l e x a n a \ n e t

March 2007

A Database Load Generation Utility

By Drew Hamre

email

Database load generators are software tools that generate workloads against a target DBMS. These tools emulate user activity against a database server, allowing you to:

- *Assess the behavior of operational components when running under a simulated workload*. For example, the tool could create a load on a server so as to understand cluster failover behavior, to create conditions for exercising operational software, or for evaluating performance monitoring tools.
- *Measure the throughput of a particular database server*. When properly implemented, load generators can be used to create benchmarks of transactional throughput for a particular database server configuration (for example, with different levels of memory expansion or various I/O configurations). Particular benchmarks, including the TPC-C transactional standard, can be viewed as rigorously-specified, load generation protocols.

This paper presents a custom load generation utility, including source code. This database load generator (DBLoadGN) consists of two components: 1) a Windows desktop client program that drives an OLTP workload by emulating a variable number of 'virtual' users; and 2) a custom companion database that optionally serves as the target of the users' transactions. The desktop application is written in Visual Basic .NET (Framework v2.0/VS2005). The companion database can be implemented in either SQL Server 2005 or Oracle 10g (though this paper will focus on SQL Server operations).

Load generation tools

Many excellent load generators are available, though they vary dramatically in complexity and cost. Commercial products are the most sophisticated, and include tools such as Quest's Benchmark Factory and Mercury's LoadRunner.

Commercial load generators may be expensive, especially where the vendor charges according to the number of emulated users. For testing protocols like TPC-C, these licensing costs may be prohibitive because of the many 'users' needed to fully saturate a modern database server (due to protocol-mandated wait states, or 'think times'). For example, a recent benchmark (4-way 2.6GHz dual-core Opteron) engaged 171,360 emulated users while attaining 213,986 tpmC.

Simpler, less expensive load generators are available from RDBMS vendors. For example, Microsoft's SQL Server 2000 Resource Kit includes the utility, 'Database Hammer' (a replacement for SQL Load Simulator). Database Hammer is a VB6 application (and thus shows its age)[1], but its design strategy – wherein a desktop application spawns multiple objects, each interacting with a target database, is common to many other tools, including DBLoadGn.

Microsoft also offers SQLIOStress - a utility that measures a system's I/O capacity – in addition to a simpler variant, 'SQLIO'. SQLIOStress (formerly SQL70IOStress) includes updates for SQL2005 but doesn't use (or require) SQL Server's database engine. Rather, the tool *emulates* the types of I/O that SQL Server generates. This emulation is extremely sophisticated, and the SQLIOStress documentation provides a rare public glimpse into the low-level internal details of the database's I/O strategies.

Shareware is another source of database load generators, including Alberto Venditti's DBStressUtil, published via the CodeProject. This utility, which can be characterized as a .NET (V1.1) update of Database Hammer, includes improvements beyond the modernized run-time environment. For example, DBStressUtil de-couples the load generator from the target database. Rather than hard-coding SQL for a specific test database (as in Database Hammer), DBStressUtil executes arbitrary external SQL scripts that can target any database. It includes a default script that targets the venerable SQL Server sample, *NorthWind*.

Academic websites also offer load generators, including implementations of the TPC-C standard (here and here). Note these TPC-C implementations are Linux-centric, complex, and admittedly rough (and therefore unlikely to generate sanctioned results).

Against these many alternatives, DBLoadGn offers a mix of benefits: it's free, source code and a custom transactional workload are included, it's written for Microsoft's latest .NET runtime, it's simple to install and use, it supports both SQL Server and Oracle on both x32 and x64, and it handles difficult operational scenarios such as cluster failovers. However (as with other non-commercial packages), DBLoadGn is capable only of generating *private* benchmarks, not *public*.

Public versus private benchmarks

Any repeatable workload can provide a benchmark of computing performance. In fact, if large datasets are available, even repeated backup operations can provide valuable metrics. However, not all benchmarks are equally sensitive to system differences, nor are all benchmarks applicable across multiple vendors or clients. In this context we distinguish *public* benchmarks (such as TPC-C, SPECint, Linpack, and others) from *private* benchmarks (including results from DBLoadGn).

