Developer life in academia & research

Tips and examples from PhD in databases

Iraklis Psaroudakis, 13/02/25, DevStaff.gr talk

About

Iraklis Psaroudakis is working remotely from Greece as a Principal Software Engineer at Elastic, focusing on the distributed scalability of Elasticsearch.

Industry 3y

Previously, he was a Principal Member of Technical Staff at Oracle Labs in Zürich, Switzerland, concentrating on parallel and distributed programming, and graph-based analytical & machine learning workloads, especially in financial crime & compliance applications.

R&D 6y

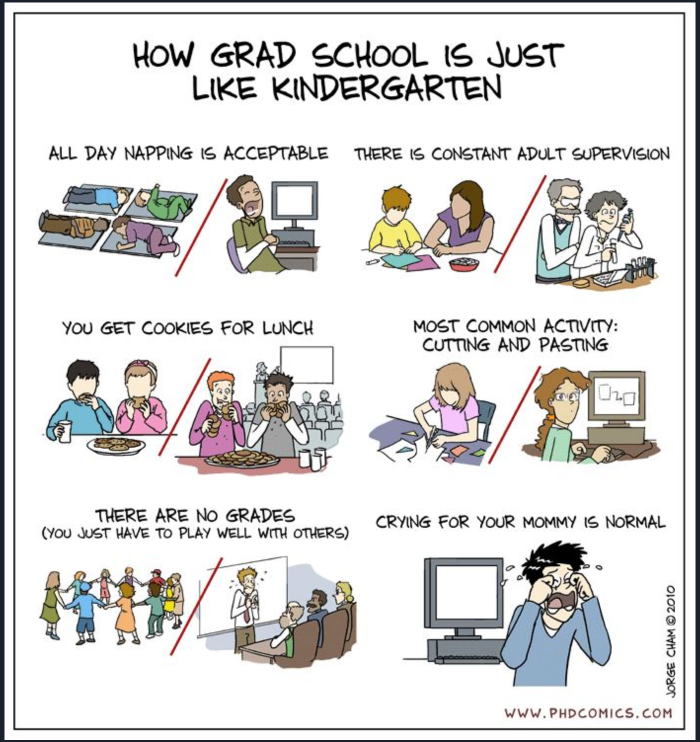
Prior to Oracle, he completed his Ph.D. at the DIAS lab of EPFL in Lausanne, focusing on scaling up highly concurrent analytical database workloads on multi-socket multi-core servers through (a) sharing data and work across concurrent queries, and (b) adaptive NUMA-aware data placement and task scheduling. During his Ph.D., he cooperated with the SAP HANA database team.

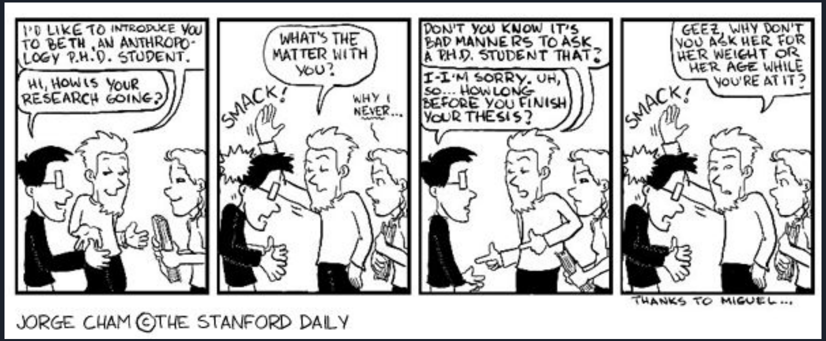
Academia (& industry collab) 6y

Before his Ph.D., he completed his studies in Electrical & Computer Engineering at NTUA in Athens.

