SIGMOD 2020

TECHNISCHE UNIVERSITÄT DARMSTADT

THE BEST RUN SAP

ROBUST PERFORMANCE OF MAIN MEMORY DATA STRUCTURES BY CONFIGURATION




Norman May
SAP SE

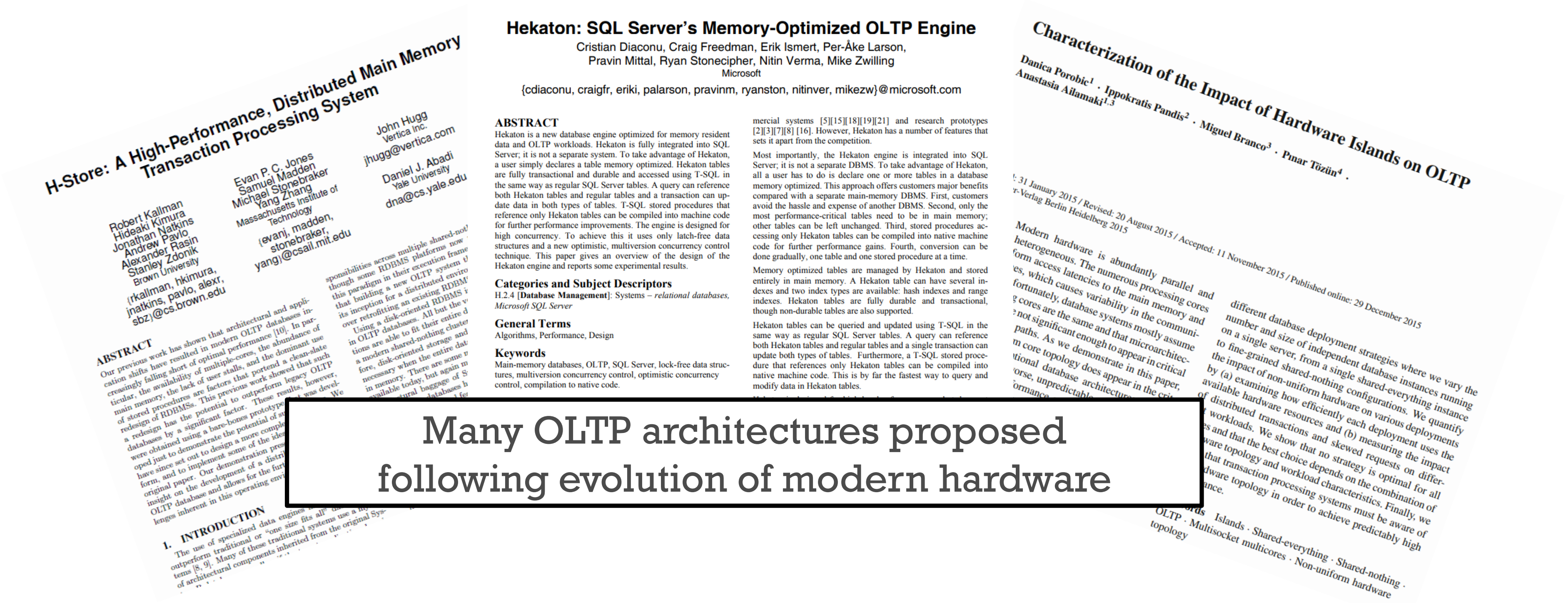
Ismail Oukid
(SAP SE)

Ilia Petrov
Reutlingen University

Carsten Binnig
TU Darmstadt

Tiemo Bang
TU Darmstadt & SAP SE

PLETHORA OF MODERN OLTP ARCHITECTURES


H-Store: A High-Performance, Distributed Main Memory Transaction Processing System

Robert Kallman, Hideaki Kimura, Jonathan Natkins, Andrew Pavlo, Alexander Rasin, Stanley Zdonik
Brown University
{rkallman, hkimura, jnatkins, pavlo, alexr, sbz}@cs.brown.edu

Evan P. C. Jones, Samuel Madden, Michael Stonebraker, Yang Zhang
Massachusetts Institute of Technology
{evanj, madden, stonebraker, yang}@csail.mit.edu

