SQL Testing through Scalability with Respect to Concurrent User Count: A Simulation Recipe

Boris Dali
Dali Database Solutions Inc.
boris@borisdali.com

This paper suggests a practical approach for *automating* the testing of scalability with respect to concurrent user count of SQL queries and PL/SQL stored code. The approach is based on a simulation of concurrent user load with exponential interarrival times to maximize its resemblance to real-life application workloads. Drawing upon the previous research and practical experience, the article presents the simulation program called **SSS-test** for **S**QL **S**calability **S**imulation and demonstrates its application. The task of forecasting performance under a requested concurrency load is solved through analytical modeling and through simulation. The comparison of both approaches is provided to highlight the singular advantages of simulation.

Categories and Subject Descriptors: H.2 [**Database Management**]: Systems–*Concurrency; Query Processing*; D4.8 [**Operating Systems**]: Performance–*Simulation; Modeling and Prediction; Queueing Theory*
General Terms: Database, SQL, Scalability, Performance, Testing

---

Introduction

This study was prompted by the following frequently-encountered situation.

> Development Leader: "We have finished developing a view, which performs well in a unit-testing/single-user environment, providing the adequate response time of 3 seconds. Could you tell us whether or not a response time below 10 seconds can be expected when this query is executed by 15 concurrent users in our production system in accordance with SLA?"

In this scenario,[1] a DBA familiar with the queueing theory and spreadsheet-based capacity planning tools for Oracle systems can make an *optimistic* forecast, which, while being an excellent initial *viability assessment*, may not be precise enough to answer the question asked. This is partly due to *out-of-the-model* scalability issues[2]; partly due to activities unrelated to the modeled query, which take place in a real-life system and result in additional workload and 'drain' of resources; and partly due to specific deployment factors, hard to account for mathematically. In addition, it is beneficial *to visualize* the outcome of the queueing theory calculations, and *to extend* forecasting performance beyond a mere query onto a set of *competing* business processes. The aforementioned questions prompted this study in the area of *scalability with respect to concurrent user count* (henceforth SCUC) in an attempt to find an alternative to modeling. The alternative presented in this paper is the *automated* benchmark-simulation (henceforth simulation).

Background

The subject of scalability, as it applies to Oracle systems in general and two aspects of scalability—scalability with respect to data volume and SCUC— in particular, are best explained in an excellent and concise article by Cary Millsap, published in 2001 (Millsap, 2001). In practice, even in 2006, a good SQL testing process, going beyond simple functionality testing is hard to come across. Even when there is one, it is usually restricted to limited performance testing, or at best includes some scalability testing with respect to data volume, possibly, scalability testing with respect to symmetric multiprocessing (SMP) order for scale-up possibilities, and/or multi-node testing for scale-out possibilities. SCUC testing is rare and usually limited to cases in which: 1) the serialization is known (e.g, dbms_lock or serializable transactions are used) or 2) large-scale application benchmarks with high-end load testing tools (e.g., Mercury's LoadRunner or Compuware's QALoad) take place. However, I am not aware of a simple ten-minute free-of-charge simulation tool for routine daily testing that would address concerns of SCUC at a query-level, let alone at a business-process-level.

In fact, the term 'scalable SQL' usually designates scalability with respect to a data-related aspect, mostly volume and sometimes distribution. When most people talk about 'scalable SQL' they do not even specify what kind of scalability they have in mind, referring exclusively to scalability *with respect to data volume*. It is conventionally agreed that scalable SQL is the one that demonstrates consistent response time when the amount of data in the underlying data sources begins to grow.[3] The question arises, however, whether the number of concurrent users (e.g., 1 vs. 15 concurrent users slated to execute a specific query) affects the response time? Does the concurrency requirement need to be taken into account when writing, tuning, and testing SQL? After all, this is what is witnessed repeatedly during benchmarks: 'offensive' SQL floats on top; it is taken to a single-user environment

[1] In another, unfortunately frequently seen variation of this scenario, a DBA is casually asked to approve a change request to promote a new functionality-extended version of the SQL to the production system, while the request is often accompanied with an innocuous remark: "It will not impair performance, will it?" Contained herein is one example of such a scenario: http://www.freelists.org/archives/oracle-l/02-2006/msg00498.html .

[2] For example, analytical modeling does not take into account Oracle latching. Latching cannot be correctly represented by an M/M/m/infinity/FCFS queueing theory model because it violates the FCFS assumption (Millsap & Holt, 2003, p. 244).

3 There is also a strong theoretical foundation for scalability with respect to data volume. One example is an O(f(n)) notation for an "asymptotic time complexity" function used in the field of algorithm analysis to denote "limiting behaviour of the complexity as size increases," whereas "time complexity" is observed to be "an algorithm expressed as a function of the size of a problem." (Aho, 1974, p. 2).

and tuned. This may make little difference for a single user response time, yet may have a profound effect when the benchmark is repeated with the tuned SQL in place.

It is widely acknowledged that the mixed blessing of SQL is in its flexibility.[4] In other words, an experienced developer should think of more than one way of translating business requirements into SQL statements. Created alternative SQL statements are, therefore, to be tested in order to evaluate them and to choose the best. Let us assume that two SQL statements are created, an original and a tuned,[5] identical with regard to functionality (i.e., producing the same result set), which, during the testing under a single-user load, manifest similar behaviors. Nevertheless, under a concurrent load, the above statements may behave differently, which by extension will determine a preference of one (tuned) statement over another (original). In order to make this choice an informed and judicial one, the performance and scalability requirements should be properly stated, [6] and simple SCUC testing should be conducted.