School 5y

Grad life


HOW GRAD SCHOOL IS JUST LIKE KINDERGARTEN
ALL DAY NAPPING IS ACCEPTABLE
THERE IS CONSTANT ADULT SUPERVISION
YOU GET COOKIES FOR LUNCH
MOST COMMON ACTIVITY: CUTTING AND PASTING
THERE ARE NO GRADES
(YOU JUST HAVE TO PLAY WELL WITH OTHERS)
CRYING FOR YOUR MOMMY IS NORMAL
JORGE CHAM © 2010
WWW.PHDCOMICS.COM


I'D LIKE TO INTRODUCE YOU TO BETH, AN ANTHROPOLOGY P.H.D. STUDENT.
HI, HOW IS YOUR RESEARCH GOING?
SMACK!
WHAT'S THE MATTER WITH YOU?
WHY I NEVER...
DON'T YOU KNOW IT'S BAD MANNERS TO ASK A P.H.D. STUDENT THAT?
I-I'M SORRY. UH, SO... HOW LONG BEFORE YOU FINISH YOUR THESIS?
SMACK!
GEEZ, WHY DON'T YOU ASK HER FOR HER WEIGHT OR HER AGE WHILE YOU'RE AT IT?
THANKS TO MIGUEL...
JORGE CHAM ©THE STANFORD DAILY


I WANT YOU TO KNOW, I EXPECT ALL MY STUDENTS TO BE IN THE LAB DURING REGULAR WORKING HOURS.
YOU MEAN 9 TO 5?
I MEAN 6 am TO MIDNIGHT.
BUT MY STIPEND ONLY COVERS 20 HOURS A WEEK.
OF QUALITY WORK, NOT GRAD STUDENT "WORK".
JORGE CHAM © 2006
WWW.PHDCOMICS.COM

Myths & truths

Academia

- Goal: innovation, research
- Freedom
- Long-term research / impact
- Flexibility in organizing self
- Difficult work-life balance
- Progress through failures
- Struggle for money (grants, collabs)
- Uncertain job security
- Peer-review scrutiny
- Lonely
- Output: publish (or perish), presentations

Industry

- Goal: monetization
- Company needs
- Short-term/changing plan, direct impact
- More fine-grained, stricter plan
- 9-5 schedule
- Should rarely fail
- Direct money flow (e.g., sales)
- Stable pay flow
- Scrutiny by direct colleagues/managers
- Team effort
- Output: product, reputation, patents

(albeit, R&D is some middle ground)

Ref: "Academia vs industry", Setia Pramana, 2013, https://www.slideshare.net/slideshow/academia-vs-industry/16334490

PhD timeline at EPFL

2011	2012	2013	2014	2015	2016	
Entry	**Candidacy exam**	**Collab, 1st paper**	**More publishing**	**Final steps**	**Graduation**	Oracle Labs In Zurich
- Bureaucracy for Lausanne, Switzerland - Search lab & advisor - Courses to pass with high grades - Fun with colleagues - Focus on setting up PhD topic and work - Idea for a paper, with a PhD and postdoc buddy	- Candidacy project - Run experiments - Reading groups - Teaching assistant - Openhouse posters - 1st submission about sharing to VLDB, rejected - EcoCloud retreat - Candidacy exam on 3 papers, by profs - Internship at SAP in Germany	- Continue PhD collab with SAP. - Meetings with 2 SAP buddies. NUMA. - EPFL <-> SAP - Review papers - Dagstuhl retreat - 1st paper VLDB @ Riva del Garda	- Tutorial & poster SIGMOD @ Utah - Conference paper - Workshop paper - Tutoring interns - CUSO winter school - NUMA impressive results and trust - Thesis proposal	- 2nd paper VLDB @ Hawaii - Paper with intern - 2nd place SIGMOD programming contest - Work on final paper	- 3rd paper VLDB 2017 @ Munich - Tutorial book - Prepping next in line for SAP collab - Thesis writing - Industry interviews (and preparation) - Thesis presentation - Graduation party	

Software development I did in research

Academia

- Open-source, C++
- Low-level (CPU instructions, caches, performance)
- Prototyping – things can be buggy and not full features (e.g., half-baked parser) – happy path
- Supervision by PhDs, postdoc, professor. Collab with other academic colleagues.
- Ultimate output: experimental results showcasing the potential

Industry collaboration

- Still prototyping, but baking inside the production code
- PRs reviewed by colleagues
- Real-world measurable impact on product
- Big hardware and clusters to use
- Advice from other industry experts
- Final result may shape the product