John Hugg
Vertica Inc.
jhugg@vertica.com

Daniel J. Abadi
Yale University
dna@cs.yale.edu

ABSTRACT

1. INTRODUCTION

Hekaton: SQL Server's Memory-Optimized OLTP Engine

Cristian Diaconu, Craig Freedman, Erik Ismert, Per-Åke Larson, Pravin Mittal, Ryan Stonecipher, Nitin Verma, Mike Zwilling
Microsoft
{cdiaconu, craigfr, eriki, palarson, pravinm, ryanston, nitinver, mikezw}@microsoft.com

ABSTRACT

Hekaton is a new database engine optimized for memory resident data and OLTP workloads. Hekaton is fully integrated into SQL Server; it is not a separate system. To take advantage of Hekaton, a user simply declares a table memory optimized. Hekaton tables are fully transactional and durable and accessed using T-SQL in the same way as regular SQL Server tables. A query can reference both Hekaton tables and regular tables and a transaction can update data in both types of tables. T-SQL stored procedures that reference only Hekaton tables can be compiled into machine code for further performance improvements. The engine is designed for high concurrency. To achieve this it uses only latch-free data structures and a new optimistic, multiversion concurrency control technique. This paper gives an overview of the design of the Hekaton engine and reports some experimental results.

Categories and Subject Descriptors

H.2.4 [Database Management]: Systems – *relational databases, Microsoft SQL Server*

General Terms

Algorithms, Performance, Design

Keywords

Main-memory databases, OLTP, SQL Server, lock-free data structures, multiversion concurrency control, optimistic concurrency control, compilation to native code.

mercial systems [5][15][18][19][21] and research prototypes [2][3][7][8] [16]. However, Hekaton has a number of features that sets it apart from the competition.

Most importantly, the Hekaton engine is integrated into SQL Server; it is not a separate DBMS. To take advantage of Hekaton, all a user has to do is declare one or more tables in a database memory optimized. This approach offers customers major benefits compared with a separate main-memory DBMS. First, customers avoid the hassle and expense of another DBMS. Second, only the most performance-critical tables need to be in main memory; other tables can be left unchanged. Third, stored procedures accessing only Hekaton tables can be compiled into native machine code for further performance gains. Fourth, conversion can be done gradually, one table and one stored procedure at a time.

Memory optimized tables are managed by Hekaton and stored entirely in main memory. A Hekaton table can have several indexes and two index types are available: hash indexes and range indexes. Hekaton tables are fully durable and transactional, though non-durable tables are also supported.

Hekaton tables can be queried and updated using T-SQL in the same way as regular SQL Server tables. A query can reference both Hekaton tables and regular tables and a single transaction can update both types of tables. Furthermore, a T-SQL stored procedure that references only Hekaton tables can be compiled into native machine code. This is by far the fastest way to query and modify data in Hekaton tables.

Characterization of the Impact of Hardware Islands on OLTP

Danica Porobic[1] · Ippokratis Pandis[2] · Miguel Branco[3] · Pınar Tözün[4] · Anastasia Ailamaki[1,3]

Keywords Islands · Shared-everything · Shared-nothing · OLTP · Multisocket multicores · Non-uniform hardware topology

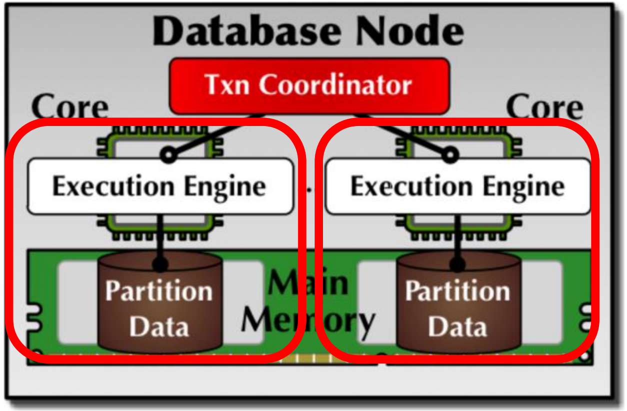
Many OLTP architectures proposed following evolution of modern hardware