The difference between an original SQL statement using a full-table scan (FTS) access path and a tuned SQL using an index-range scan (IRS) may not be as significant from the SCUC perspective as it is from the perspective of scalability with respect to data volume.[7] However, ignoring this difference means leaving it up to the users to find out whether or not the given SQL statement can sustain the concurrent-user load while maintaining adequate response time.

Simulation Approach – SQL Scalability Simulation (SSS)-test

To revisit the original question of the development team leader: there is a query with a response time of 3 seconds for a single-user, and you are asked whether or not it will still provide a response time of under 10 seconds for 15 concurrent users. How can this be approached?

Craig Shallahamer distinguishes among the Simple Math, Regression Analysis, Queueing Theory, Ratio Analysis, and Benchmark approaches and ranks them by relative cost, precision, flexibility, and usability factors (Shallahamer, 2003, p. 12). Herein, I will focus on queueing theory/analytical modeling which employs a simple M/M/1 queueing model to answer the question by generating results, which I will later compare with the results of simulation.

The simulation approach is based on a gradual increase in concurrency while measuring its effect on response time. Each test consists of a number of runs, usually 10. During each run, which normally lasts 1 minute, *concurrent* users, scheduled to execute a query in question, log in according to a requested distribution, run their query, and log

[4] "The sad fact is that SQL fails in all too many ways to support the relational model properly; it suffers from numerous sins of both omission and commission …. Because SQL's support for the model is so deficient, it gives you rope to hang yourself" (Date, 2005, p. xvi). Testing more than a single version of the functionally equivalent SQL is essential in the case of SQL statements which are either important to business, or resource-consuming, or frequently-executed, or executed by a large number of concurrent users. "The benefit-to-cost ratio is so high that all significant database-based applications should have their high-load SQL tuned" (Tow, 2004, p. 2).

[5] There is no shortage of SQL tuning means in the Oracle realm today: rewording SQL; using hints directly (given access to the source code) or indirectly (via stored outlines); modifying system/session parameters; seeding statistics with dbms_stats; modifying system statistics; using SQL profiles or dbms_advanced_rewrite. See, for example, Lewis (2005) and Tow (2004).

[6] By "properly stated" I mean "properly quantified." In successful projects, the step of quantification of business requirements is usually done in a business specification document.

[7] The FTS asymptotic time complexity function is O(n), while IRS complexity function is $O(\log_2 n)$. See Millsap, 2001, p. 2 for details; and Dali, 2006, Doc. A for basic test results. That is not to say that an execution plan containing FTS is necessarily bad from a *performance point of view*; that is to say, however, that it is usually is a 'suspect' from a *scalability point of view*, if the data source is expected to grow fast.

out.[8] The users are 'concurrent' not in the sense of simultaneous login, but in the sense of simultaneously executing a query *some time* within this 1 minute run.

As a basis for measurement recording, I chose Tom Kyte's test-harness (e.g., Kyte, 2003). I am confident that measurement recording can also be based on SQL tracing (e.g., Millsap & Holt, 2003), provided that a large number of trace files are processed through some automatic means (Dali, 2006, Doc. B). I have expanded the test-harness to accommodate more than two runs (e.g., McDonald, 2004, p. 166) and to keep recordings of resource consumption separate from those of response time. Snapshots are taken in equal predefined intervals (e.g., 1 minute) at the end of each run independently of response time, i.e., regardless of whether the workload ends at the snapshot point or not. Response time, however, is recorded when a workload execution is completed.

To realistically simulate a concurrent user load and to compare the results of simulation with the results of analytical modeling, it is necessary to 'inject' users in exponentially distributed interarrival times (or equivalently, with a Poisson distribution of the number of arrivals per unit of time). This is the primary distribution used in the simulations. In addition, I operate with the concept of *zero-distribution* of arrival times (i.e., all users log in at once) and the concept of *deterministic distribution* (i.e., no variance in login intervals). Notwithstanding my choice of distributions for the purposes of this paper, *any* custom distributions can also be accommodated.

The above approach is materialized in a program called SSS-test for SQL Scalability Simulation, which aides in answering SCUC concerns via simulation

Scenario

A company expands its operation to ten locations. At present, there are a million employees (EMP) uniformly distributed across 10,000 departments with the same average salary in each department (DEPT). A simple script to create this scenario using classic EMP and DEPT tables is shown in listing 1 of (Dali, 2006, Doc. C).

The query undergoing unit-testing is ready to be deployed in a production system, where it will be executed concurrently by 15 users every minute, according to SLA is shown listing 2:

Listing 2: Query in question and the underlying view

```
SELECT count(*), avg(sal) FROM v_dept_emp_big_trunc
WHERE  loc_trunc between 4 and 5;

CREATE  VIEW v_dept_emp_big_trunc AS
SELECT  d.*, e.*
FROM    dept_big d, emp_big e
WHERE   d.deptno=e.deptno_trunc;
```

The objective is to find the number of employees and their average salaries in two locations. Due to the company's transitional state, the employee data are: 1) changing continuously; 2) used by multiple human resources (HR) applications, and 3) expected to be queried frequently and concurrently.[9]

[8] If the stateless connection mode is requested, otherwise user logins occur at the beginning of each run. See stateless vs. stateful paragraph for further details

[9] I realize that this scenario is somewhat artificial (e.g., INVOICE and INVOICE_LINE would probably be more realistic); however, the purpose is to show a simple aggregation over a two-way join using only DEPT and EMP tables.

The developer also suggested an alternative—'tuned'—solution effectively swapping an original hash join (HJ) to a nested loops (NL) join method shown in listing 3:

Listing 3: Alternative, 'tuned' solution for the query in quesion

```
SELECT count(*), avg(sal) FROM v_dept_emp_big_trunc_hint
WHERE  loc_trunc between 4 and 5;

CREATE  VIEW v_dept_emp_big_trunc_hint AS
SELECT  /*+ index(e emp_big_deptno_trunc_idx) */ d.*, e.*
FROM    dept_big d, emp_big e
WHERE   d.deptno=e.deptno_trunc;
```

Single-User Test Results

The results of 3 consecutive runs of the original and 'tuned' SQL statements, as reported to a development leader, are summarized in table 1 (see Dali, 2006, Doc. D & Doc. E for details, including the use of SSS-test to automate calculation of service time).

