A State Machine for Database Non-functional Testing

Jorge Augusto Meira, Eduardo Cunha de Almeida, Yves Le Traon

A State Machine for Database Non-functional Testing

Jorge Augusto Meira[1,2], Eduardo Cunha de Almeida[1,2], and Yves Le Traon[1]

[1]University of Luxembourg – {jorge.meira, yves.letraon}@uni.lu

[2]Federal University of Paraná, Brazil – eduardo@inf.ufpr.br

ABSTRACT

Over the last decade, large amounts of concurrent transactions have been generated from different sources, such as, Internet-based systems, mobile applications, smart-homes and cars. High-throughput transaction processing is becoming commonplace, however there is no testing technique for validating non functional aspects of DBMS under transaction flooding workloads. In this paper we propose a database state machine to represent the states of DBMS when processing concurrent transactions. The state transitions are forced by increasing concurrency of the testing workload. Preliminary results show the effectiveness of our approach to drive the system among different performance states and to find related defects.

1. INTRODUCTION

High-throughput transaction processing is becoming commonplace due to the number of devices submitting concurrent transactions, including Internet-based systems, mobile applications, smart-homes and cars. The major vendors of DBMSs claim that their systems treat large numbers of concurrent transactions and keep performance by scaling up their environments. However, there is no testing technique for validating non functional aspects of DBMS under transaction flooding circumstances, such as: performance and scalability. Indeed, performance and scalability problems can conduce the DBMS from a steady to a thrashing state in seconds. In general, the testing techniques rely on a combination of unit tests with some "mocking" approach that cannot mimic large-scale workloads and environments. Traditional DBMSs are typically tested through well-known benchmarks, such as TPC-like [1] and most recently by the Yahoo Cloud Serving Benchmark (YCSB) [2]. While benchmarks are suited for metrics comparison, they can not expose performance related defects. In this work we propose a Database State Machine (DSM) to represent the DBMS under transaction flooding circumstances. The DSM is composed by 5 states to model performance behaviours while the transaction workload shifts. We force the state transitions by increasing concurrency of the testing workload. The goal is to pinpoint conditions of performance loss and predict stressing states that can damage the system execution.

[1]http://www.tpc.org/

IDEAS '14, July 07 - 09 2014, Porto, Portugal.

ACM 978-1-4503-2627-8/14/07 ...$15.00
http://dx.doi.org/10.1145/2628194.2628201.

2. THE DATABASE STATE MACHINE

In this section, we present our DSM (see Figure 1).

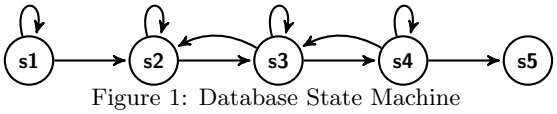

Figure 1: Database State Machine

We define three performance constraints to model the state transitions: performance variation, throughput and performance trend.

DEFINITION 1. *(Performance variation) We define performance variation, denoted by Δ, as the variation of the number of transactions treated per second (TPS).*

DEFINITION 2. *(Throughput) We define throughput, denoted by δ, as the maximum transactions treated per second in which at least n% percent of the transactions response time is less than or equal to m seconds.*

DEFINITION 3. *(Performance trend) The performance trend, denoted by the function $\varphi = x' - x$, where x is the actual performance, and x' is the performance prediction.*

2.1 The Database States

State 1 (s1): Warmup (W) considers the startup process carried out by the DBMS to let the internal services running (e.g., the transaction management service). Normally the warmup is defined based on *"performance variation". State Condition:* $\neg(\Delta \rightarrow 0)$.
State 2 (s2): Steady (S) was first defined by the Debit-Credit benchmark [1] as a non-saturated state on top of

[2]https://github.com/brianfrankcooper/YCSB

the performance constraints. We generalize the performance constraints to fit any transactional environment into the DSM. *State Condition:* $(\Delta \wedge \delta)$.
State 3 (s3): Under Pressure (UP) represents the DBMS on the limits of performance. During this state the system could still be able to deal with the same amount of transactions reached in the previous state. However, the δ condition is not respected due to response time degradation. *State Condition:* $(\Delta \wedge \neg\delta)$.
State 4 (s4): Stress (ST) is reached by pushing the DBMS beyond its performance limits. The goal is to conduct the system to a sensitive performance loss with respect to the number of transactions treated per second. This state differs from the *Under Pressure State* in the sense that DBMS is not able to keep the previously reached $(\neg\Delta)$. *State Condition:* $(\neg\Delta \wedge \neg\delta)$.
State 5 (s5): Thrashing State (Th) is classified as a period when a large amount of computer resources is used to do a minimal amount of work, with the system in a continual state of resource contention and unable to deal with any new request. We call $\varphi \to 0$ as the "thrashing-threshold". *State Condition:* $(\neg\Delta \wedge \neg\delta \wedge \varphi \to 0)$.