Public benchmark protocols are designed and monitored by industry and academic groups. Because such tests are recognized by multiple vendors, the benchmarks provide valuable cross-platform performance comparisons and help assess solution costs. Because the algorithms are mature and exhaustively reviewed, the

---

[1] Microsoft hasn't released a SQL2005-equivalent of 'Database Hammer', but the old utility can be used against the newer DBMS (though limited to legacy connection objects and the VB6 runtime environment). For the .NET environment, Microsoft offers Application Center Test which simulates loads against web applications (and thus indirectly places loads on back-end databases, as did an earlier utility – Microsoft's WAS (Web Application Stress) tool).

benchmarks are sensitive to a broad range of performance influences. However, this complexity and maturity comes at a price, and the only programs capable of generating auditable transactional benchmarks tend to be high-end commercial packages (such as LoadRunner) or mature proprietary software from the DBMS or hardware vendor. In addition, auditable tests require experienced specialists.

Private benchmarks (such as those created by DBLoadGn or by a dataset backup) may be an appealing alternative if *relative* performance is the concern (for example, measuring throughput on a 4-gigabyte system, and then again after expanding to 16-gigabytes). Private benchmarks allow the comparison of different server models and I/O configurations, and can quantify the impact of a design change. They may be configured with representative transactional workloads, and their simplicity is well suited to repeated use by operational staff. However, private benchmark results may be compared only to results obtained from the identical toolset; comparisons can't be made to results from other tools or to results from other test protocols.

For a deeper appreciation of the complexity of mature public benchmarks, refer to the latest TPC-C Standard Specification document, or to Jan Mulder's recent TPC-C 'primer'.

An overview of DBLoadGn

DBLoadGn adopts an approach similar to *Database Hammer* and *DBStressUtil* (above). The chief components of DBLoadGn are its client program and an optional companion database. The client program reads a batch of database commands, and then spawns a set of threads, each of which loops through the command batch, repeatedly executing the commands against the target database. See Figure 1.

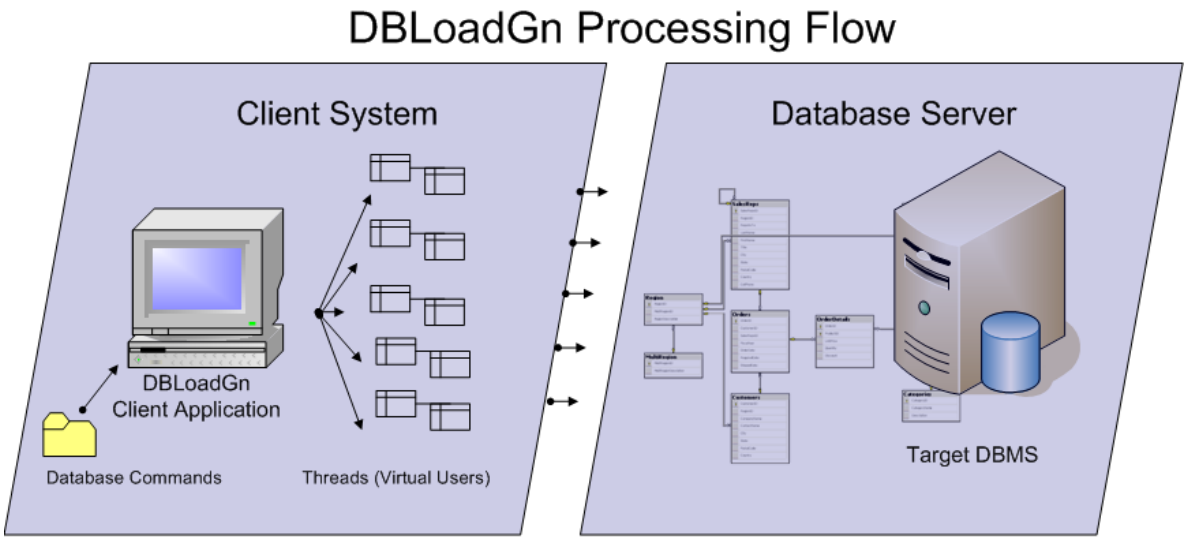

Figure 1: Overview of DBLoadGn Processing

Each background thread (and there may be many hundreds) emulates an interactive database user who is issuing database commands. These commands may be either simple one-line SQL statements or may invoke complex stored procedures.

To increase the load created by DBLoadGn and help drive server saturation, the 'think time' between each command can be shortened, or the number of 'virtual users' may be increased (and the latter is the preferred method.) The client program

interface is typically used by a DBA or test administrator, and is shown below in Figure 2.