Table 1: Response time and LIO comparison of Original vs. 'Tuned' SQL for a single-user test

Joining Method	Response time – ela [sec]		CPU time [sec]	# of LIO [thousands]	
	Over 3 runs	Avg. per run	Avg. per run (from SSS-test)	Over 3 runs	Avg. per run
Original – HJ	11.06	3.69	3.23	22.4	7.5
Hinted – NL	7.60	2.53	2.04	17.8	5.9

There is only about a one-second difference in response times between the two alternative statements, and both times are well below the SLA 10-second limit. The 'tuned'[10] statement does not appeal to the development leader due to the use of an 'unsafe' hint,[11] especially since this statement appears to achieve only marginal performance gains. For the purpose of this scenario, the data in the data sources involved is not volatile, i.e., has a growth rate from low to none; therefore, scalability with respect to data volume is not a concern. SCUC, however, is.

To be sure, the above results are those of a single-user testing. If it turns out that in an SLA-specified 15-concurrent-user run the gap between the original and 'tuned' statements widens to the point in which the former becomes prohibitively slow (over 10 seconds), while the latter still performs within the response time requirement, the development leader may look at it differently.

The Analytical Modeling Solution

Below are the input parameters for the capacity planning exercise:

$\tau_4 = \tau_8$ = average interarrival time = 60 seconds/15 users = 4 [sec]

$\lambda_4 = \lambda_8$ = average arrival rate = $1/\tau_4$ = 0.25 [1/sec]

[10] This is, of course, not the only alternative and not the best one (Dali, 2006, Doc. F); but it is good enough to demonstrate the concepts discussed in this paper.

R_{max} = worst tolerated response time = 10 [sec]

$\mu_1 = \mu_2 = \mu_3 = \mu_4$ = average service rate of the original SQL = 1/3.23 [1/sec]

$\mu_5 = \mu_6 = \mu_7 = \mu_8$ = average service rate of the hinted SQL = 1/2.14 [1/sec]

To allow for the future simple simulation, I assume a single-instance single-CPU system (i.e., a single queue and a single channel). The results of analytical modeling of both an original and a 'tuned' SQL statements performed using a capacity modeling workbook found in Millsap & Holt (2003) are presented in chart 1. The original workbook is slightly expanded to demonstrate four *evolution* points per SQL alternative: the response time is calculated at 1, 10, 12, and 15 users per minute. All 8 points are highlited in green in the table and appear on the graph in the form of ■ for the original query and in the form of ◆ for the 'tuned' query.

As clear from table 1, there is only about a one-second difference in both the CPU-times and elapsed times for a single user. Yet, as follows from chart 1, the two SQL statements look rather different under the requested concurrency load of 15 users. Therefore, from an analytical modeling point of view, while the 'tuned' SQL statement is still acceptable (with an average expected response time of 4 seconds), the original one is not (with an average response time of 17 seconds), since the latter exceeds the SLA limit of 10 seconds.

[11] There is no lack of discussions on the subject of 'safe' vs. 'unsafe' hints. E.g., one good example would be this thread: http://asktom.oracle.com/pls/asktom/f?p=100:11:0::::P11_QUESTION_ID:7038986332061 and another is this article: http://www.jlcomp.demon.co.uk/hinted_sql.html

Queueing Theory Multiserver Model

M/M/m 3.1e (2003/03/11)

	Alternative A				Alternative B			
users #	1	10	12	15	1	10	12	15
name	$value_1$	$value_2$	$value_3$	$value_4$	$value_5$	$value_6$	$value_7$	$value_8$
Units of measure								
Jobunit				stmt				stmt
Timeunit				sec				sec
queueunit				system				system
serverunit				CPU				CPU
serviceunit				sec/stmt				sec/stmt
throughputunit				stmt/sec				stmt/sec
Service level agreements								
λ	0.017	0.167	0.200	0.250	0.017	0.167	0.200	0.250
r_{max}	10	10	10	10	10	10	10	10
Architecture								
Q	1	1	1	1	1	1	1	1
M	1	1	1	1	1	1	1	1
μ	0.310	0.310	0.310	0.310	0.467	0.467	0.467	0.467
Performance forecasts								
color code	■				◆			
model	1 x M/M/1				1 x M/M/1			
ρ	5.4%	53.8%	64.6%	80.8%	3.4%	34.0%	40.8%	51.0%
S	3.230	3.230	3.230	3.230	2.040	2.040	2.040	2.040
W	0.184	3.766	5.894	13.549	0.072	1.051	1.406	2.123
R	3.414	6.996	9.124	**16.779**	2.112	3.091	3.446	**4.163**
$CDF(r_{max})$	**90.40%**	**68.13%**	**58.39%**	**37.92%**	**97.74%**	**92.48%**	**90.19%**	**85.36%**
1 - $CDF(r_{max})$	**9.60%**	**31.87%**	**41.61%**	**62.08%**	**2.26%**	**7.52%**	**9.81%**	**14.64%**