2.2 The State Transitions

The state transitions represent performance variation and throughput that we observe while testing performance of the DBMS. The goal is exposing performance defects and the conditions that made the DBMS to crash. State transitions are represented on Table 1.

Table 1: *State Transitions*

States	W	SS	UP	St	Th
W	$\neg\Delta$	Δ	-	-	-
SS	-	δ	$\neg\delta$	$\neg\Delta$	-
UP	-	δ	$\neg\delta$	$\neg\Delta$	-
St	-	-	Δ	$\neg\Delta$	$\varphi \to 0$
Th	-	-	-	-	$\varphi = 0$

The state machine is fed by an *alphabet* composed by $(\Delta, \delta, \varphi)$. We force state transitions and accelerate the DBMS lifetime, based on stimuli presented in two previous work [3, 2]. In order to calculate φ we use a set of observations given by a time-framed bounded (sliding window). For instance, in a graph viewer an observation is a 2D point representing the system performance in a given moment. With an optimal pre-set number of observations is possible to determine a performance slope by using an approximation solution method. We use the *Least Squares* method that consists in approximating the independent variable x_i and the dependent variable y_i to draw a curve, the performance slope (see Figure 2). Since the performance slope is known, we can compute its trend in order to predict the moment of transition from the stress to thrashing state. We use the derivative calculation to compute tangent line (i.e., point x'). In our context, the tangent line is the performance trend used to predict how far the system is from crashing (see Definition 3).

3. EVALUATION AND FUTURE WORK

We proceed experimental evaluation of our approach by testing 6 DBMS through the DSM upon TPC transactions: 3 disk-oriented (PostgreSQL, MySQL, Commercial DBMS-X) and 3 in-memory (VoltDB, NuoDB, Oracle NoSQL). We only present the PostgreSQL results, due space limitations. PostgreSQL passed through all the states of our DSM and presented a chaotic behaviour when placed from the stress state to thrashing. When exposed to these two states, the DBMS presents defects related to the inability to keep the pre-set performance, for instance, maintain the number of concurrent transactions. Besides, the response time per transaction increased from 1 to over 10 seconds. We also observed that the workload has a direct impact on the code coverage (i.e., from 40% (Steady) up to over 70% (Stress)), mainly in three packages: *freespace, page, transaction manager*. At the stress state, PostgreSQL was not able to manage the large number of requests to find free space. This behaviour is considered as a defect, since the disk has available space with low throughput. Beyond the increase in the code coverage, the DSM can also relate the *"workload defects"* to a specific DSM state, this means, pinpoint exactly which condition exposes the DBMS to such defect. Thus, our approach implements a non-functional testing that aims mainly ensuring the system reliability. Our main goal for future work is presenting a testing facility for helping the development of database auto-rescue mechanisms. This means, a testing facility to validate performance on-the-fly based on the DSM states, allowing performance improvements by increasing or decreasing computing resources, as needed.

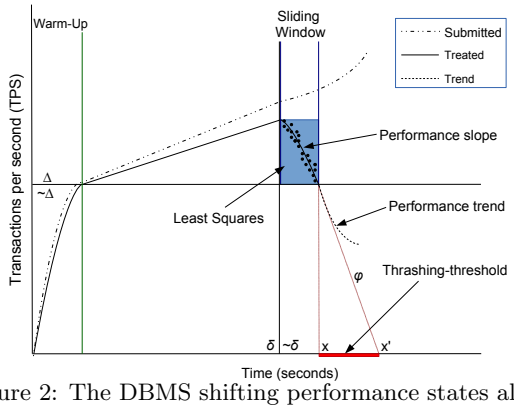

Figure 2: The DBMS shifting performance states along time.

Acknowledgement: Supported by the National Research Fund, Luxembourg Project 902399 / TOOM Project: C12/IS/4011170 and SERPRO/UFPR.

4. REFERENCES

[1] A. et al. A measure of transaction processing power. *Datamation*, 31(7):112–118, Apr. 1985.
[2] A. G. Fior, J. A. Meira, E. C. de Almeida, R. G. Coelho, M. D. D. Fabro, and Y. L. Traon. Under pressure benchmark for ddbms availability. *JIDM*, 4(3):266–278, 2013.
[3] J. A. Meira, E. C. de Almeida, G. Sunyé, Y. L. Traon, and P. Valduriez. Stress testing of transactional database systems. *JIDM*, 4(3):279–294, 2013.