TECHNISCHE UNIVERSITÄT DARMSTADT

THE BEST RUN SAP

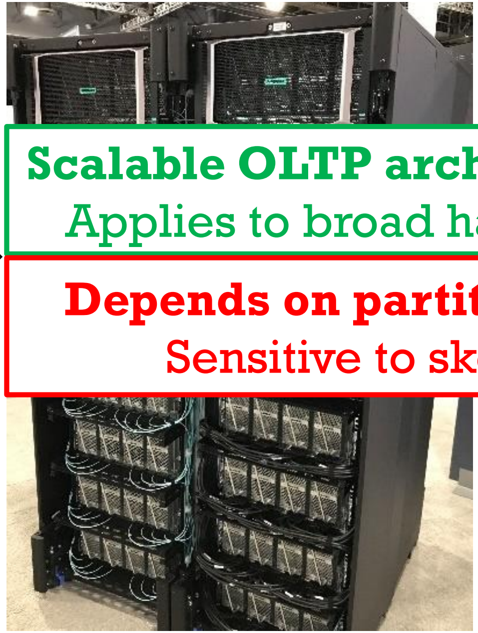
CANDIDATE 1: H-STORE

Kallman et al.: Fine-grained shared-nothing architecture[1]



Partitioning per core

Gen-Z Sandbox: 2000 cores, 200 TB memory[3]



Scalable OLTP architecture:
Applies to broad hardware

Depends on partitioning:
Sensitive to skew!

[1] Kallman et al. 2008. H-Store: a High-Performance, Distributed Main Memory Transaction Processing System. Proc. VLDB Endow.
[2] Pavlo. 2011. Magical Parallel OLTP Databases. https://hstore.cs.brown.edu/slides/hstore-houdini-nov2011.pdf
[3] Burts. 2018. HPE BOOTS UP SANDBOX OF THE MACHINE FOR EARLY USERS. https://www.nextplatform.com/2018/06/21/hpe-boots-up-sandbox-of-the-machine-for-early-users/

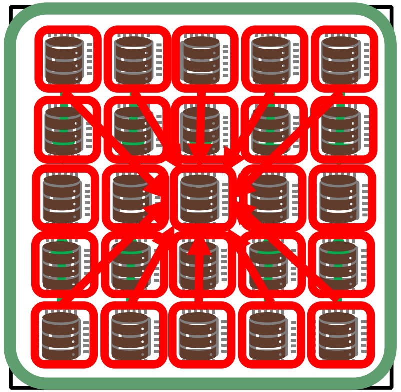
CANDIDATE 2: HEKATON

Diaconu et al.: Efficient shared-everything architecture[4]

Joint operation by all resources on all data:

+ Non-partitionable workloads
+ Fluctuating workloads
- Physical contention
- NUMA effects