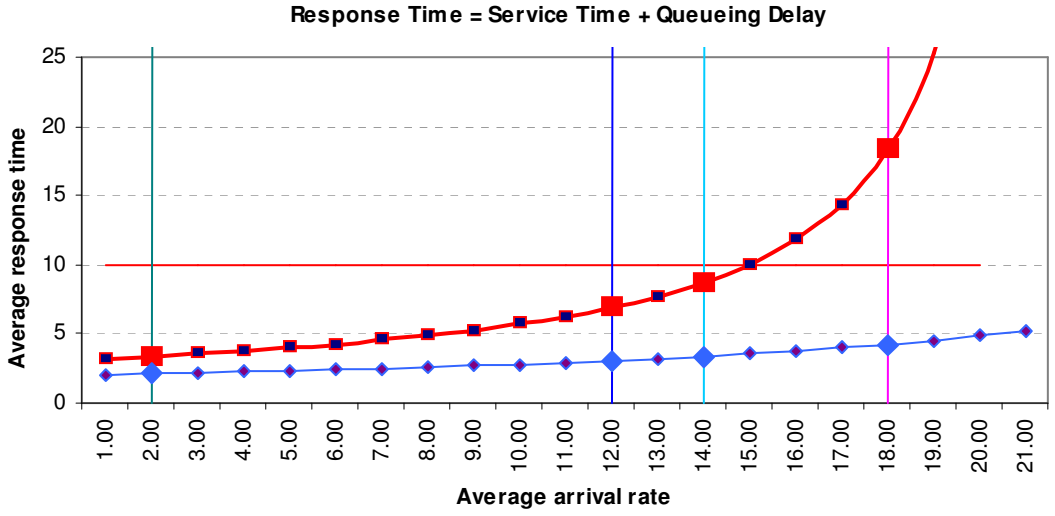

Chart 1. Comparison of an Original and a 'Tuned' SQL Statements using Cary Millsap's Forecasting Workbook (Millsap & Holt, 2003). Reproduced with permission.

The Simulation Solution: SSS-test

Application of SSS-test to solving a SCUC question in a given scenario are presented below

Exponentially Distributed Interarrival Times

To generate exponentially distributed interarrival times I applied a method known as an "inverse transform technique."[12] The implementation guided by Ross (2001, p. 45) is presented in Dali (2006, Doc. G) and used as follows:

- In the first run of a test, SSS-test checks whether a 'user-login-times' file is available. If it is, SSS-test asks whether it is to be re-used or whether a new one is to be generated (*generate new* vs. *reuse* modes).
- If a new file is requested, SSS-test generates U(0,1) and applies an inverse transform technique for a discrete random variable to create a 'user-login-times' file for a run.
- SSS-test repeats the previous steps for all, usually 10, subsequent runs with different λ in each.

The strategy is to use SSS-test once in the *generate new* mode for the original query test and then to repeat it for the 'tuned' query test in the *reuse* mode. In this way, random numbers feeding the inverse transformation, which in turn generates exponentially distributed interarrival times, are the same for both tests. This effectively makes the ramp-up identical, which, in turn, allows for congruent comparison.

Response Time Measurement

Two alternative solutions are used by SSS-test to capture a response time. The first method, used in earlier versions of SSS-test, relies on measuring an execution time from the beginning of a run and subtracting a corresponding sleep time. For zero-distribution (as defined on page 4), there is no sleep time; for deterministic distribution sleep time is known; for exponential distribution, it becomes dynamic and varying depending on the user. Therefore, with two simple distributions—zero-distribution and deterministic distribution—the response time measurement is straightforward (Dali, 2006, Doc. H). I capture the first and the last user response time and derive an average response time for a run based on these numbers. Measurements become more complex in the case of Poisson arrivals, in which I still capture the first/last-user *preliminary* response time with a needed correction on the now dynamic sleep time experienced by a user prior to the login. Then, the sleep time is subtracted from the preliminary response time, and the result represents the *final* response time. The sleep time varies from run to run due to the nature of a random variable used to generate exponentially distributed interarrival times. After the first/last-user response time calculation, I use a weighted-averages approach to obtain the average response time of a run. Since in theory about 63% of interrarrival times are less than a mean interarrival time, I assign this 'weight' to the first-user response time, and 37% for the last-user response time.[13]

[12] "This method is based on the observation that given any random variable x with a CDF F(x), the variable u=F(x) is uniformly distributed between 0 and 1. Therefore, x can be obtained by generating uniform random numbers and computing $x=F^{-1}(u)$" (Jain, 1991, p. 474).

[13] This is the way it is implemented in the current version of SSS-test. Needless to say that "average response time of a run" is crude approximation. One obvious improvement would be to capture response time of all users. Meanwhile, the calculation of the average response time for the run can probably be enhanced by replacing the theoretical 63% with the real/dynamic percentage of users arriving at (or prior to) a mean interarrival time.

The method used in the later versions is more straightforward. It simply captures beginning (via a shell's co-process) and end of a query execution and does the arithmetic. Also the notions of the first/last users are replaced with user with best/worse response time users respectively (now denoted as BUser, WUser) and consequently the weighted average approach reduces to a simple case of arithmetic average.

Simulation Results

An excerpt of the results (Dali, 2006, Doc. I) for a single test of 10 runs is shown in listings 4 and 5 for the original and 'tuned' SQL respectively and the comparison is presented graphically in chart3.

