DBMS Annihilator: A High-Performance Database Workload Generator in Action

Alberto Lerner
University of Fribourg
Switzerland

Matthias Jasny
TU Darmstadt
Germany

Theo Jepsen
Stanford University
USA

Carsten Binnig
TU Darmstadt & DFKI
Germany

Philippe Cudré-Mauroux
University of Fribourg
Switzerland

ABSTRACT

Modern DBMS engines can achieve unprecedented transaction processing speeds thanks to the invention of clever data structures, concurrency schemes, and improvements in CPU and memory subsystems. However, developing realistic and efficient networked clients to benchmark these systems remains daunting. Simply put, traditional client-side networking stacks present high overheads and thus cannot exercise the high performance that modern DBMSs can, in principle, provide. In this demo, we propose a different approach to benchmarking; we showcase a new framework that leverages hardware-software co-design. With our system, which we call the DBMS Annihilator, workloads are specified using a high-level language that is then converted into hardware (FPGA) for execution. The hardware we use is a commodity Smart NIC, allowing workloads to be fully reproducible to anyone using such hardware. A software console and dashboard provide real-time visibility and interactivity, which we explore in this demo.

PVLDB Reference Format:
Alberto Lerner, Matthias Jasny, Theo Jepsen, Carsten Binnig, and Philippe Cudré-Mauroux. DBMS Annihilator: A High-Performance Database Workload Generator in Action. PVLDB, 15(12): 3682 - 3685, 2022.
doi:10.14778/3554821.3554874

1 INTRODUCTION

Modern database engines can sustain rates of above 150M transactions per second (tps) [2, 3, 11, 14], but testing these systems at these transaction rates is challenging. Such rates correspond to between 100 and 200 Gbps of network traffic. Networking at this speed is not the problem; the market currently offers 200 Gbps network interface cards (NIC) and off-the-shelf 400 Gbps network switches. The problem is the difficulty of building benchmarking software that simulates enough database clients to request millions of transactions consistently. One may imagine that generating such a workload in a controlled way is possible, given that modern techniques like RDMA [8] and DPDK [5] enable building high-speed database clients in software. In reality, however, these techniques have limitations.

Proceedings of the VLDB Endowment, Vol. 15, No. 12 ISSN 2150-8097.
doi:10.14778/3554821.3554874

At 100 Gbps, a new packet can be sent into the network every 6.7 ns [12]. A single client machine cannot build new packets at this pace because of DRAM latency. Therefore, fast benchmarking tools use one of two techniques. First, they deploy several clients, generating a faster aggregated packet stream. The workload is, therefore, partitioned across the clients. Given that, in a typical workload, one client's transaction choice affects another client's transaction options, the clients are forced to synchronize painstakingly. Second, workloads can have their transaction requests pre-generated in memory. Unfortunately, copying pre-generated transactions onto the network sometimes causes duplicate or reordered transactions to be issued [1]. Workloads generated this way are random.

Developers of modern database systems often circumvent these problems by avoiding networked workloads. They run the workload generator on the same machine as the DB server and use pre-generated transactions. The workloads demonstrate the raw speed of the server, but, arguably, skipping networking frees CPU and memory resources artificially that would otherwise not be available to the database.

It is this lack of tooling for benchmarking fast databases that we address in this paper. We propose an entirely new approach to benchmarking that dynamically generates and issues controlled database workloads at high speeds. We are developing a solution that allows a user to describe the desired workload in a high-level language but provides easy-to-use tools that convert and run the workload in hardware (FPGAs). We call our system the DBMS Annihilator.

In summary, this demo paper introduces the DBMS Annihilator and makes the following contributions:

- We propose a software/hardware co-designed approach for database benchmarking tools.
- We implement such an approach using an off-the-shelf FPGA-based Smart NIC. Anyone with access to this Smart NIC can faithfully reproduce a benchmark.
- We show that the workload can be generated dynamically and interactively without needing a fleet of client machines or sacrificing speed.

In the remainder of the paper, we describe the DBMS Annihilator in detail in Section 2 and elaborate on the scenarios we intend to bring to the demo in Section 3. We comment on related work and future directions in Section 4. In the demonstration, the user will interact with our system's runtime tooling and experience how to find the limits and performance characteristics of an actual high-speed database.