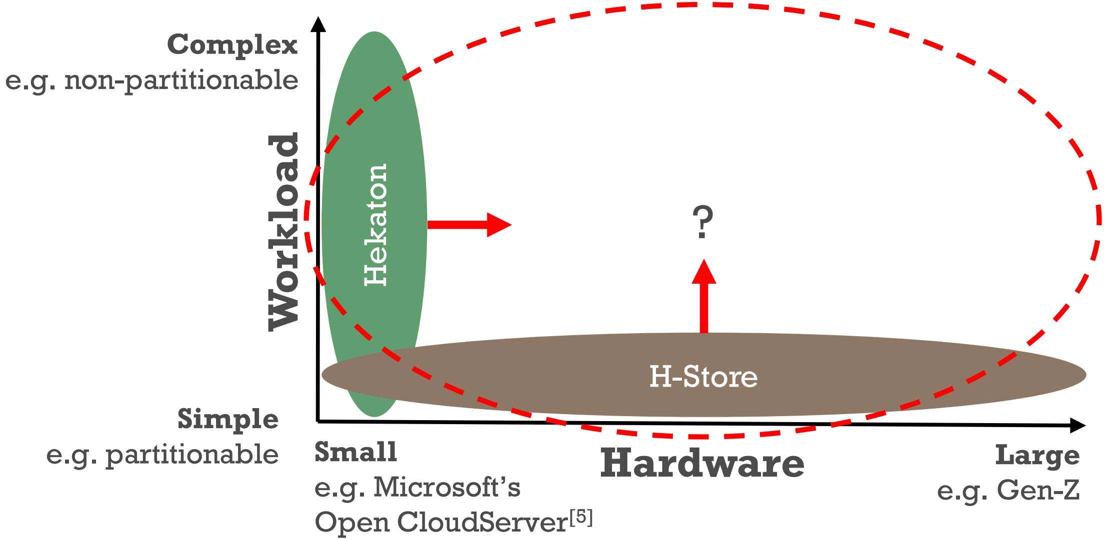
E.g.: TPC-C on many-core hardware

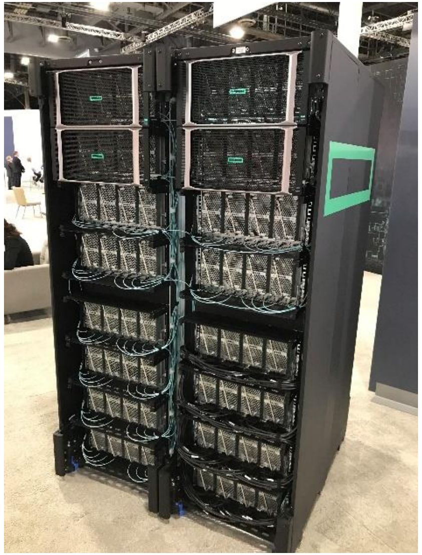


Few warehouses

[4] Diaconu et al. .2013. Hekaton: SQL Server's Memory-Optimized OLTP Engine. SIGMOD.

DESIGN SPACE OF OLTP ARCHITECTURES





[3]

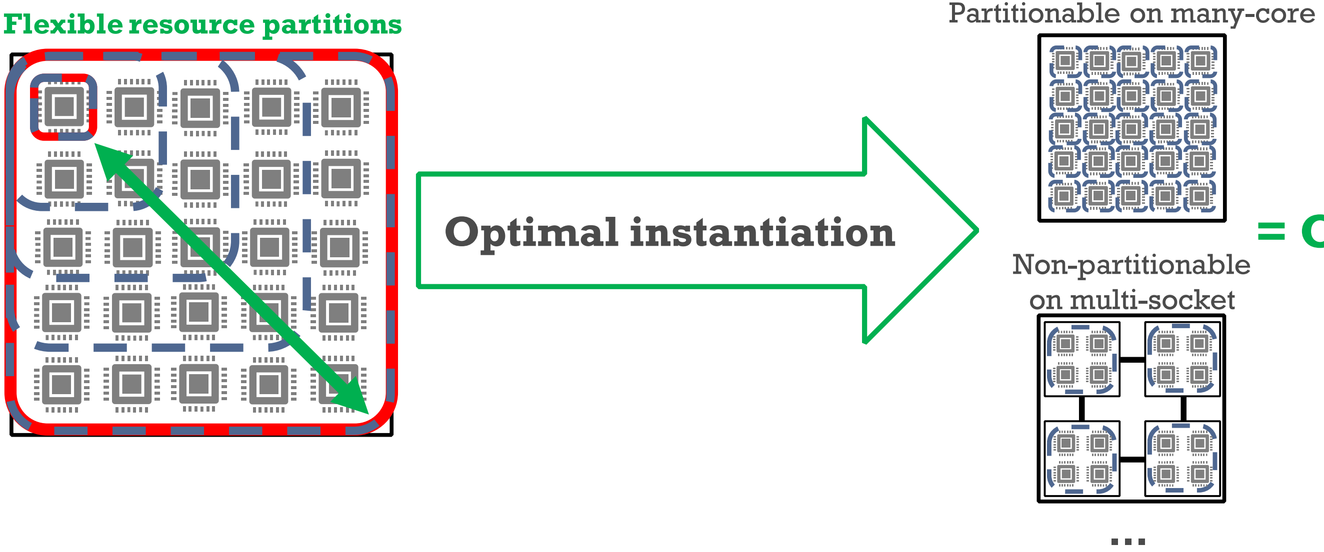
➢ **How to achieve robust performance for entire design space?**

