generated will be in the range of minimum value and the maximum value.

- CHAR

String value, auto generated value, minimum is the minimum length of the string, maximum is the maximum length of the string value, all characters will be random selected between ‘A’ and ‘Z’, the maximum length is 255 characters.

- FLOAT

  Float value, auto generated value, minimum is the minimum value, maximum is the maximum value, the value auto generated will be in the range of minimum value and the maximum value.

- DOUBLE

  Double value, auto generated value, minimum is the minimum value, maximum is the maximum value, the value auto generated will be in the range of minimum value and the maximum value.

- DATE

  Sting value with date format (YYYY-MM-DD), auto generated value, minimum is the days between the start day and today, maximum is the days between today and the end days. For example, minimum “-10” means 10 days ago, and “10” means 10 days later.

- TIMESTAMP

  Sting value with date format (YYYY-MM-DD HH24:MI:SS), auto generated value, minimum is the days between the start day and today, maximum is the days between today and the end days. For example, minimum “-10” means 10 days ago, and “10” means 10 days later.

You can use different variable types for random value generation with specific data distributions, it will give you exact performance data without pull down the production database data.

CONFIGURATION FILE

The configuration file tell the testing utility everything except the parallel degrees, it’s a text file contains the following sections, the bold words is the key words reserved for

specific purpose.

OPTION

- Options

DECLARE

- Variables Definition

BEGIN

- SQL Statements

END

Options Section

The options section contains 8 properties of the testing.

- user

  Tell the testing utility how to connect to the database with 5 different properties by a formatted string (username/password@hostname:port:database), both Oracle and MySQL use the same pattern, if you are a DBA, you should be familiar with it.

- loop

  Specify how many times to be executed for the SQL statements, default value is 100000000 times.

- name

  The testing name, it will appear in the log file, just a name to identify the testing case.

- wait

  The sleep time in microseconds during each loop, not sleep time between different SQL statement, default value is 0, means no sleep time.

- log

The output log file name pattern, you can use “%p” to represent the parallel slave id to create a log file for each parallel testing process.

- show

  The interval time to report the response time during a long run testing, default is 60, means it will write the response time to log file at every minute.

- tran

  Transaction mode, default is off, which mean every SQL statement is committed after a success execute. If you enable it by set this option to “yes”, all SQL statements will be execute in one transaction for every loop.

- commit

  The commit size of the testing, default value is 1, it will issue an explicit commit command after each execution loop, since we are testing the business logic, and all the SQL statements should be in one transaction. If you are generating large volume of testing data, you can set it to 100 or 1000 or higher value to improve the speed. This option is only value when you set option “tran” to “yes”.

- time

  Specify how duration to be executed for the SQL statements, default value is 3600 seconds, you can specify “d1” for one day, “h1” for on hour, “m1” for one minute, and “s60” for 60 seconds.

A complete example of options section.

```
option
   user <user>/<pass>@<ip>:<port>:<dbname>
   loop 500000
   log   select_userview_16k_%p.log
   name Random_Userview_Select
declare
```

Variables Definition

The variables definition section contains the random value generator, you must define every variable you referred in all the SQL statements. The variables will generate random value before each loop, not before each SQL execution, unless you have just one SQL statement in the testing case, which means if you referred the same variable in different SQL statements it will use the same value when executing. Please refer the random value generator chapter for the

A complete example of variables definition section.

declare

uid1 int 10000000 11000000

uid2 int 10000000 11000000

uid3 int 10000000 11000000

uid4 int 10000000 11000000

uid5 int 10000000 11000000

If your SQL statements referred lots' of variables, you must define very variable referred, else it will be treated as NULL values.

SQL Statements

The SQL statements section contains all the SQL query including SELECT, INSERT, UPDATE and DELETE statement. Or database scripts quoted by "{" and "}". Every SQL statement should be terminated by semicolon(";").

begin

select col1 from T_KC_CENTER where col1 = :id;

select col1, col2 from T_KC_CENTER where col1 = :id;

*select * from T_KC_CENTER where col1 = :id;*

```
   { begin update t_kc_center set col2=col2 – 1 where col1 = :id;
          update t_kc_center set col2=col2 + 1 where col1 = :id + 1; end; }
end;
```

If you want a SQL statement to be executed by a random percent, you could prefix the SQL statement with “RANDOM n ” pattern, where n is a number between 1 and 100.

```
begin
   select col1 from T_KC_CENTER where col1 = :id;
   RANDOM 50 select col1, col2 from T_KC_CENTER where col1 = :id;
   RANDOM 20 select * from T_KC_CENTER where col1 = :id;
end
```

If you want to fetch an query result (usually one row returned) into variable, just prefix the SQL statement with “SETVAR “ pattern, it will put the query result into the variable with the same name as column alias.

```
begin
   SETVAR select dbms_random.value ID from dual;
   select col1, col2 from T_KC_CENTER where col1 = :id;
   select * from T_KC_CENTER where col1 = :id;
end
```

Here is an example of the configuration file I used for MySQL testing, the MySQL server and the testing utility is running on the same machine.

```
option
   name mysql_test
   loop 2000
   user /@::test
```

```
declare
   a int 20000 30000
   b int 20000 30000
begin
   select * from t_mytest where col1 = :a;
   random 50 select * from t_mytest where col1 = :b;
end
```

RUN TESTING CASE

The parallel degree is not specified in the configuration file, it's a command line option of the testing utility as following, the "query" option is the file of the testing case.

```
./oradbtest_linux64.bin    query=<test case file> degree=8
```

The maximum parallel degree you can specify is 128.

REPORT FILE

The output file is a text file contains detail response time information, and the execution times, affected rows or fetched rows of the SQL statements. Let's take a look at the following example.

```
ORADBTEST: Oracle Database Test Utility , Release 1.0.1
(@) Copyright Lou Fangxin (AnySQL.net) 2012 - 2013, all rights reserved.
======================= Oracle_Test =======================
SQL01    exe=5000 row=4999 ela=7570 ms avg=1514 us
SQL01       2 ms    exec=       4756, ela=          7018 ms, avg=       1475 us, pct= 95, 95
SQL01       3 ms    exec=        242, ela=           545 ms, avg=       2255 us, pct=    4, 99
SQL01       4 ms    exec=          2, ela=             6 ms, avg=       3253 us, pct=    0,100
Total    tran=5000=657/s, qtps=5000=657/s, ela=7573 ms, avg=1514 us
```

From the above output, we can see that by average (The total line) the requests is

completed in 1.5 ms (1514 us), the transaction per second is 657 for a single thread, if you want to get the total TPS, please multiply the degree, I use 8 in this testing, so the total TPS is about 5200.

Then you can check every SQL statements, the “SQL01” mean the first SQL in the test case (“SQL02” will be the second SQL statement in the test case, and so on), about 95 percent of the requests are returned within 2 ms, and another 4 percent of the requests are returned within 3 ms, and two times with 4 ms.

The testing utility does not gather database load information, please use other database performance tuning utility to capture the database load data during testing.

GET SOFTWARE

For MySQL Database
http://www.mydul.net/software/mydbtest_linux32.zip
http://www.mydul.net/software/mydbtest_linux64.zip
For Oracle Database
http://www.mydul.net/software/oradbtest.zip

ABOUT ME

My name is Fangxin Lou, Oracle ACE, about 15 years Oracle DBA career life. The author of Oracle data recovery utiltiy (AUL, also named MyDUL). You can get touch with me by skype (anysql) or gmail (anysql@gmail.com).

Thanks!