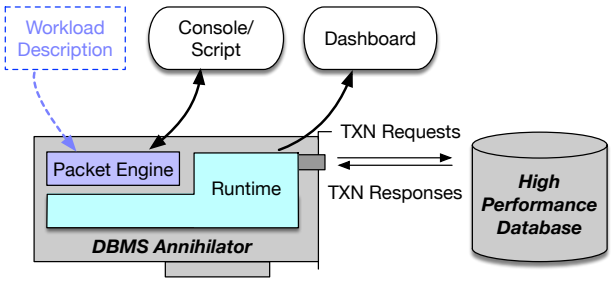

Figure 1: Workload descriptions are compiled into a hardware-based (transaction) packet generation engine. The engine and the runtime work autonomously, without any server assistance. They issue transactions against a database and record their responses. A Console and a Dashboard control and monitor the process, respectively.

2 THE DBMS ANNIHILATOR

The DBMS Annihilator is a hardware/software co-designed platform that leverages FPGA-based Smart NICs in implementing our approach. Figure 1 shows the system's main components. We present an overview of our system in Section 2.1 and describe its internal architecture in Section 2.2.

2.1 System Overview

At the heart of our solution lies a domain-specific language that can describe database workloads. The language has constructs for the static and dynamic components of the workload. In short, the static components comprise the transaction types to be issued. For instance, in TPC-C, those would be `NewOrder`, `Payment`, and so on. The dynamic components encompass the logic to generate transaction arguments to use during the run time and to mix the transaction types into a controlled workload. Using the TPC-C example once more, `NewOrder` and `Payment` should dominate the workload mix. Based on this workload description, the DBMS Annihilator then compiles a workload description into a hardware-based *Packet Engine* as the benchmark's driver.

Hardware Platform. The Packet Engine runs on a single FPGA-based network card. However, thanks to the hardware-based nature of our solution, the workload generated even by a single 100 Gbps Smart NIC replaces several software clients. We are currently targeting three boards: the Xilinx ZCU106, Alveo U50 and the Xilinx VCU129. The ZCU106 and U50 are cost-conscious, FPGA-based NICs with 10 and 100 Gbps network ports, respectively. The VCU129 is not exactly a network card because it is not a PCIe peripheral. It is a standalone platform containing the necessary hardware (i.e., 56G transceivers) to generate workloads at 400, 800, or 1200 Gbps.

Such a bandwidth easily matches and surpasses the network capacity of a single server. Currently, a server can support 400 Gbps of traffic by using two Mellanox ConnectX-6 200 Gbps NICs, each in a PCIe Gen4 ×16 or dual Gen3 ×16 slots. That is where our second platform, the VCU129, can be helpful. For instance, the extra capacity of the DBMS Annihilator can be used to generate traffic for the secondary servers in a replication scenario. As discussed in the previous section, producing such a coordinated workload in a software-based generator would not be practical, if at all possible.

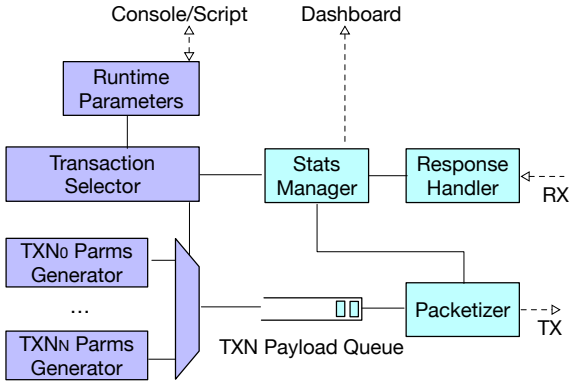

Figure 2: The DBMS Annihilator's architecture. Purple modules are tailored to the workload description.

Benchmarking Workflow. To run a benchmarking workload, we synthesize a bitstream, which we use to program the NIC's FPGA. The bitstream captures the *Workload Description* provided by the programmer and generates a *Packet Engine*. In turn, a *Runtime* subsystem in the card leverages the Packet Engine to produce and ship transaction requests using a networking protocol that the Database Under Test (DUT) understands. We currently support an RDMA-based protocol for its high-throughput and low latency, but we also plan to implement an ODBC/TCP-IP stack. Note that the network address of the DUT and the protocol to be used can be configured at run time.

The DUT sends back transaction response packets, which are captured and decoded by the Runtime subsystem. At this time, the system can calculate each transaction's latency and the instantaneous aggregated throughput the server is delivering. The results are displayed via a *Dashboard*. Optionally, the database can be instrumented to report on its resource consumption (e.g., CPU or memory utilization) in aggregate or on a transaction-basis level. These additional metrics can also be retrieved from the Dashboard.