Listing 4: Simulation results of Original SQL with exponentially distributed interarrival times with stateless connections

LATCH...shared pool	789	827	1,386	1,610	1,572	1,808	2,760	3,221	3,932	8,928
LATCH...library cache pin	844	894	1,477	1,672	1,609	1,842	2,798	3,270	3,983	9,126
LATCH...simulator hash latch	856	1,807	2,574	3,446	4,423	5,140	8,571	10,290	12,862	17,007
LATCH...checkpoint queue latch	1,051	971	1,042	1,023	1,483	1,043	1,013	1,049	1,058	1,088
LATCH...SQL memory manager wor	1,415	1,350	1,360	1,431	1,362	1,366	1,382	1,390	1,402	1,523
LATCH...library cache	1,527	1,634	2,721	3,118	2,996	3,442	5,254	6,152	7,504	17,097
LATCH...cache buffers lru chai	5,963	11,799	17,826	23,719	29,830	35,513	59,272	70,956	88,375	117,997
LATCH...cache buffers chains	21,897	42,722	64,231	84,986	106,361	126,752	210,792	252,692	315,289	421,818
... Elapsed Time: Snap [cs]	6,374	6,018	6,022	6,030	6,019	6,015	6,018	6,017	6,018	6,438
... Elapsed Time: BUser [cs]	450	731	383	1,075	343	350	320	1,107	378	2,766
... Elapsed Time: WUser [cs]	450	753	507	1,193	782	668	1,334	1,604	2,111	4,612
Resources (stat/latch) & Ela	Run1 - 001	Run2 - 002	Run3 - 003	Run4 - 004	Run5 - 005	Run6 - 006	Run7 - 010	Run8 - 012	Run9 - 015	Run10 - 020

Listing 5: Simulation results of 'Tuned' SQL with exponentially distributed interarrival times with stateless connections

LATCH...shared pool	766	830	1,590	1,250	1,473	1,700	2,559	2,982	3,610	4,682
LATCH...library cache pin	843	918	1,742	1,366	1,600	1,844	2,786	3,260	3,970	5,148
LATCH...checkpoint queue latch	1,280	1,227	1,434	1,114	1,074	1,348	1,079	1,120	1,125	1,167
LATCH...SQL memory manager wor	1,413	1,346	1,360	1,350	1,352	1,354	1,362	1,366	1,372	1,382
LATCH...library cache	1,521	1,672	3,224	2,524	2,962	3,418	5,192	6,082	7,408	9,627
LATCH...cache buffers chains	10,879	20,923	31,306	40,760	50,748	61,124	100,809	120,710	150,719	200,669
... Elapsed Time: Snap [cs]	6,165	6,014	6,027	6,015	6,017	6,017	6,020	6,018	6,017	6,018
... Elapsed Time: BUser [cs]	342	529	279	319	276	221	220	420	239	735
... Elapsed Time: WUser [cs]	342	534	362	570	478	437	662	831	1,129	2,021
Resources (stat/latch) & Ela	Run1 - 001	Run2 - 002	Run3 - 003	Run4 - 004	Run5 - 005	Run6 - 006	Run7 - 010	Run8 - 012	Run9 - 015	Run10 - 020

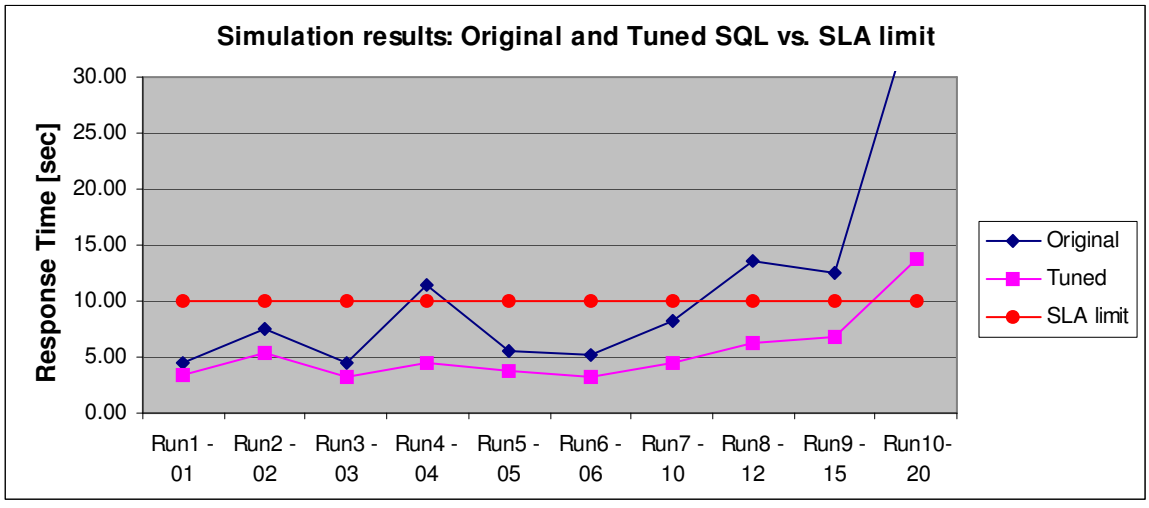

Chart 3: Response time comparison of the Original vs. 'Tuned' SQL simulations

As follows from Chart 3:

- the response time for the 'tuned' SQL stays below the SLA requirement of 10 seconds for the requested concurrency of 15 users;
- at the low concurrency load of up to 10 users, the response times of *both* SQL statements remain similar and below the SLA limit. This means that if there is room for negotiating the SLA concurrency load, it may be a more sensible course of action than engaging in tuning activities;

Simulation Results of Five Consecutive Tests To Assess Sampling Variation

How reliable can a simulation based on random numbers be? The simulation is repeated five times (i.e., a new set of exponentially distributed interarrival times is generated for each test) to check sampling variation (i.e., variation from one run to another) for the 'tuned' query. Results are presented in listing 6 and chart 4 (Dali, 2006, Doc. J, listing 7).

Listing 6: Percentage of users arriving at or prior to a theoretical mean interarrival time

Test#	Run01-01	Run02-02	Run03-03	Run04-04	Run05-05	Run06-06	Run07-10	Run08-12	Run09-15	Run10-20	Avg
1		100	100	75	100	100	90	91	86	95	83.7
2		100	100	100	100	66	90	75	93	90	81.4
3		100	100	75	80	83	80	100	80	90	78.8
4		100	100	100	60	66	90	83	73	90	76.2
5		100	100	75	80	83	80	100	100	90	80.8
											80.2

As clear from listing 6, the majority of interarrival times are below the theoretical mean, as expected. In fact, the majority of interarrival times is greater than 63% (i.e., 80.2%), due to the approximations involved in generating exponential distribution. Chart 4, in turn, shows that there are some variations in response time; however, the shape of the curves remains stable.

