# A Data-oriented Transaction Execution Engine and Supporting Tools

Ippokratis Pandis[†‡] Pınar Tözün[‡] Miguel Branco[‡] Dimitris Karampinas[#] Danica Porobic[‡]
Ryan Johnson[*] Anastasia Ailamaki[‡†]

[†]Carnegie Mellon University, Pittsburgh, PA, USA
[‡]EPFL, Lausanne, VD, Switzerland
[#]University of Patras, Rio, Greece
[*]University of Toronto, Toronto, ON, Canada

## ABSTRACT

Conventional OLTP systems assign each transaction to a worker thread and that thread accesses data, depending on what the transaction dictates. This *thread-to-transaction* work assignment policy leads to unpredictable accesses. The unpredictability forces each thread to enter a large number of critical sections for the completion of even the simplest of the transactions; leading to poor performance and scalability on modern manycore hardware.

This demonstration highlights the chaotic access patterns of conventional OLTP designs which are the source of scalability problems. Then, it presents a working prototype of a transaction processing engine that follows a non-conventional architecture, called data-oriented or DORA. DORA is designed around the *thread-to-data* work assignment policy. It distributes the transaction execution to multiple threads and offers predictable accesses. By design, DORA can decentralize the lock management service, and thereby eliminate the critical sections executed inside the lock manager. We explain the design of the system and show that it more efficiently utilizes the abundant processing power of modern hardware, always contrasting it against the conventional execution. In addition, we present different components of the system, such as a dynamic load balancer. Finally, we present a set of tools that enable the development of applications that use DORA.

## Categories and Subject Descriptors

H.2.4 [Database Management]: Systems - transaction processing.

## General Terms

Management, Performance, Design.

## Keywords

Data-oriented transaction execution, DORA, Runtime re-balancing, Transaction flow graph generation, Partitioning.

## 1. INTRODUCTION

The emergence of manycore processors as the dominant processor technology has several implications in the design of database management systems [3]. On-line transaction processing (OLTP), as one of the most important and also complicated database management applications, is greatly affected by the emergence of this new technology.


*SIGMOD'11*, June 12–16, 2011, Athens, Greece.
ACM 978-1-4503-0661-4/11/06.

The majority of the transaction processing systems were designed in an era during which most computers were uniprocessors with high latency I/O subsystems. Therefore, such systems feature high *concurrency* –support for multiple in progress operations– to interleave the execution of a large number of transactions, most of which are idle waiting for I/O completions or processor cycles, at any given moment. As the number of processing cores per processor increases, in step with Moore's law, the transaction processing systems need to exhibit equally high *execution parallelism* –support for concurrent operations to proceed simultaneously. Unfortunately, internal serializations often prevent the conventional transaction processing systems to translate the high concurrency of transactional workloads to proportionally high execution parallelism and performance [5].

The conventional transaction processing systems follow a *thread-to-transaction* policy of assigning work to threads. Each incoming transaction is assigned to a worker thread and it is the transaction that dictates what data that worker thread will access. The random nature of the requests for the majority of transaction workloads leads to unpredictable accesses. Hence, each thread has to acquire logical locks through a centralized lock manager and enter a large number of critical sections, even for the completion of simple transactions; eventually leading to poor performance and scalability due to contention inside the lock manager.

### 1.1 Data Oriented Architecture (DORA)

To address the growing bottleneck presented by the conventional transaction execution, we have designed a system around a *thread-to-data* assignment of work policy [6]. The, so called, DORA design takes the novel approach of breaking each transaction to smaller requests and sending work to where the data resides in the system. By concentrating related requests for data accesses together, the system is able to coordinate them efficiently and in a largely distributed fashion.

DORA decomposes the database into logical partitions. The partitioning is enforced by a set of *routing rules,* one per table. Upon a reception of a new transaction, DORA decomposes it into a set of actions based on the routing rules and the data this transaction intends to access.