Note that the Packet Engine's dynamic components can be modified during runtime. For instance, the frequencies with which each transaction appears in the workload can be adjusted. The modifications are made through the *Interactive Console* or via *scripting*. Furthermore, one can use *execution signals* to vary the workload. In other words, the workload can be changed in response to prior database errors or variations in performance. We describe such scenarios in more detail in Section 3.

2.2 System Architecture

The Packet Engine and the Runtime modules are composed of several sub-modules. The sub-modules interact to generate the workload and produce execution statistics, as shown in Figure 2. We describe each of the sub-modules in turn.

Transaction Parameter Generator. Each transaction type in the Workload Definition is represented by one of these modules. They produce the arguments that are used in each new transaction request. The generators are independent of one another and run in parallel. This means that the machinery can choose the next transaction among the provided types without restrictions.

Transaction Selector. This is the module that performs such a choice. It is responsible for mixing the transaction types in a workload according to the workload description. It uses two sources of information to make this decision: the Runtime Parameters and statistics.

The Runtime Parameters module. It is also a generated module. Its goal is to record the desired mix of transactions in the workload. For instance, in a workload like TPC-C, the transactions should be issued with a specified frequency. In this module, the frequency of different transaction types can be modified via the Console interactively. Any change is immediately reflected in the generated transaction stream of the workload.

Stats Manager. This module keeps track of past executed transactions. For each transaction type, the module maintains counters for successful and failed transactions, along with their latency. The counters are sampled frequently, and it is the result of this sampling that the Dashboard shows.

Modules for Packet Handling. The remaining modules of the DBMS Annihilator's runtime revolve around packet handling. The *Packetizer* module takes a transaction's parameters and generates the corresponding network packet to be issued against the DUT. The converse of that module is the *Response Handler*, which extracts execution information from transaction response packets sent by the server.

The Packetizer and Response Handler modules communicate directly with the network-related IP block. Simply put, the latter manages all the low-level aspects of the communication. Currently, the modules are hard-coded to issue and receive RoCE packets (RDMA over Converged Ethernet).

3 THE DEMONSTRATION

The demo consists of an instance of the DBMS Annihilator executing TPC-C against a high-performance database. We chose to use `NetSilo`, a version of the Silo [14] transaction engine that we modified to add RDMA support. Silo is based on an in-memory data structure that has been used as a baseline in numerous works (e.g., [3, 11]). Since our workload description language is still under development, we hand-code Verilog modules for the TPC-C workload in the way we expect the language compiler to produce them. The machine running Silo is powered by a Xeon 4216 CPU with 16 cores (32 threads) running at 2.1 GHz and with 192 GB of memory and a Mellanox ConnectX-5 100 Gbps network card. The client machine hosts the U50 loaded with the DBMS Annihilator logic, where the console and dashboard also run. Based on this setup, the demo allows a user to experience different scenarios, which we will describe next.

3.1 Scenario I: Interactive Console

We developed a GUI that allows the user to control and analyze a workload interactively. The GUI will be generated according to the workload description, i.e., it will adapt to the number and type of transactions, just as the workload generator. For the demonstration, we hand-coded the GUI to "pilot" a TPC-C workload. Figure 3 shows a screenshot of the GUI, which is built around two panels.

The first panel, on the bottom half of the screen, is the console. As discussed above, it controls the target workload throughput. The console has a widget for the target rate counter that can be adjusted up and down in 1 ktxn/s increments. When the user changes the counter, the DBMS Annihilator reacts immediately by adjusting its transaction rate. The console also has *slider* widgets, one per transaction type, representing the latter's relative frequency. In the beginning, the sliders are positioned according to the TPC-C-recommended transaction mix. The user can freely change the frequencies by moving the sliders, thus altering the workload. When a slider is moved, the other sliders compensate to ensure that the transaction frequencies add up to 100%.

The second panel, on the top half of the screen, is the dashboard. It plots the instantaneous transaction rate that the DUT achieves. The transaction rates are color-coded according to their type, and the curves are cumulative. This way, the user can simultaneously see the aggregated workload and its components. The dashboard has a mouse-over feature that shows workload details. This information appears in Figure 3 as gray labels. The dashboard also records changes in the workload rate, shown as black labels.

3.2 Scenario II: Scripted Console

The previous scenario allows interactive variations in the generated workload. Such interactivity is very useful for performance exploration, but sometimes, one may wish to execute a workload that is determined upfront. This second scenario shows how to execute such scripted workloads using the DBMS Annihilator.