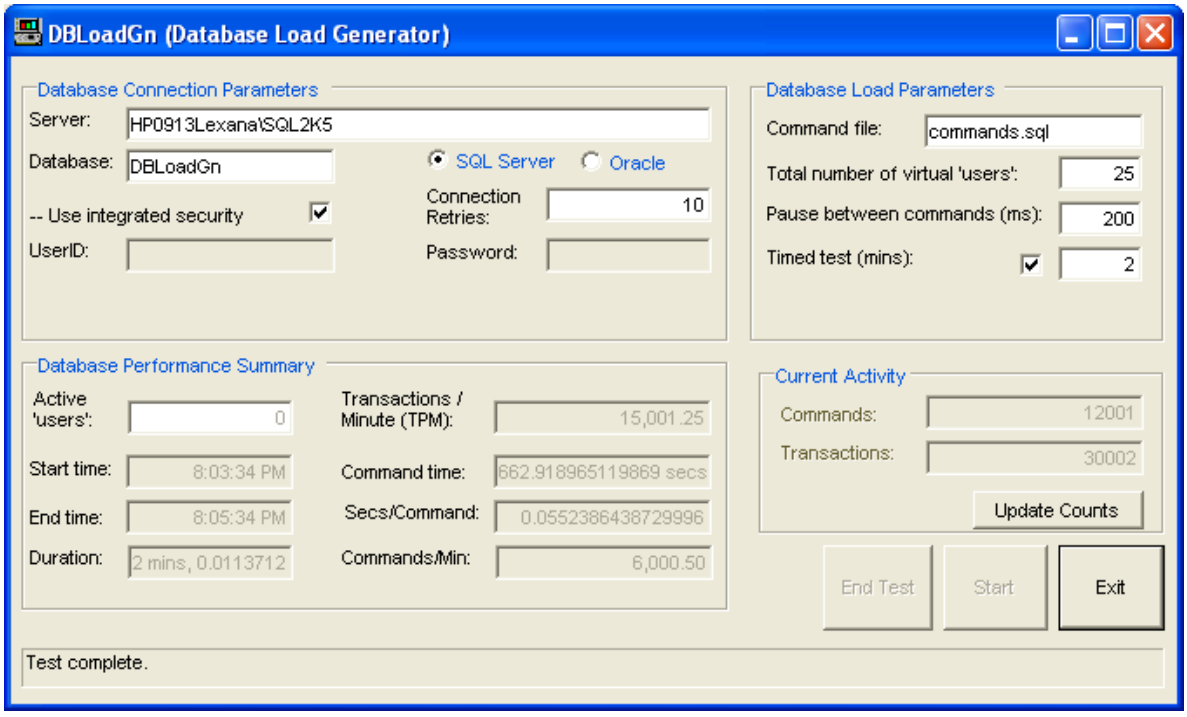

Figure 2: DBLoadGn After 2-Minute Timed Demonstration

The commands issued by DBLoadGn can target any accessible database. However, DBLoadGn's default workload runs against a custom companion database that is created on the target server specifically to support DBLoadGn testing. This database is modeled after a typical transactional 'order entry' design.

The default workload scenario is implemented as a collection of stored procedures, which are called by the client program. These procedures loosely mimic the TPC-C transaction profile, emulating an order-entry system with a percentage mix of transaction types as follows:

- Insert new order (roughly 40%) -- The new order transaction consists of inserting an order, along with a varying number of corresponding order detail records.
- Update quantity (roughly 30%) – For a randomly-selected order, this transaction updates the quantity value on all associated order detail records.
- Get order value, get category summary, and other lookup/reporting operations (roughly 30%) These query operations include retrievals to a) get the value of an order (sum across order detail records for a particular randomly-generated order ID); b) get the total quantity for a given product category name; and c) get total sales for a given fiscal year

The schema for the custom companion database is shown in Figure 3 below:

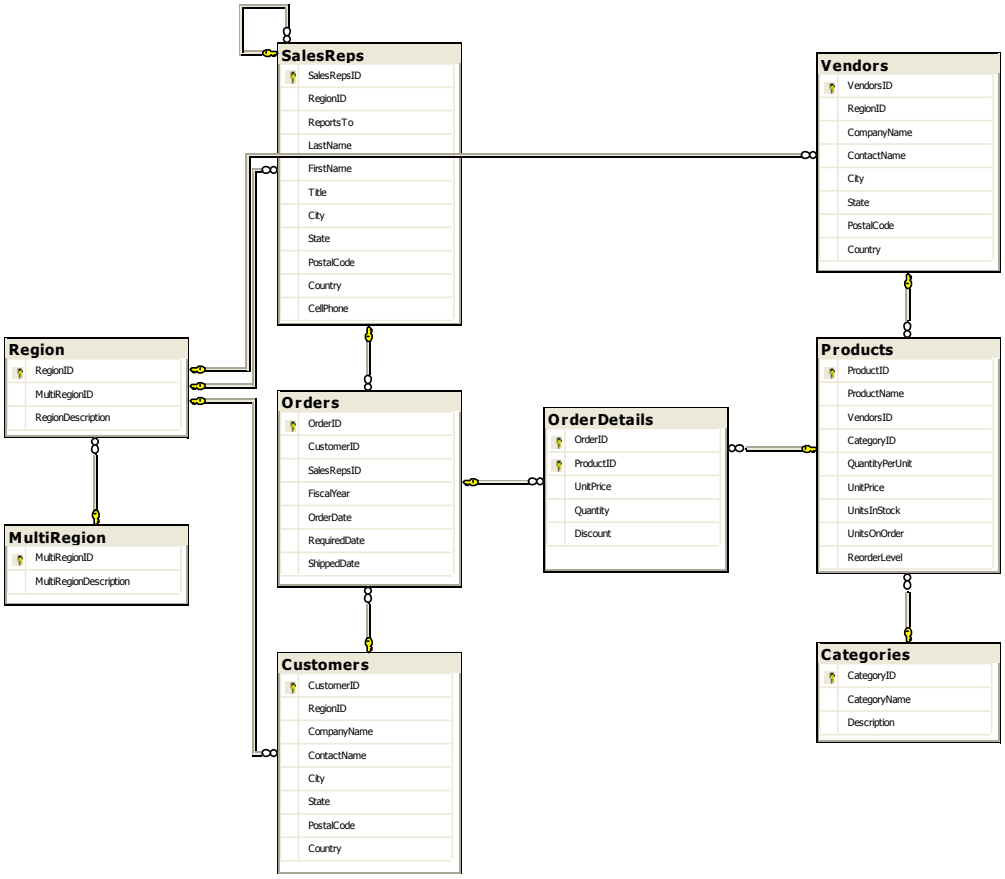

Figure 3: DBLoadGn Companion Database Schema

Installing the DBLoadGn client software

The DBLoadGn desktop client software runs under either Windows XP or Windows Server 2003 (x86 or x64[2]). Microsoft's .NET Framework 2.0 Data Provider is used for both SQL Server and Oracle. For full Oracle functionality, Oracle client software must also be loaded on the DBLoadGn system. (At the time of testing, Oracle's ODP.NET managed provider couldn't be used due to reliability issues on x64.[3])

To load the DBLoadGn client software, copy DBLoadGn.exe and the associated default command files (*commands.ora* and *commands.sql*) to the target system as shown in Figure 4. The command files must be copied into the same directory as the .exe file. All files are included toward the end of this paper.

---

[2] When compiling *DBLoadGn* to run under Windows Server 2003 (x64), change VS2005's target CPU from the default (*AnyCPU*) to *x86* to avoid runtime errors when linking to the Oracle client libraries.

[3] The Microsoft interfaces require *System.Data.OracleClient.dll* and import *System.Data.OracleClient.* By contrast, the Oracle interfaces require *oracle.dataaccess.dll* and import *Oracle.DataAccess.Client.* All Oracle tests with DBLoadGn used version 10.2.0.2.20 of Oracle's client libraries for Windows, and ran against remote Oracle 10g databases (both RAC and non-RAC) running under SUSE Linux 9/SP3 (x64).