Similar to the operator-centric approach used by QPipe query processor [2], DORA creates largely independent micro-engines, or worker threads, each of which handles all requests for a partition of the data from the database. Each worker thread receives actions and executes them in a sequential fashion while maintaining a private lock table. The lock table enforces consistency among concurrent actions: it only allows actions that have no conflicting accesses to complete. Thus, it guarantees that an action that can execute and complete legally according to the local lock table, can also execute without any restriction and complete in the scope of the entire database.

Hence, when the worker thread executes an action, it bypasses the centralized lock manager.

The threads that serve actions on behalf of the same request share common objects, called the rendezvous points (RVPs), which are initialized to the number of threads that have to report to them. Each thread serves requests in isolation and updates the counter of the corresponding RVP upon completion. The last thread to report on a rendezvous point decides whether the corresponding transaction should commit or abort, or whether a new set of actions needs to be submitted to the system. Upon the system-wide completion of the transaction, each worker thread that participated in its execution updates the local lock tables, possibly allowing actions on behalf of other transactions to proceed.

Finally, unlike traditional shared-nothing approaches (e.g., [7]), the partitions in DORA are purely logical and serve only to distribute requests evenly throughout the system. There is no need for executing distributed transactions, with all the accompanying overheads [4][1]. When changes in load or access patterns result in an imbalance, DORA easily re-partitions the database, adapting to the change.

Further details about DORA can be found at [6].

## 1.2 Demonstration

This system demonstration exposes the key novel features of DORA and also provides an intuitive way of visualizing the effects of varying load, access patterns, and partitioning decisions within the database engine. Traditional engines are typically demonstrated as "black boxes," due to tight integration of system components. DORA, on the other hand, naturally breaks up transaction processing to independent actions and has different worker threads to execute each action, allowing for a visually appealing demonstration.

We implemented a DORA prototype on top of Shore-MT [5], a storage manager developed specifically to take advantage of manycore hardware. Shore-MT provides the access methods and the buffer pool, logging, and recovery services. The modifications needed in Shore-MT were minimal.

The demonstration presents two main components:

**Live Systems:** A fully interactive demonstration of a live conventional system and a DORA prototype running on identical manycore hardware. The user will be able to modify system parameters and workload characteristics, including the number of hardware contexts available to the system, number of clients, transaction mix, and skewness of data accesses. They will be able to watch various system statistics and compare the behavior of the two systems, noticing the contrast between the unpredictable accesses of the conventional system and the predictable ones by DORA.

**Designer:** We exhibit two tools which can be used for the development of applications over partitioning-based systems, such as DORA. The first tool is a semi-automated transaction plan generator for DORA. The user can input arbitrary transactions (in SQL text), see the generated execution plans, modify and run them. The second tool is for automating the physical design. Given a workload, this tool, suggests a set of indexes along with the partitioning scheme in order to balance the load across the system and minimize the non-partitioning aligned accesses.

All stories are demonstrated through pre-designed scenarios. We also have traces of the scenarios, to be able to demonstrate them even in case of connectivity problems with the server. Next we will describe in detail the demonstration story line and conclude with the "take-away" message for this demonstration.

# 2. STORY LINE

The demonstration consists of three parts. The first part underlines the problems of conventional OLTP designs, and introduces the viewer to the concept of DORA. Next, through a monitor that is connected to a live conventional system and a DORA prototype, the viewer can modify multiple workload parameters and see how the two systems behave. The third part presents two developer support tools for partitioning-based OLTP systems; one that can be used for the semi-automated generation of transactions and the other one for automating the physical design.

## 2.1 Part 1: Introduction

A poster and a set of slides are used to highlight the inherent scalability problems of conventional OLTP designs and introduce the viewer to the concept of data-oriented execution. We highlight key features of the system and illustrate how DORA handles challenges that arise while processing transactions. We show how a single transaction is broken to a set of actions (which can proceed in parallel) and rendezvous points, and executed. For multiple transactions we show how actions are queued up in their partition and how the partition "owner" worker thread executes them by decentralizing the lock manager, while isolation is maintained.