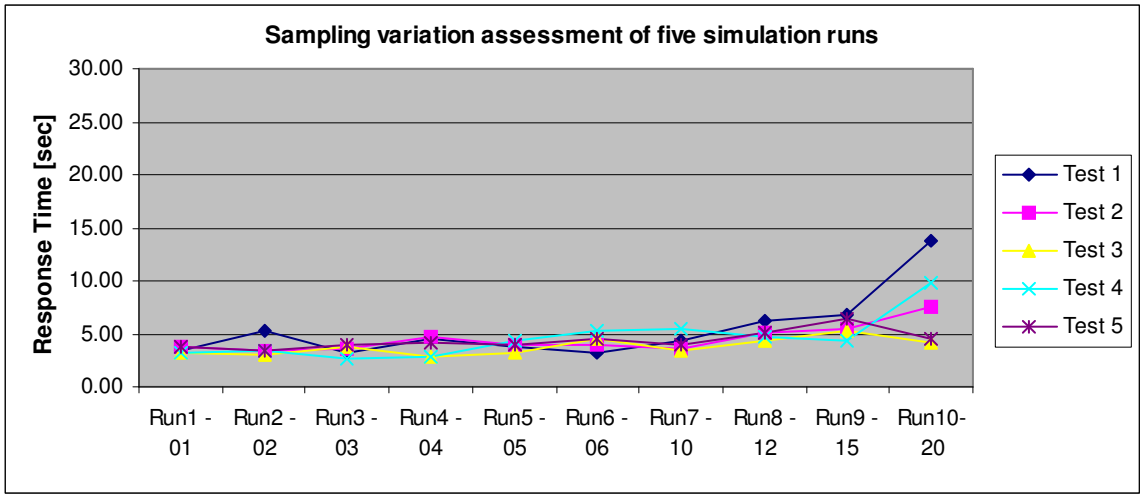

Chart 4: Response time measured in five consecutive simulation tests

Comparison of Analytical Modeling and Simulation Solutions

The response time comparison of both original and 'tuned' SQL is presented in Chart 5:

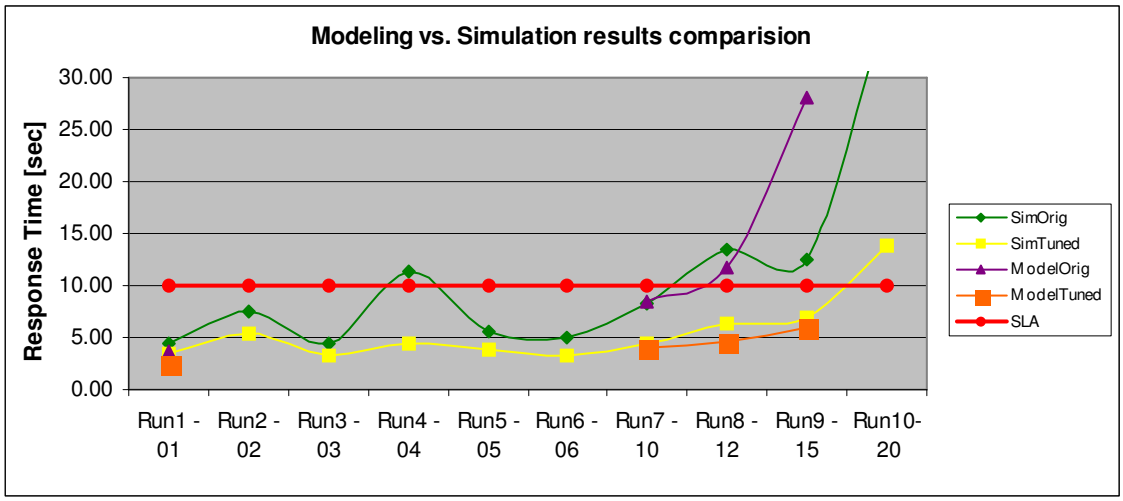

Chart 5: Response time comparison of analytical modeling vs. simulation results

The simulation forecasts and their modeling counterparts have similar curves, which reflect the similar performance. The response time forecasted by both mathematical modeling and simulation for 15 concurrent users is within the SLA requirement of 10 seconds for the 'tuned' SQL (4 seconds in modeling, and 7 seconds in simulation) and outside the 10-second requirement for original SQL (17 seconds in modeling, 12 seconds in simulation). The convergence of results from two evaluation methods—analytical modeling and SSS-test—is a solid indication that the forecasts are accurate.[14]

For a proposed OLTP system, as well as for many existing applications, the Poisson arrivals assumption is quite reasonable. However, what if there are other competing activities on the system, lock or latch contention, or the measurements (e.g., Oracle standard auditing) signal that an arrival process is more *aggressive* than Poisson's (i.e., the number of arrivals is greater at the beginning of the interval than in Poisson's distribution)? The last condition can happen, when, for example, the majority of users log in precisely at the beginning of the working shift at a plant at 7AM, right after punching a time card; or when at 2PM on Friday all the departments have to submit a certain HR report. This is when an arrival process more resembles a zero-distribution case than Poisson.[15]

In the above scenarios or their combinations, even the 'tuned' SQL may fall short of the SLA 10-second requirement (Dali, 2006, Doc. H). This could not be discerned through analytical modeling, which relies solely on exponential distribution of both service and arrival times, disregards competing activities, and does not take into account any imperfections of the real-life system (e.g., as mentioned by one of the reviewers, what if the NL plan

[14] "Do not trust results of an analytical model until they have been validated by a simulation model or measurements" (Jain, 1991, p. 32). Although by "simulation" Dr. Jain probably meant 'classic simulation' rather than the 'benchmark-simulation' discussed in this study, the idea of verifying the evaluation results through an alternative method is still valid.