[5] Drake et al. 2015. Microsoft's Open Cloud Server. http://download.microsoft.com/download/B/1/7/B179029E-7AE8-447A-B8C9-B823B3DFC727/Microsofts_Open_CloudServer_Strategy_Brief.pdf

IDEA:
CONFIGURE OLTP ARCHITECTURE

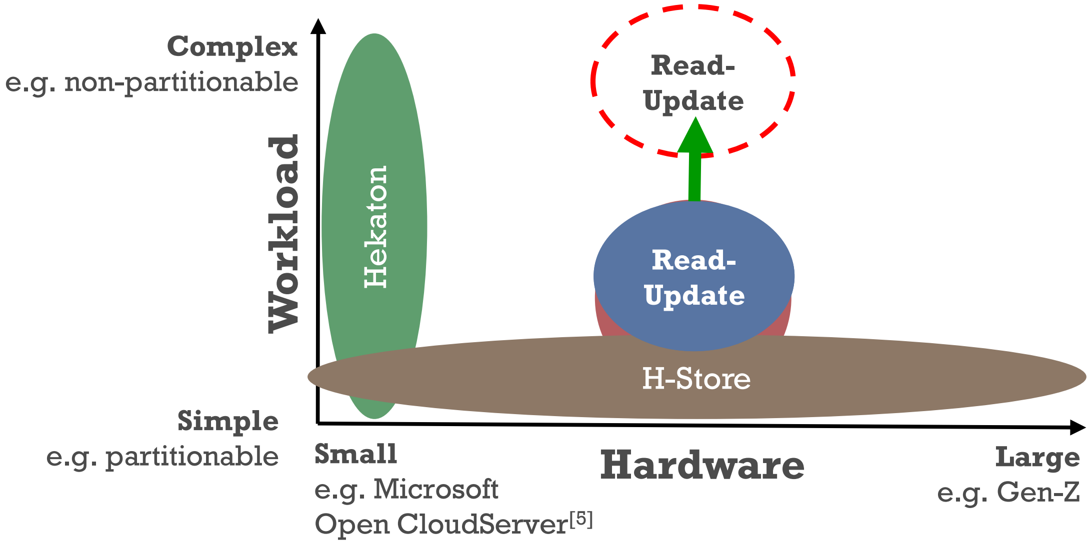
One size does not fit entire design space!

Simply resize to fit point in design space!

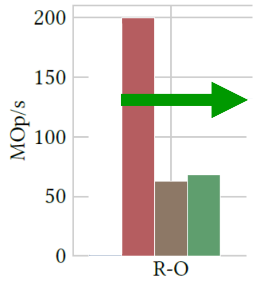


= Optimal architecture for any workload on any hardware

EXAMPLE: CONFIGURATION FOR WORKLOADS



Throughput of YCSB workloads on 8-socket hardware



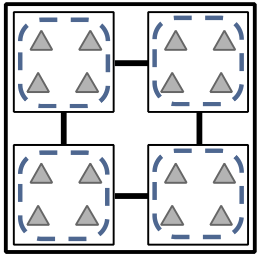
➢ **Simply optimise architecture for given workload!**

CONFIGURATION OPPORTUNITIES

Configuration = flexible resource partitions + optimal instantiation

Existing architectures

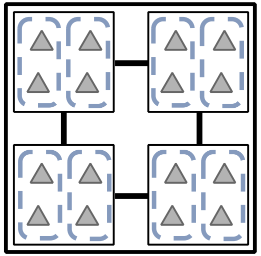
Mimic:



e.g. NUMA-aware for read-only workload

Fitting partitioning

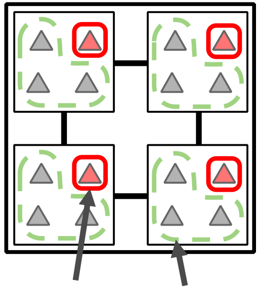
Optimally sized:



e.g. reduce contention for read-update workload

For discrete components

Individually sized:

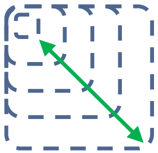


e.g. H-Store for hot + NUMA-aware for cold indexes in single architecture

➢ **Configure optimal DBMS architecture without redesign!**

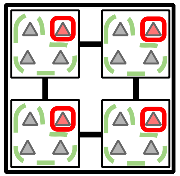
CONFIGURATION APPROACH

Configurable Virtual Domains



Virtual resource partition +
Contention and locality **Domain**

Configuration procedure

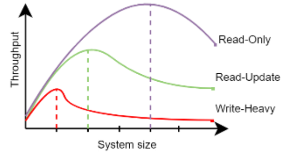


Linear program
instantiates virtual domains
and assigns data structures

Deploy on runtime

Establish efficient operation
between partitions…

Initial Calibration



Capture general behaviour
of data structures

Actual workload and hardware

E.g. hot and cold indexes
on 4-socket server

[6] Chiarandini. Linear and Integer Programming Lecture Notes. https://imada.sdu.dk/~marco/Teaching/AY2014-2015/DM554/Notes/dm554-main.pdf

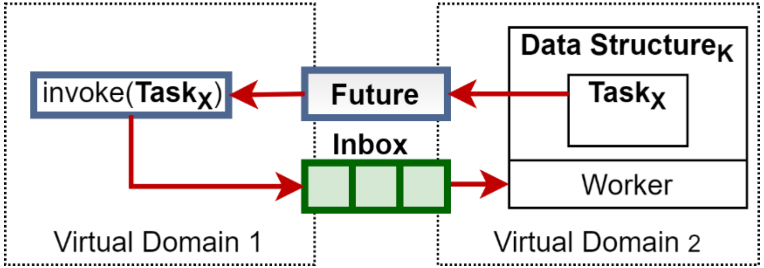
CONFIGURATION RUNTIME

Allow efficient execution across individually optimised partitions

Access partitions via Data-aware Task:

- **without interfering**
- **with maximal utilisation**
- **with minimal overhead**

Delegation + **async. execution** of **Data-aware Tasks**

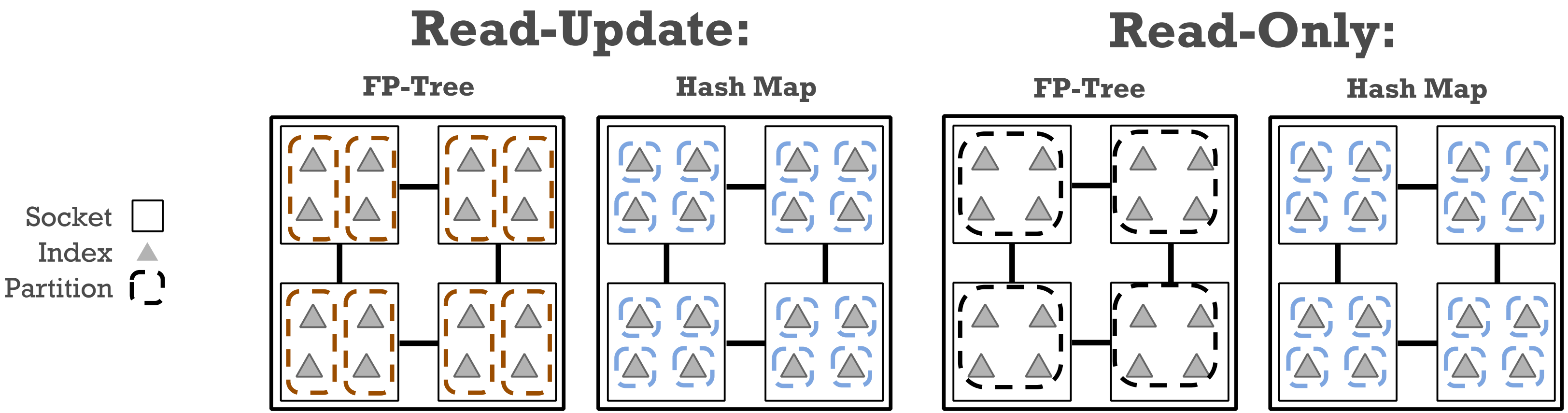


In-memory Messaging

➢ Robust performance by efficient execution across optimal partitions!