Paper anatomy: Intro, related work

Title

Authors

Abstract

Intro

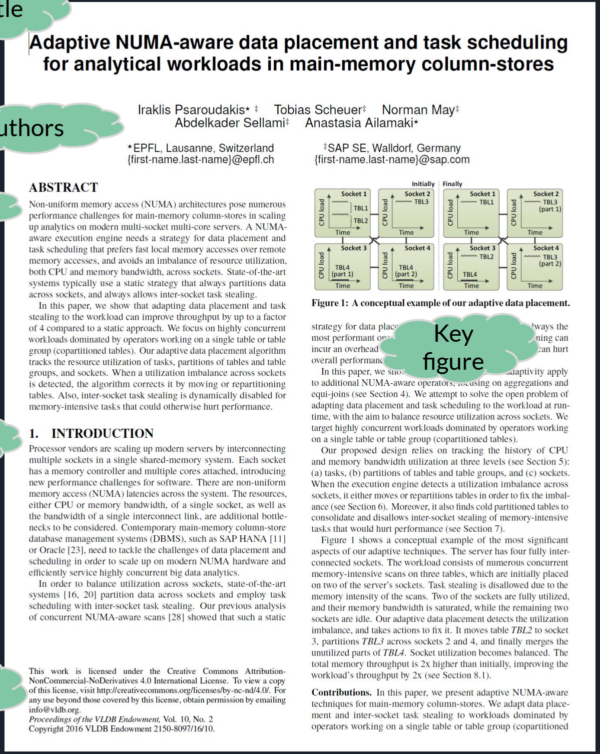
Copyright

Key figure

>= 3 contributions

LaTeX template ~ feels like coding

Related work



Adaptive NUMA-aware data placement and task scheduling for analytical workloads in main-memory column-stores

Iraklis Psaroudakis*‡ Tobias Scheuer‡ Norman May‡
Abdelkader Sellami‡ Anastasia Ailamaki*

*EPFL, Lausanne, Switzerland
{first-name.last-name}@epfl.ch

‡SAP SE, Walldorf, Germany
{first-name.last-name}@sap.com

ABSTRACT

Non-uniform memory access (NUMA) architectures pose numerous performance challenges for main-memory column-stores in scaling up analytics on modern multi-socket multi-core servers. A NUMA-aware execution engine needs a strategy for data placement and task scheduling that prefers fast local memory accesses over remote memory accesses, and avoids an imbalance of resource utilization, both CPU and memory bandwidth, across sockets. State-of-the-art systems typically use a static strategy that always partitions data across sockets, and always allows inter-socket task stealing.

In this paper, we show that adapting data placement and task stealing to the workload can improve throughput by up to a factor of 4 compared to a static approach. We focus on highly concurrent workloads dominated by operators working on a single table or table group (copartitioned tables). Our adaptive data placement algorithm tracks the resource utilization of tasks, partitions of tables and table groups, and sockets. When a utilization imbalance across sockets is detected, the algorithm corrects it by moving or repartitioning tables. Also, inter-socket task stealing is dynamically disabled for memory-intensive tasks that could otherwise hurt performance.

1. INTRODUCTION

Processor vendors are scaling up modern servers by interconnecting multiple sockets in a single shared-memory system. Each socket has a memory controller and multiple cores attached, introducing new performance challenges for software. There are non-uniform memory access (NUMA) latencies across the system. The resources, either CPU or memory bandwidth, of a single socket, as well as the bandwidth of a single interconnect link, are additional bottlenecks to be considered. Contemporary main-memory column-store database management systems (DBMS), such as SAP HANA [11] or Oracle [23], need to tackle the challenges of data placement and scheduling in order to scale up on modern NUMA hardware and efficiently service highly concurrent big data analytics.

In order to balance utilization across sockets, state-of-the-art systems [16, 20] partition data across sockets and employ task scheduling with inter-socket task stealing. Our previous analysis of concurrent NUMA-aware scans [28] showed that such a static

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-nd/4.0/. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org.
Proceedings of the VLDB Endowment, Vol. 10, No. 2
Copyright 2016 VLDB Endowment 2150-8097/16/10.

Figure 1: A conceptual example of our adaptive data placement.