[15] The standard analytical solution for this case is "segmentation of a modeling domain" described in Millsap & Holt, 2003, pp. 238-239

serializes worse on, say, latch acquisition). From a different angle, this highlights the advantages of the SSS-test for evaluating SCUC and predicting performance.

Now, when the development leader is made aware of the above situation as a result of simulation, he/she may explore a number of options: to re-negotiate the SLA response time or concurrency requirements with the business; to further improve the response time of the query in question (service time in particular); to tune other parts of the system; or perhaps the most 'trendy' is to upgrade the hardware (e.g., to purchase faster CPUs to reduce service time and/or additional CPUs to "increase the number of parallel service channels" to delay the "knee in the curve" (Millsap & Holt, 2003, pp. 256-261).[16]

Stateless vs. Stateful simulations

All of the tests above were conducted assuming a stateless connection model, often found in Web applications. SSS-test allows for both stateless and stateful simulations. This is an important decision that a designer of any database-driven application makes, and the incorrect choice is rated as number one mistake in Oracle performance documentation ("Top Ten Mistakes Found in Oracle Systems", Release 9.2)

Under the stateless paradigm SQL*Plus session is opened every time a virtual user has to run a query under test. In effect, response time here measures not only the time it takes to run a query but also the login 'overhead'. In the stateful world, all users login up in front[17] and their login time is not counted in the response time. User logs in and immediately gets blocked on dequeing a message, which is not enqueued by a co-process controller 'daemon' until it is time for a user to run his query.

Which paradigm to use obviously depends on the application being tested. Some thought should be given to how well SQL*Plus logins resemble a given application logins. SSS-test also allows to run a simple head-to-head comparison of both paradigms.[18] Results of a test comparing 5 runs stateless against 5 runs stateful simulations with zero-distribution of arrival times are presented in the listing 7 and graphically in the chart 6 below:

Listing 7: Results of the stateLESS vs. stateFUL comparison test for the zero distribution

	Zero (all-at-once) Distribution									
	StateLESS					StateFUL				
... Elapsed Time: FUser [cs] c	329	1,099	1,858	2,578	3,491	699	855	1,467	2,126	2,739
... Elapsed Time: LUser [cs] c	329	1,106	2,028	2,919	3,766	699	1,005	1,892	2,737	3,625
Resources (stat/latch) & Ela	Run1 - 001	Run2 - 005	Run3 - 010	Run4 - 015	Run5 - 020	Run1 - 001	Run2 - 005	Run3 - 010	Run4 - 015	Run5 - 020

[16] Needless to say that the last approach is suboptimal and generally reprehensible

[17] In the current version "up-in-front" means the beginning of each run. It would probably be more accurate to determine the maximum number of users in all runs for a given test and log them in once per test, as opposed to once per each run in a test. This enhancement is planned for the future version of SSS-test.

[18] In the current version only zero and deterministic distribution comparisons are implemented, while the exponential distribution is planned for the future version of SSS-test.

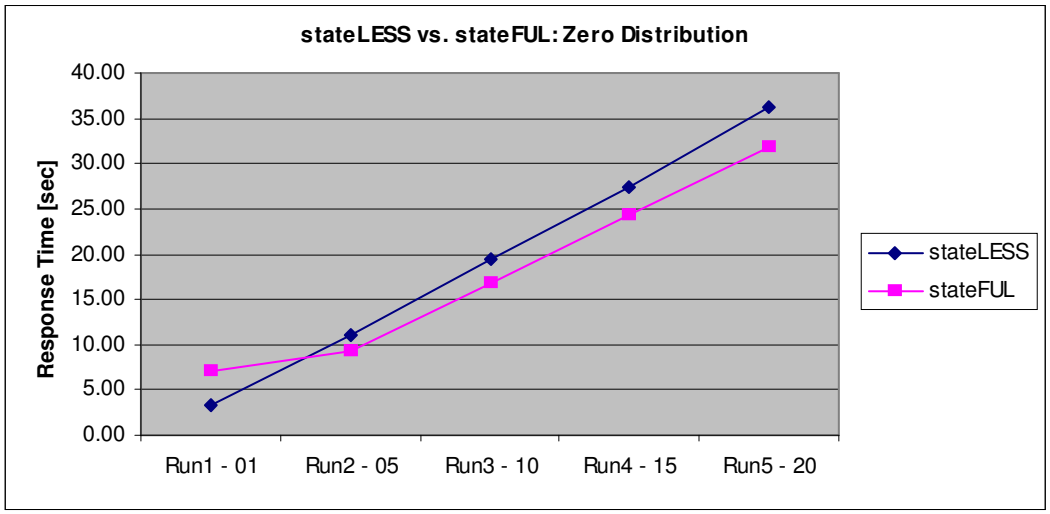

Chart 6: Results of the stateLESS vs. stateFUL comparison test for the zero distribution

Running SSS-test: A Step Sequence

SSS-test engenders a great deal of flexibility. A step sequence presented below is suggested but by no means compulsory.