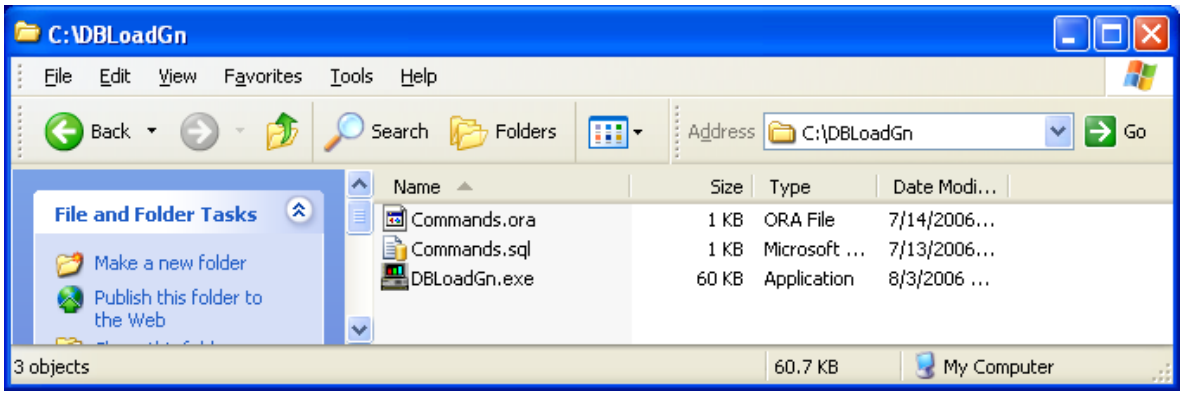

Figure 4: DBLoadGn's Installation Directory

The DBLoadGn client program can target any database to which it can connect, either locally (where the database is on the same system as the client software) or across a network. When benchmarking, the DBLoadGn client should be executed from a remote system rather than on the database server's system. This gives the DBMS full access to the server's resources, without competition from the emulated clients.

Installing the DBLoadGn companion database

The DBLoadGn companion database is created by running load scripts on the target database server. The scripts perform the following tasks in sequence:

- Create the test database and its tables.
- Create stored procedures and constraints.
- Create indices to improve performance.
- Populate the lookup tables with initial data. Note that the *Orders* and *OrderDetails* tables won't have any records added during installation; instead, records will be inserted here during DBLoadGn test runs.

DBLoadGn processing flow

DBLoadGn consists of two VB.NET modules, 1) a Windows form that implements the application UI and supporting logic, and 2) a class module that implements the virtual user. To run a benchmark, testers enter runtime parameters and click 'Run', activating a startup timer that fires periodically, each time instantiating a new virtual user. The timer de-activates itself once the requested number of users are created.

Each of the newly-created virtual user objects loops through a batch of database commands. For performance tracking purposes, DBLoadGn accumulates the elapsed time spent processing each command using high-resolution performance counters.

DBLoadGn will run for an indefinite period, either until manually stopped, or - for timed tests - until the requested test interval has ended.

DBLoadGn activity reports

At the end of a test, DBLoadGn summarizes the results in its client interface, including the following measures (see Figure 2, above):

'Active Users' – This count reflects the current number of active threads ('virtual users'). The count is initially zero, but is updated dynamically once the test run begins. It shows the progress in creating the requested number of background threads. It drops to zero as the test ends.

'Update Counts' – This command button can be clicked during *ad hoc* tests. Once clicked the main program will iterate through each background thread, and calculate the current number of successfully completed commands and transactions.

Database Performance Summary – At the completion of a test run, DBLoadGn will echo additional summary information:

- *Start Time, End Time,* and *Duration* show the overall span (wall-clock) of the test
- *Command Time* is updated to show the total wait time across all threads. This is calculated as (command_completion_time – command_start_time) and summed for all commands, and then accumulated across all threads. Because of the number of threads, and because this reflects wall-clock time, the total Command Time may be a larger interval of time than Test Duration. The average time to complete any single command is reported as *Secs/Command*.
- *Transactions/Minute* is a key summary statistic, but is available only if the default companion command files and database are used. If this default workload is used, then DBLoadGn has enough information to convert the raw commands/minute rate into a measure of completed transactions/minute. Note again that this metric relates only to DBLoadGn, and is not equivalent to or expressible as the tpmC standard.

Performance and Error Logs - DBLoadGn emits a single CSV-formatted performance log for each run; it's written to the same directory as the exe, and its name is date-stamped to allow multiple logs to be stored in the same location. In addition to the performance log, each 'virtual user' thread may create an error log file, should they encounter an exception. The error logs' filenames are based on the performance log name, with the thread's sequence number appended (e.g., '061013-175240.csv' might be accompanied by '061013-175240-err-35.log').