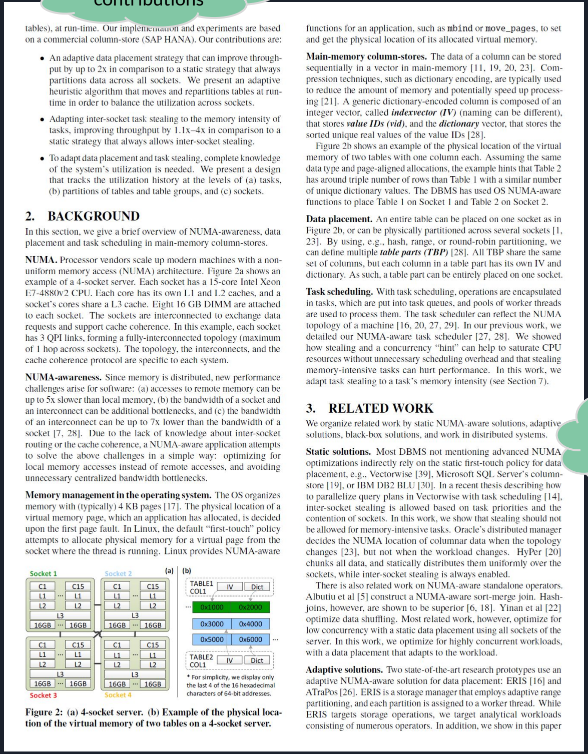
strategy for data plac[illegible] ways the most performant op[illegible] ning can incur an overhead [illegible] can hurt overall performanc[illegible]

In this paper, we sho[illegible] adaptivity apply to additional NUMA-aware operators, focusing on aggregations and equi-joins (see Section 4). We attempt to solve the open problem of adapting data placement and task scheduling to the workload at run-time, with the aim to balance resource utilization across sockets. We target highly concurrent workloads dominated by operators working on a single table or table group (copartitioned tables).

Our proposed design relies on tracking the history of CPU and memory bandwidth utilization at three levels (see Section 5): (a) tasks, (b) partitions of tables and table groups, and (c) sockets. When the execution engine detects a utilization imbalance across sockets, it either moves or repartitions tables in order to fix the imbalance (see Section 6). Moreover, it also finds cold partitioned tables to consolidate and disallows inter-socket stealing of memory-intensive tasks that would hurt performance (see Section 7).

Figure 1 shows a conceptual example of the most significant aspects of our adaptive techniques. The server has four fully inter-connected sockets. The workload consists of numerous concurrent memory-intensive scans on three tables, which are initially placed on two of the server's sockets. Task stealing is disallowed due to the memory intensity of the scans. Two of the sockets are fully utilized, and their memory bandwidth is saturated, while the remaining two sockets are idle. Our adaptive data placement detects the utilization imbalance, and takes actions to fix it. It moves table *TBL2* to socket 3, partitions *TBL3* across sockets 2 and 4, and finally merges the unutilized parts of *TBL4*. Socket utilization becomes balanced. The total memory throughput is 2x higher than initially, improving the workload's throughput by 2x (see Section 8.1).

Contributions. In this paper, we present adaptive NUMA-aware techniques for main-memory column-stores. We adapt data placement and inter-socket task stealing to workloads dominated by operators working on a single table or table group (copartitioned



tables), at run-time. Our implementation and experiments are based on a commercial column-store (SAP HANA). Our contributions are:

- An adaptive data placement strategy that can improve throughput by up to 2x in comparison to a static strategy that always partitions data across all sockets. We present an adaptive heuristic algorithm that moves and repartitions tables at run-time in order to balance the utilization across sockets.
- Adapting inter-socket task stealing to the memory intensity of tasks, improving throughput by 1.1x–4x in comparison to a static strategy that always allows inter-socket stealing.
- To adapt data placement and task stealing, complete knowledge of the system's utilization is needed. We present a design that tracks the utilization history at the levels of (a) tasks, (b) partitions of table and table groups, and (c) sockets.

2. BACKGROUND

In this section, we give a brief overview of NUMA-awareness, data placement and task scheduling in main-memory column-stores.