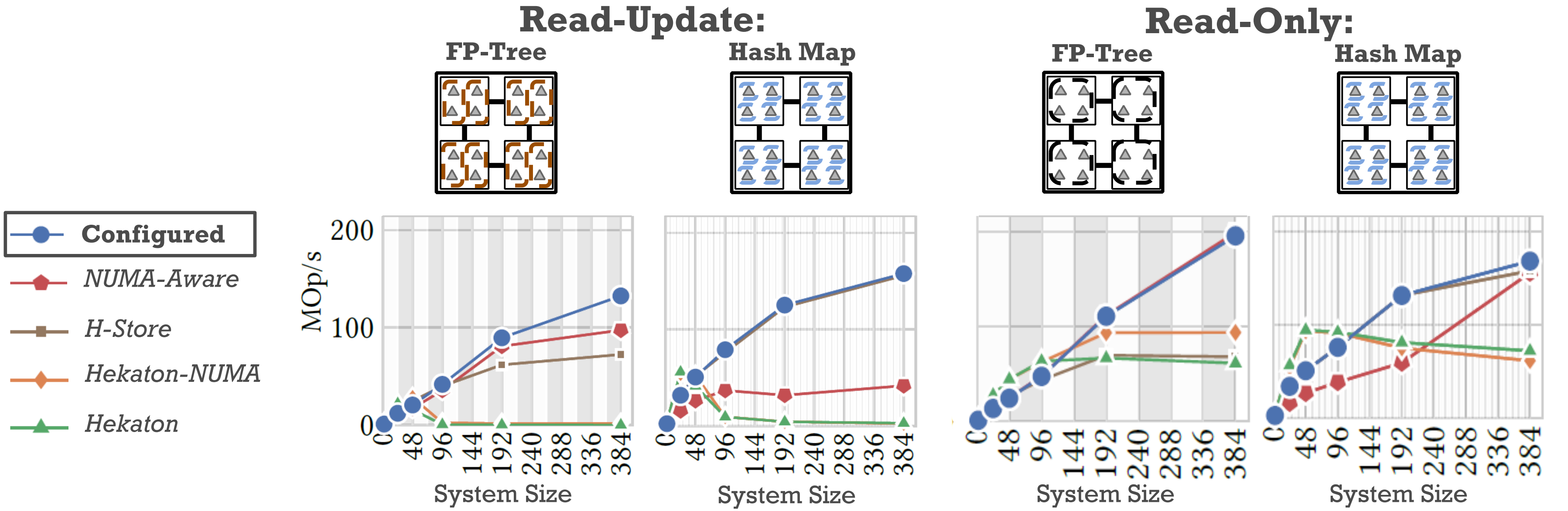
EVALUATION: SIMPLE WORKLOAD (YCSB)

Configuration of 2 Key-Value Stores for differing YCSB workloads



EVALUATION: SIMPLE WORKLOAD (YCSB)