1. Gather SLA performance requirements for the task at hand: the number of concurrent users in an interval; the frequency of execution; and the expected response time;
2. Run the SSS-test in the *get-service-time* mode to calculate a mean service time and a standard deviation for a single user over a desired number of runs (usually 10);
3. Run the SSS-test in the *get-response-time* mode for a requested number of runs (usually 10). In each run:
   - Determine the mean interarrival time via τ=<time interval>/<# of concurrent users>;
   - Calculate the probability density function (PDF) of exponentially distributed interarrival times in the interval of 0..30 seconds using $P_t = P(T=t)= \lambda e^{-\lambda t}$;
   - Generate a U(0,1) set of random numbers equal to the number of concurrent users in a run (In the above scenario, there are 1-6, 10, 12, 15 and 20 users in runs 1-10 respectively);
   - Apply an inverse transform technique on U(0,1) by comparing it to PDF to obtain exponentially distributed interarrival times;
   - Optionally, verify that about 63% of the generated interarrival times are indeed below the mean τ (and about 63% are below a half-mean);
4. Optionally, triangulate. Since an arrival process *distribution* affects response time, consider the two extreme cases below, keeping in mind that the real-life response time may fall someplace in the middle (probably, close to the Poisson arrivals case simulated in step 3):
   - Run a zero-distribution simulation to check the 'worst case' scenario of all users logging in *at precisely the same time*;
   - Run a deterministic distribution simulation to check the 'perfect' scenario of what response time would be if the users log in *precisely at the same mean interarrival time interval* from the previous user's login time.

Absent "true randomness,"[19] as long as utilization, derived as $\rho = \chi / \mu$, is less than 100%, i.e. a service rate is greater than an arrival rate, a system should be able to handle the load.

5. Optionally, as demonstrated in the 'five-test simulations' section, despite the random numbers, variations in response times are reasonably small. Nevertheless, it may still be prudent to treat a single test as a sample, to repeat step 3 a number of times, and to use the rules of statistical inference to arrive at a better understanding of response time. One way to do it, in the above scenario, is to run a test of 10 runs with 15 concurrent users in each run, and to calculate the average response time.
6. Optionally, compare the simulation results with the results of analytical modeling.

Conclusion

There are a number of differences between the analytical modeling and the simulation solutions presented in this paper. In analytical modeling, exponential distribution of *both* the interarrival times and the service times *had to* be assumed. In simulation, the same assumption *may* still be kept regarding the interarrival times, but the service times are simply *measured*. By measuring as opposed to calculating using the model, any imperfections of the real system (e.g., latching) disregarded by analytical modeling are automatically and explicitly taken into account. Simulation also allows for measuring the effect of other concurrent business tasks on a query in question.

In conclusion,

- SCUC testing should be a part of routine SQL testing.
- Two SQL statements yielding close response time in single-user testing may scale differently with respect to concurrent user count.
- It is not difficult to simulate a query or a business process even on a system that has a plenty of other concurrent activities, as long as these activities can be reproduced at the time of a simulation. In other words, simulation testing is not limited to answering only stand-alone query scalability concerns.
- An arrival process that does not comply with Poisson distribution limits the use of analytical modeling to smaller windows where Poisson distribution assumption holds true. The SSS-test, however, is more accommodating and less restrictive, as it allows for injecting users in any custom 'user login times' distribution.
- The SSS-test can be used by designers and architects who model/ simulate new ideas. It can also be used by developers in the same situations in which they use Tom Kyte's standard test-harness and, of course, by DBA/QA/QC personnel prior to approval of a request to implement a change in a production system.
- It is my hope that SCUC will become an integrative part of the routine unit testing. Moreover, I hope that adoption of SCUC will encourage developers *to request* a 'concurrency spec' when they are given a development assignment.

[19] "Arrival times sometimes have appearance of being clustered with occasional large gaps separating clusters, because of the substantial probability of small interarrival times and small probability of large interarrival times, but such irregular pattern is all part of true randomness" (Hiller, 2002, p. 843).

Acknowledgments

I thank Cary Millsap for his continuous support and encouragement in the course of this study as well as reviewing the preliminary version of this paper and providing useful comments. I also thank Jean-Christophe Marly, a QA architect at Compuware, for reviewing the first draft.

References

1. Millsap, C. with Holt, J. 2003. *Optimizing Oracle Performance*. O'Reilly & Associates, Sebastopol, CA.
2. Millsap, C. 2001. "Scalability is a Rate of Change." Available at: < http://www.hotsos.com >.
3. Aho, A. V. 1974. *The design and analysis of computer algorithms*. Reading, Mass.: Addison-Wesley.
4. Date, C. J. 2005. *Database in depth: relational theory for practitioners*. O'Reilly Media, Sebastopol, CA.
5. Tow, D. 2004. *SQL Tuning*. O'Reilly & Associates, Sebastopol, CA
6. Lewis, J. 2005. *Cost-based Oracle fundamentals*. Apress, New York, NY.
7. Dali, B. 2006. "SSS-test tool." Available at: < http://www.borisdali.com >.
8. Shallahamer, C. 2003. "Course of Forecasting Oracle Performance." OraPub, Nashville, TN.
9. Kyte, T. 2003. *Effective Oracle by design*. McGraw-Hill/Osborne, Emeryville, CA.
10. McDonald, C. with Katz, C., Beck, C., Kallman, J.R., Knox, D.C. 2004. *Mastering Oracle PL/SQL: Practical solutions*. Apress, New York, NY.
11. Ross, S. M. 2001. 3rd ed. *Simulation*. Academic Press, San Diego, CA.
12. Jain, R. 1991. *The art of computer systems performance analysis: Techniques, for experimental design, measurement, simulation, and modeling*. John Wily & Sons, New-York, NY.
13. "Top Ten Mistakes Found in Oracle Systems." In *Oracle9i Database Performance Planning*, Release 9.2. Available at < http://download-east.oracle.com/docs/cd/B10501_01/server.920/a96532/ch2.htm >
14. Hillier, F. S. 2002. 7th ed. *Introduction to operations research*. McGraw-Hill, New York, NY.