Using DBLoadGn to drive system load

DBLoadGn can be used to create operational loads on a system. For example, it can:

- Emulate user activity during cluster failover (shown below)
- Emulate user conditions for testing 'hot' backups or disaster recovery
- Drive extreme system loads so as to test SLA alert thresholds

In Figure 5 (below), DBLoadGn was used to push transactions against a virtualized SQL Server instance running on a two-node PS cluster. When the active node (Server01, the red line) was disabled, PS moved the running database instance to the second node.

After a brief interruption during failover, DBLoadGn transactions resumed on Server02 as indicated by a jump in Server02's processor activity (blue line). After a minute, the SQL Server instance was returned to its original node via PS's Dynamic Re-Hosting. Note that because the DBLoadGn desktop client connected to SQL Server via a virtualized server address (courtesy of DNS and PS), the application's connection string remained unchanged throughout both failover and fallback.

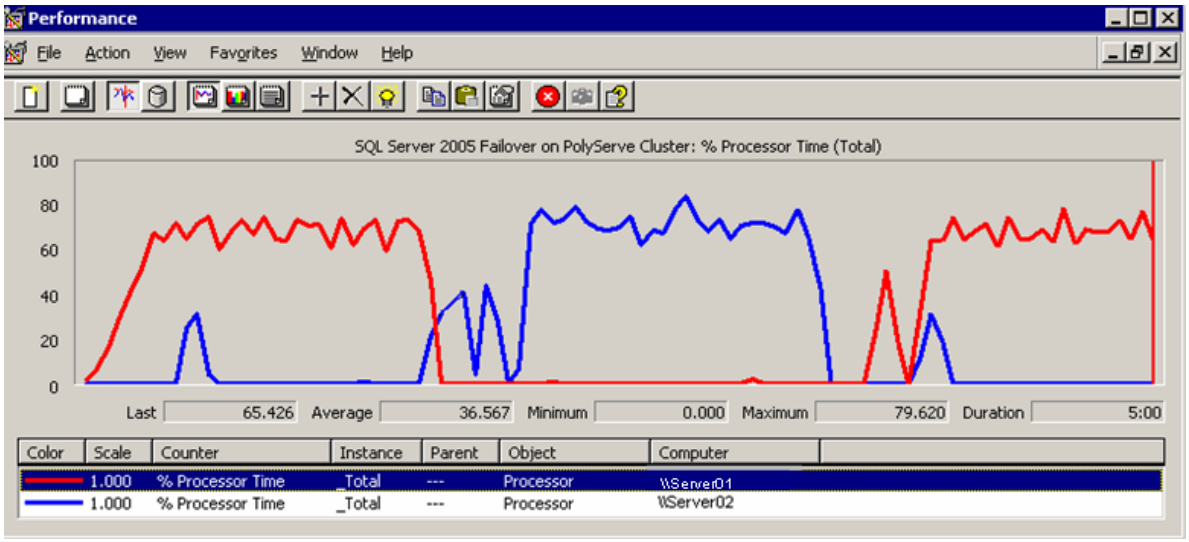

Figure 5: DBLoadGn transactions during cluster failover / fallback

Note that both database instance moves (first to the standby node, then back to the original) occurred comfortably within a five-minute test window – an impressive performance given the high transaction rates (> 100,000 TPM) being recorded.

Although DBLoadGn is able to remain 'alive' during cluster failover, it requires heavy-handed error-trapping to do so – primarily because .NET connection pools often become corrupt after a connection failure. To help make retries more graceful, Microsoft is enhancing their software and adding new language operators (*ClearPool* and *ClearAllPools*), releasing an operating system hotfix, and (for database mirroring) adding new connection parameters (*Failover Partner*=;).

Using DBLoadGn to measure system performance

DBLoadGn can be used to generate *private* benchmarks of system performance. To do so, run DBLoadGn repeatedly with increasing numbers of virtual users (and therefore, increasing transaction requests). Increase the user count until the server becomes saturated and the transaction curve flattens as the server has no capacity for additional work. The number of completed transactions/minute at this saturation point is DBLoadGn's estimated private benchmark of maximum throughput.

The results below contrast performance for SQL Server 2005 across samples of three classes of hardware: laptop, desktop, and server.

In these examples, DBLoadGn was run repeatedly on a laptop, first with 25 virtual users (attaining 15.7K TPM), then with 50 users (26.6K TPM), then with 100 users (33.4K TPM), and finally with 150 users (32.6K TPM). The estimated maximum DBLoadGn TPM rating for the system is therefore roughly 33,000. Note also that once saturation is reached for the system, performance typically degrades slightly for the next higher test increment, as system inefficiencies appear under stress. See Figure 6.