When in scripted mode, the DBMS Annihilator follows the instructions contained in a script file. The script file can be developed offline by a programmer. It can also be recorded during an interactive session. The instructions are timed and reflect GUI manipulations that would be done had the session been interactive. The results shown by the dashboard can be saved and analyzed offline later.

In this scenario, we will explore two features. First, we will show how to change some workload characteristics on a millisecond scale, which is not possible manually. Second, we will execute the same script repeatedly against different database configurations and analyze which has the best performance impact.

3.3 Scenario III: Threshold Analysis

The scenarios above revolve around issuing workloads independent of how the DUT reacts to them. In this scenario, we are interested in adapting the workload according to the DUT's performance. For example, we may want to find the limits of a given DUT's concurrency control (CC) scheme. The idea is to increase some aspect of the workload, transaction contention in this case, until we see diminishing returns or performance regression. We can then adjust or experiment with different CC schemes and find how well they work for the workload.

In this scenario, we use a modified workload generator that creates a feedback loop. The Transaction Selector issues transactions with increasingly smaller intervals and monitors the transaction throughputs rather than their relative frequency. When the throughput stops increasing, the Selector stabilizes onto the maximum sustainable transaction rate.

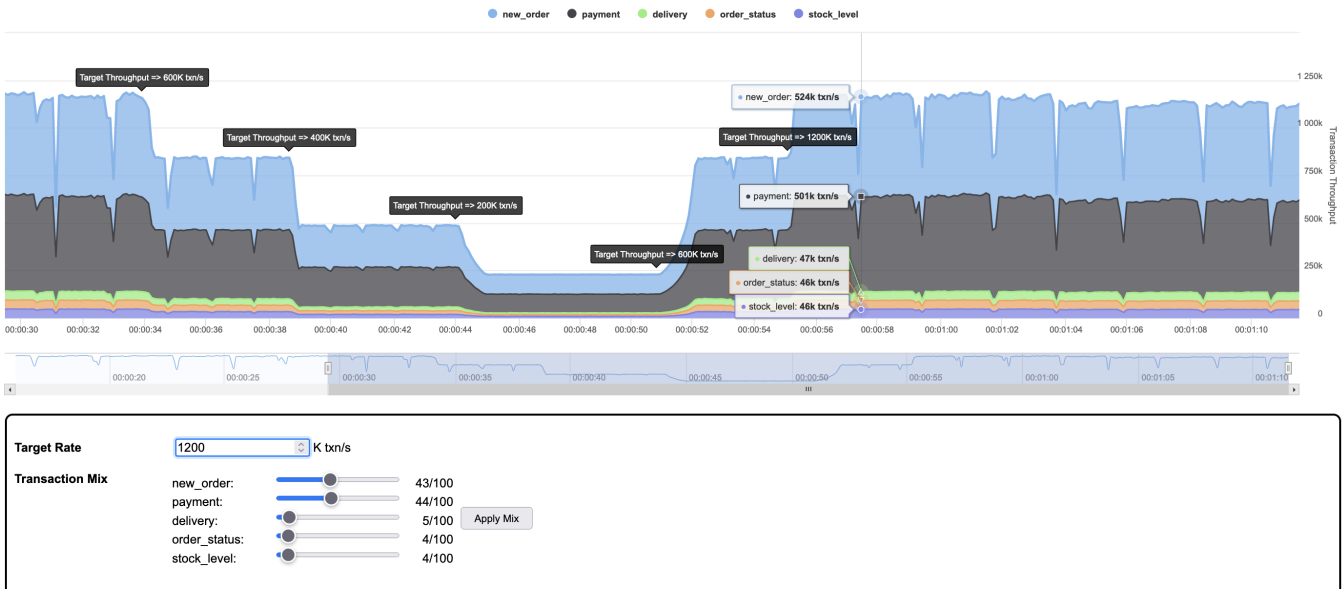

Figure 3: The Dashboard (top) shows how the database is performing and the Console (bottom) controls workload characteristics.

4 RELATED AND FUTURE WORK

The DBMS Annihilator combines a network traffic generator and a database benchmarking tool. Regarding traffic generators, a close tool to ours is Caliper [7]. It can inject packets with highly accurate intervals and, at the time of its publication, at high speeds. That system, however, is software-based, which would make it difficult to scale at current networking speeds. Regarding database benchmarking frameworks, a close system to ours is DIAMetrics [4]. The system can generate transaction requests and graphically analyze the database responses. It is also a software tool, and while it scales in size, it does so by forfeiting the precision that the network traffic generators can offer. In contrast, the DBMS Annihilator is unique in that it can generate highly-crafted workloads at a very high speed by leveraging hardware/software co-design.