NUMA. Processor vendors scale up modern machines with a non-uniform memory access (NUMA) architecture. Figure 2a shows an example of a 4-socket server. Each socket has a 15-core Intel Xeon E7-4880v2 CPU. Each core has its own L1 and L2 caches, and a socket's cores share a L3 cache. Eight 16 GB DIMM are attached to each socket. The sockets are interconnected to exchange data requests and support cache coherence. In this example, each socket has 3 QPI links, forming a fully-interconnected topology (maximum of 1 hop across sockets). The topology, the interconnects, and the cache coherence protocol are specific to each system.

NUMA-awareness. Since memory is distributed, new performance challenges arise for software: (a) accesses to remote memory can be up to 5x slower than local memory, (b) the bandwidth of a socket and an interconnect can be additional bottlenecks, and (c) the bandwidth of an interconnect can be up to 7x lower than the bandwidth of a socket [7, 28]. Due to the lack of knowledge about inter-socket routing or the cache coherence, a NUMA-aware application attempts to solve the above challenges in a simple way: optimizing for local memory accesses instead of remote accesses, and avoiding unnecessary centralized bandwidth bottlenecks.

Memory management in the operating system. The OS organizes memory with (typically) 4 KB pages [17]. The physical location of a virtual memory page, which an application has allocated, is decided upon the first page fault. In Linux, the default "first-touch" policy attempts to allocate physical memory for a virtual page from the socket where the thread is running. Linux provides NUMA-aware

Figure 2: (a) 4-socket server. (b) Example of the physical location of the virtual memory of two tables on a 4-socket server.

functions for an application, such as mbind or move_pages, to set and get the physical location of its allocated virtual memory.

Main-memory column-stores. The data of a column can be stored sequentially in a vector in main-memory [11, 19, 20, 23]. Compression techniques, such as dictionary encoding, are typically used to reduce the amount of memory and potentially speed up processing [21]. A generic dictionary-encoded column is composed of an integer vector, called ***indexvector (IV)*** (naming can be different), that stores ***value IDs (vid)***, and the ***dictionary*** vector, that stores the sorted unique real values of the value IDs [28].

Figure 2b shows an example of the physical location of the virtual memory of two tables with one column each. Assuming the same data type and page-aligned allocations, the example hints that Table 2 has around triple number of rows than Table 1 with a similar number of unique dictionary values. The DBMS has used OS NUMA-aware functions to place Table 1 on Socket 1 and Table 2 on Socket 2.

Data placement. An entire table can be placed on one socket as in Figure 2b, or can be physically partitioned across several sockets [1, 23]. By using, e.g., hash, range, or round-robin partitioning, we can define multiple ***table parts (TBP)*** [28]. All TBP share the same set of columns, but each column in a table part has its own IV and dictionary. As such, a table part can be entirely placed on one socket.

Task scheduling. With task scheduling, operations are encapsulated in tasks, which are put into task queues, and pools of worker threads are used to process them. The task scheduler can reflect the NUMA topology of a machine [16, 20, 27, 29]. In our previous work, we detailed our NUMA-aware task scheduler [27, 28]. We showed how stealing and a concurrency "hint" can help to saturate CPU resources without unnecessary scheduling overhead and that stealing memory-intensive tasks can hurt performance. In this work, we adapt task stealing to a task's memory intensity (see Section 7).

3. RELATED WORK

We organize related work by static NUMA-aware solutions, adaptive solutions, black-box solutions, and work in distributed systems.

Static solutions. Most DBMS not mentioning advanced NUMA optimizations indirectly rely on the static first-touch policy for data placement, e.g., Vectorwise [39], Microsoft SQL Server's column-store [19], or IBM DB2 BLU [30]. In a recent thesis describing how to parallelize query plans in Vectorwise with task scheduling [14], inter-socket stealing is allowed based on task priorities and the contention of sockets. In this work, we show that stealing should not be allowed for memory-intensive tasks. Oracle's distributed manager decides the NUMA location of columnar data when the topology changes [23], but not when the workload changes. HyPer [20] chunks all data, and statically distributes them uniformly over the sockets, while inter-socket stealing is always enabled.

There is also related work on NUMA-aware standalone operators. Albutiu et al [5] construct a NUMA-aware sort-merge join. Hash-joins, however, are shown to be superior [6, 18]. Yinan et al [22] optimize data shuffling. Most related work, however, optimize for low concurrency with a static data placement using all sockets of the server. In this work, we optimize for highly concurrent workloads, with a data placement that adapts to the workload.