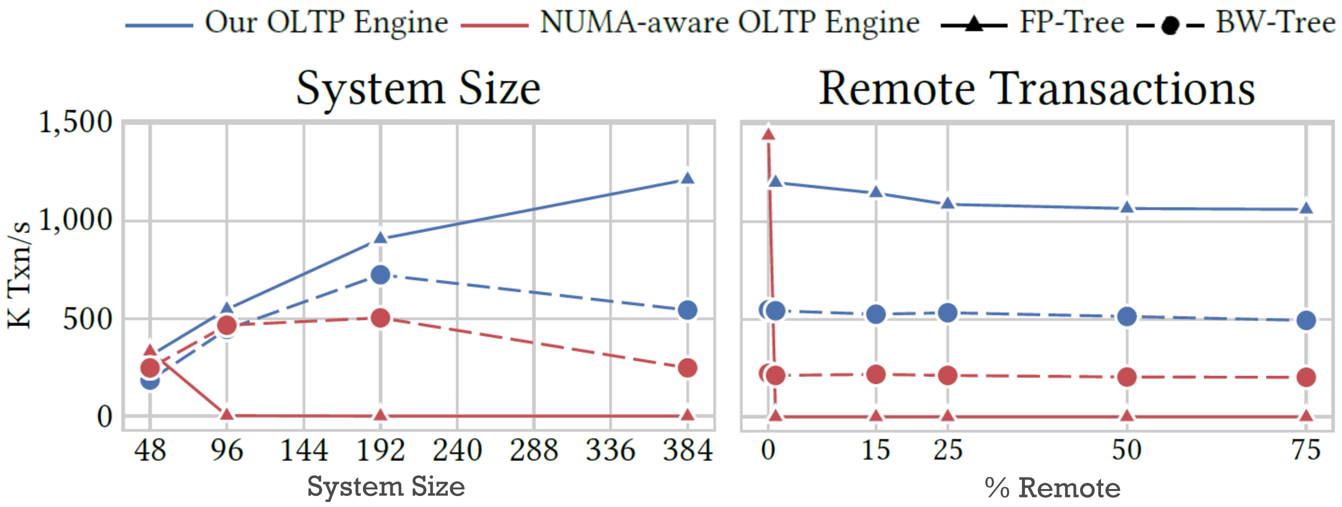
Throughput under increasing system size for differing workload



➢ **Robust across workloads, system sizes & data structures!**

EVALUATION: COMPLEX WORKLOAD (TPC-C)

Configuration for non-partitionable TPC-C workload, 8 warehouses
(*details in the paper*)



➢ **Robust performance for complex workload by configuration!**

ALL DETAILS…

Robust Performance of Main Memory Data Structures by Configuration

Tiemo Bang*
TU Darmstadt & SAP SE

Ismail Oukid†
Snowflake Inc.

Norman May
SAP SE

Ilia Petrov
Reutlingen University

Carsten Binnig
TU Darmstadt

Abstract

In this paper, we present a new approach for achieving robust performance of data structures making it easier to reuse the same design for different hardware generations but also for different workloads. To achieve robust performance, the main idea is to strictly separate the data structure design from the actual strategies to execute access operations and adjust the actual execution strategies by means of so-called configurations instead of hard-wiring the execution strategy into the data structure. In our evaluation we demonstrate the benefits of this configuration approach for individual data structures as well as complex OLTP workloads.

ACM Reference Format:

Tiemo Bang, Ismail Oukid, Norman May, Ilia Petrov, and Carsten Binnig. 2020. Robust Performance of Main Memory Data Structures by Configuration. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD'20), June 14–19, 2020, Portland, OR, USA*. ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3318464.3389725

1 Introduction

Motivation: Within the last decade, we have seen different hardware trends that significantly affected the design of single-node database systems: (1) Increases in main-memory capacities made it possible to hold even larger data sets secondary storage (e.g., hard drives). (2) Moore's Law and Dennard Scaling required processor designers to move from single-socket and single-core designs to multi-socket and multi-core designs. As a result of these trends, we have seen a rapid evolution of hardware designs differing in essential characteristics not only memory capacities but also the underlying topology of how cores and memory are connected as well as cache sizes and coherence protocols.

A considerable body of existing work in DBMS research has thus focused on optimising the design of core DBMS data structures such as indexes for specific hardware configurations and workloads. For example, there have been various design alternatives proposed for classical B-trees to adapt them to modern memory hierarchies and make them more cache-conscious for read-heavy workloads [33, 34] or to optimise their behaviour for high-contention scenarios [25] under write-heavy workloads. A significant issue with this manual tailoring of core DBMS data structures is that not only their redesign involves high effort and reintegration into the DBMS but also that a design optimal for one hardware generation and one workload might induce severe performance degradation on another hardware generation when underlying assumptions change.

An alternative to this approach is designing data structures that can provide robust performance [18]. At its core, robust

https://doi.org/10.1145/3318464.3389725

CONCLUSION

- **Optimise DBMS architecture without redesign by configuration!**

https://doi.org/10.1145/3318464.3389725

Stay healthy and see you next year in person!