There have been, however, other works that leverage networking hardware for transaction execution as opposed to generation. Regarding switch-based efforts, Transaction Triaging [10] rearranges the stream of transaction requests sent to the server in a way to foster performance while P4DB [9] offloads OLTP transaction processing for hot tuples onto a programmable switch, significantly improving latency as well. Regarding NIC-based efforts, D-RDMA [13] gives the card an active role in determining how data should be transferred into the network. We see an increasing number of works that, like ours, rethinks the division of functionality between hardware and software.

Moreover, we believe that our contributions can go beyond benchmarking. Applications are workload generators in a sense, but this time the feedback cycle, that is, the decision on which transaction to issue next, is not based on the database system performance. It comes from user interactions. We believe that our techniques can be used to replace the traditional software stack in database clients with an entirely new, accelerated communication stack. We intend to explore these advancements in an upcoming paper.

ACKNOWLEDGMENTS

We thank Sangjin Lee and André Ryser for their help setting up experiments. We also thank Alex Forencich for making Corundum [6] available and for the discussions about its inner workings. This work was partially funded by the German Research Foundation (DFG) under the grants BI2011/1 & BI2011/2 (DFG priority program 2037) and the DFG Collaborative Research Center 1053 (MAKI).

REFERENCES

[1] Alan Arondel. 2021. TX Burst and Ordered Packets. https://www.mail-archive.com/users@dpdk.org/msg05660.html.
[2] Robert Binna, Eva Zangerle, Martin Pichl, Günther Specht, and Viktor Leis. 2018. HOT: A Height Optimized Trie Index for Main-Memory Database Systems *(SIGMOD '18)*.
[3] Badrish Chandramouli, Guna Prasaad, Donald Kossmann, Justin Levandoski, James Hunter, and Mike Barnett. 2018. FASTER: A Concurrent Key-Value Store with In-Place Updates *(SIGMOD '18)*.
[4] Shaleen Deep, Anja Gruenheid, Kruthi Nagaraj, Hiro Naito, Jeff Naughton, and Stratis Viglas. 2021. DIAMetrics: Benchmarking Query Engines at Scale. *SIGMOD Rec.* 50, 1 (2021), 24–31.
[5] DPDK [n.d.]. Data Plane Developemnt Kit. https://dpdk.org/.
[6] Alex Forencich, Alex C. Snoeren, George Porter, and George Papen. 2020. Corundum: An Open-Source 100-Gbps NIC. In *28th IEEE International Symposium on Field-Programmable Custom Computing Machines*.
[7] Manya Ghobadi, Geoffrey Salmon, Yashar Ganjali, Martin Labrecque, and J. Gregory Steffan. 2012. Caliper: Precise and Responsive Traffic Generator. In *2012 IEEE 20th Annual Symposium on High-Performance Interconnects*.
[8] Infiniband Architecture Specifications [n.d.]. Infiniband Architecture Specification. https://www.infinibandta.org/ibta-specifications-download/.
[9] Matthias Jasny, Lasse Thostrup, Tobias Ziegler, and Carsten Binnig. 2022. P4DB - The Case for In-Network OLTP *(SIGMOD '22)*.
[10] Theo Jepsen, Alberto Lerner, Fernando Pedone, Robert Soulé, and Philippe Cudré-Mauroux. 2021. In-Network Support for Transaction Triaging. *Proc. VLDB Endow.* (2021), 1626–1639.
[11] Viktor Leis, Alfons Kemper, and Thomas Neumann. 2013. The adaptive radix tree: ARTful indexing for main-memory databases *(ICDE '13)*.
[12] Juniper Networks. [n.d.]. How many Packets per Second per port are needed to achieve Wire-Speed? https://kb.juniper.net/InfoCenter/index?page=content&id=kb14737.
[13] André Ryser, Alberto Lerner, Alex Forencich, and Philippe Cudré-Mauroux. 2022. D-RDMA: Bringing Zero-Copy RDMA to Database Systems *(CIDR '22)*.
[14] Stephen Tu, Wenting Zheng, Eddie Kohler, Barbara Liskov, and Samuel Madden. 2013. Speedy Transactions in Multicore In-Memory Databases *(SOSP '13)*.