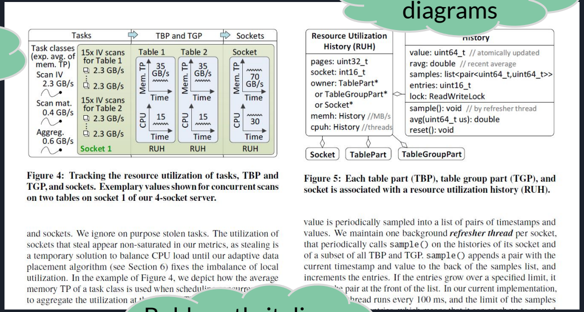
Adaptive solutions. Two state-of-the-art research prototypes use an adaptive NUMA-aware solution for data placement: ERIS [16] and ATraPos [26]. ERIS is a storage manager that employs adaptive range partitioning, and each partition is assigned to a worker thread. While ERIS targets storage operations, we target analytical workloads consisting of numerous operators. In addition, we show in this paper

Visual cues

Figures help readability

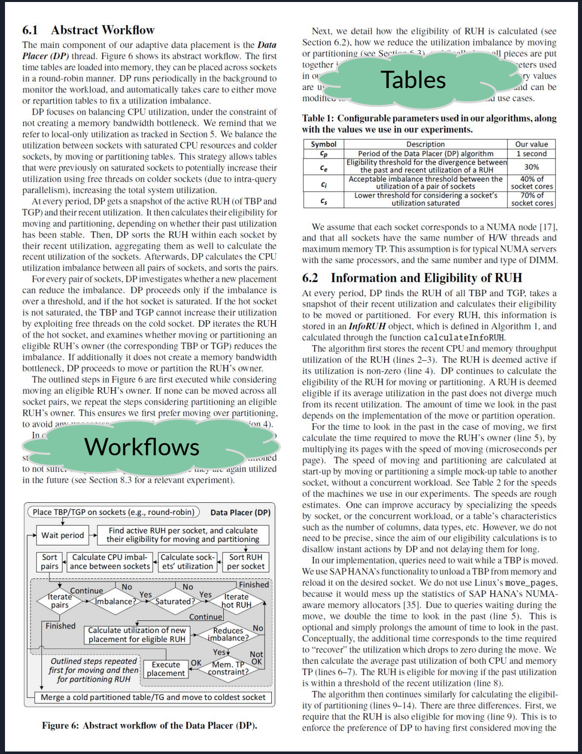
UML diagrams

Bold, math, italics, acronyms, paragraphs



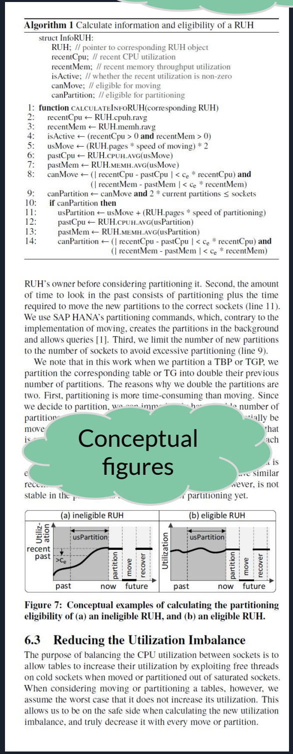
Tables

Workflows



Algorithms

Conceptual figures



Experimental evaluation

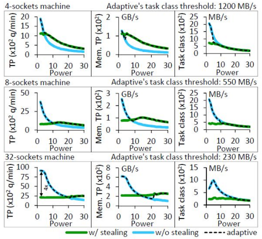


Figure 9: Calibration experiment for three NUMA servers.

saturates the interconnect network and overwhelms the already saturated remote memory controllers. The overwhelmed memory controllers achieve much lower overall memory TP (for both local and remote tasks) than the case of disallowing stealing.

As n increases, the workload becomes more CPU-intensive, and the memory TP of the system and the task class finally starts decreasing. In terms of CPI (cycles per instruction), e.g., on the 4-socket server for the case when stealing is disabled, it starts at 0.83 and gradually increases up to 1.25 (for $n = 30$). At a switching point, stealing becomes better, and remote tasks can be satisfied through the interconnects and the remote memory controllers sufficiently.