## 2.2 Part 2: Live Systems

This part of the demonstration will use a graphical user interface (as in Figure 1) which monitors, through a socket interface, a live conventional system and the DORA prototype running on identical machines. Both systems are build over the same storage engine (Shore-MT).

The main system overview panel provides real-time statistics about the two systems. The DORA monitor shows the throughput and utilization of the different worker threads (micro-engines) along with the data partitioning information which dynamically changes, as DORA adjusts to reflect workload changes. A workload panel allows the user to modify load parameters, such as number of clients, the mix of transactions to execute, and the distribution of data accesses. Below we describe the main exhibited scenarios.

**Access Patterns.** The user decides between pre-defined workloads (e.g., TATP and TPC-C) and can see the accesses made to the database records of each table by the various threads in the two systems, as shown in Figure 1. The accesses of the conventional system are random while in DORA there is a predictable order.

**Performance Under Varying Load.** The user can vary the hardware resources the two systems may utilize, as well as other workload parameters, such as the number of clients and their think times. We demonstrate how DORA maintains high performance as load varies from idle to saturated or to oversubscribed. During idle execution, DORA exploits intra-transaction parallelism to improve response times and machine utilization; during busy periods, the system benefits from reduced overhead by avoiding centralized lock management; during oversubscribed periods, the queues naturally impose a form of admission control that allows the system to maintain peak throughput long after the traditional system begins to lag.

**Load balancing.** We present two load balancing components through a load and accesses overview panel. The first component observes the action queues of each worker thread and re-partitions, reducing the load of threads whose input queue is long, while merging partitions of the threads whose action queues are not loaded. In the demo, the user can select from several types of distribution functions to specify the kind of skew and also slide it

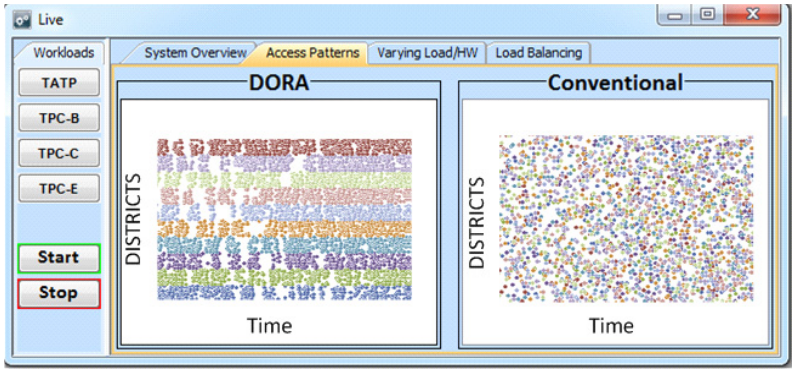


**Figure 1. The monitor of the live systems**

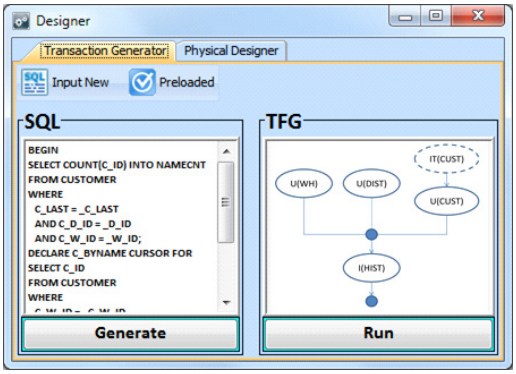


**Figure 2. The DORA designer**

around to vary the locations of "hot spots." With each change, DORA adapts its partitioning, in real-time, to maintain the highest possible performance.

A second component aims to reduce non-partitioning aligned accesses, which significantly reduce the performance of DORA. When it observes a rapid increase in the number of non-partition aligned accesses, it suggests adjusting the partitions based on the fields that are most frequently used making most of the accesses aligned to the partitioning again. For this part, the user starts executing a transaction which accesses fields other than those used for the partitioning, reducing performance. The increasing frequency of non-partitioning aligned accesses will be observed and the tool will suggest to re-organize the partitioning scheme according to the new access field for improving performance.