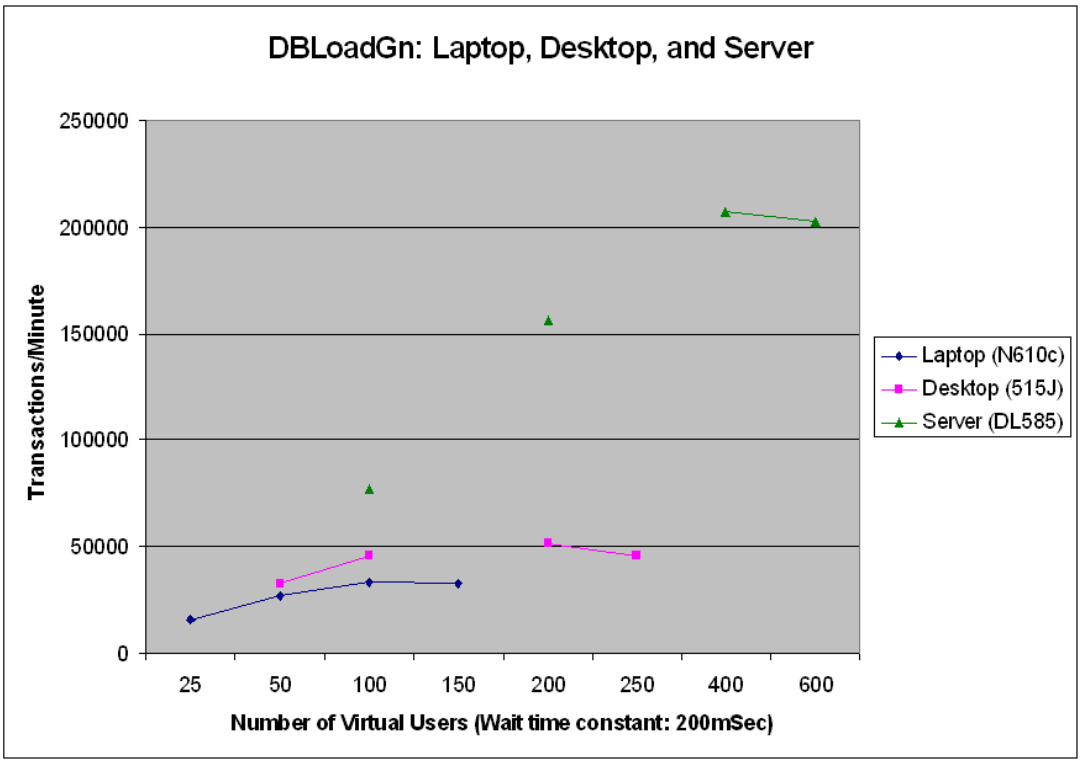

Figure 6: DBLoadGn and Relative System Performance

Similar tests on more powerful hardware yielded correspondingly faster results, peaking at roughly 51,000 TPM for a desktop system and roughly 207,000 for a DL585. Note that DBLoadGn needs far fewer 'users' to generate these transaction rates than corresponding TPC-C software because DBLoadGn's 'think times' (inter-command wait intervals) are nearly nonexistent (whereas TPC-C think-times are lengthy so as to emulate 'actual' user behavior). In addition, TPC-C transactions are far more complex than DBLoadGN's.

DBLoadGn limitations and enhancements

DBLoadGn is limited to a maximum of slightly more than 1400 virtual users per active instance of the desktop program. Planned enhancements include improved reporting, a re-scaled workload (more complex transactions with larger row sizes), and an automated 'seek' function for performance maximums.

Additional Resources

How to Set Up a SQL Server Stress Test Environment in 8 Steps by Geert Vanhove
http://www.sql-server-performance.com/gv_stress_test_lessons_3.asp

Acknowledgement

Thanks to reviewers for their time and suggestions; any inaccuracies remain mine.

Summary

This paper describes a customizable database load generator with two components: 1) a Windows desktop program that drives an OLTP workload by emulating a variable number of 'virtual' users; and 2) a companion database that serves as the target of the virtual users' transactions. The tool generates transactional workloads to measure relative system performance, or to help perform operational assessments.

Drew Hamre is a principal of lexana\net and lives in Golden Valley, Minnesota.