## 2.3 Part 3: Designer

We have developed a set of tools that facilitate the implementation of transactions and the physical design of partitioning-based OLTP systems. We use a second graphical user interface (shown in Figure 2) to demonstrate those tools.

**Semi-automated transaction generation.** DORA decomposes transactions into a set of actions and rendezvous points (RVPs), which are placed between actions with data dependencies that need to execute sequentially. The graph of actions and RVPs constitute the flow graph of the transaction.

This tool automatically generates transaction flow graphs and allows the user to modify them (e.g., selecting to run actions in parallel or serially), as long as the data dependencies allow. Modifying the order of some actions can be useful, for example, in the case of actions with high abort frequency. In the demo, as shown in Figure 2, the user can input arbitrary transactions in SQL, see the generated execution plans, modify and run them.

**Semi-automated physical design.** DORA applies logical-only partitioning which makes it highly flexible to modify. This tool, given a workload, tries to balance the load across the system and minimize the non-partitioning aligned accesses. The user inputs a workload as a set of transactions with their expected execution frequencies. Then, the tool suggests a partitioning scheme to reduce the need of re-partitioning at run time. The suggestions include which fields should be used for partitioning each table, the number of partitions, and the size of each partition. Also, it proposes the physical design of the indexes to be used. For example, it may propose to prepend a column in an index, which initially wouldn't have that column, because this column is used for the partitioning and prepending this column would reduce the number of non-partitioning aligned accesses. This tool can be employed for the development of applications for other partitioning-based OLTP systems (such as [7]) as well.

# 3. "TAKE-AWAY" MESSAGE

This demonstration highlights the inherent scalability limitations of conventional OLTP designs, stemming from their unpredictable access patterns. Then, it presents a working prototype of a transaction processing engine that follows the principles of a non-conventional data-oriented architecture, which offers predictability in the data accesses. This change in the architecture allows to decentralize services, such as the lock management, which typically require the execution of contented critical sections and thereby are performance bottlenecks. We explain the design of DORA and present a set of tools that enable the development of applications that use it. We show that this logical partitioning based design achieves the goal of improved performance and scalability, while it is capable of reacting to load imbalances quickly and the development of applications for it is simple.

# 4. ACKNOWLEDGEMENTS

This work was partially supported by Sloan research fellowship, NSF grants CCR-0205544, IIS-0133686, and IIS-0713409, an ESF EurYI award, and SNF funds.

# 5. REFERENCES

[1] C. Curino, E. Jones, Y. Zhang, and S. Madden. "Schism: a workload-driven approach to database replication and partitioning." In *VLDB*, 2010.

[2] K. Gao, S. Harizopoulos, I. Pandis, V. Shkapenyuk, and A. Ailamaki. "Simultaneous pipelining in QPipe: exploiting work sharing opportunities across queries." In *ICDE*, 2006.

[3] N. Hardavellas, I. Pandis, R. Johnson, N. Mancheril, A. Ailamaki, and B. Falsafi. "Database servers on chip multi-processors: limitations and opportunities." In *CIDR*, 2007.

[4] P. Helland. "Life beyond distributed transactions: an apostate's opinion." In *CIDR*, 2007.

[5] R. Johnson, I. Pandis, N. Hardavellas, A. Ailamaki, and B. Falsafi. "Shore-MT: a scalable storage manager for the multicore era." In *EDBT*, 2009.

[6] I. Pandis, R. Johnson, N. Hardavellas, and A. Ailamaki. "Data-oriented transaction execution." *PVLDB* 3(1), 2010.

[7] M. Stonebraker, S. Madden, D. Abadi, S. Harizopoulos, N. Hachem, and P. Helland. "The end of an architectural era (it's time for a complete rewrite)." In *VLDB*, 2007.