At the switching point, we mark the memory TP of the task class as the threshold for stealing vs. not stealing. Our adaptive task stealing uses this threshold at run-time. If the exponential average of a task class becomes higher than the threshold, stealing is disallowed for tasks of this task class. If the exponential average becomes lower than the threshold, stealing is allowed.

Figure 9 shows the threshold pinpointed for each server, and also shows the adaptive task stealing that uses this threshold. The adaptive line achieves the best throughput for all cases of power n, successfully allowing stealing at the switching point.

Finally, we note that the calibration experiment can be further extended to specialize the threshold for more cases of: different number of sockets having data, different CPU utilization per socket, etc. Our current calibration experiment is sufficient for our use cases and experiments, as it roughly finds out the switching point for the "middle" case where half of the server's sockets have active data.

8. EXPERIMENTAL EVALUATION

We first present our experimental configuration. Then, we present results of a custom benchmark and finally of the TPC-H benchmark.

Experimental configuration. We use a prototype built on SAP HANA (SPS11), a commercial main-memory column-store, with our NUMA-aware task scheduler and scans [28]. We add support for more NUMA-aware operators (see Section 4), track resource utilization (see Section 5), and employ our adaptive NUMA-aware data placement (see Section 6) and task stealing (see Section 7).

Configuration

For all experiments, we warm up the DBMS, we admit all clients and disable result caching. LLC misses, CPU load, and memory TP are gathered from Linux and H/W counters (integrating Intel PCM [9]). The ... ly utilizations we track (v... ce metric corresp... sockets, when cal... averaged sampled ... eral is decreased with a stable work... with a timeline consist of a single run, while any data points (in previous sections as well) are averages of at least three iterations with a standard deviation <10%.

Repetitions, STD

Table 2 shows characteristics of the servers we use. The first is the one of Figure 2a. The second is an 8-socket server. The third is a rack-scale 32-socket SGI UV 300H server. NUMA characteristics, such as local and inter-socket idle latencies and peak memory bandwidths, are measured with Intel Memory Latency Checker [34].

Table 2: Characteristics of the three NUMA servers we use.

Statistic \ Machine	4x15-core Intel Xeon E7-4880v2 at 2.50GHz	8x15-core Intel Xeon E7-8880v2 at 2.50GHz	32x18-core Intel Xeon E7-8890v3 at 2.50GHz
Memory per socket	128 GB	128 GB	512 GB
Local latency	108 ns	110 ns	120 ns
1 hop latency	170 ns	320 ns	320 ns
Max hops latency	170 ns	390 ns	590 ns
Local B/W	70 GB/s	70 GB/s	45.5 GB/s
1 hop B/W	12.5 GB/s	10.5 GB/s	15 GB/s
Max hops B/W	12.5 GB/s	9.5 GB/s	7.3 GB/s
Total local B/W	280 GB/s	570 GB/s	1363 GB/s
Stealing threshold	1200 MB/s	550 MB/s	230 MB/s
Move us/page	59	63	60
Partition us/page	109	123	129

Custom benchmark. Our dataset has 64 tables (TBL1-64). For each table we generate a CSV file of 50 million rows, around 3.2 GB, for a total of 204 GB files. Each table has an ID integer column (PK), 8 additional columns (COL1-8) of random integers (uniform distribution), and a partitioning specification (hash) on ID. The 8 columns have bitcases 17 to 24, so as to have different number of unique values. Each experiment m... ment.

The workload is generated ... t server. Clients continuous... throughput (TP). At each ex... are used, which query type... arget, and which selectivity they use. Th... ypes are:

Setup

(a) `SELECT COLx FROM TBLy WHERE COLx >= ? AND COLx <= ?`. The client selects a random column from its target table. The query involves both scan phases mentioned in Section 4.

(b) `SELECT COL1, SUM(TO_DOUBLE(COLx)) FROM TBLy WHERE COLx >= ? AND COLx <= ? GROUP BY COL1`. The client selects a random column (COL2-8) from its target table to aggregate and group-by COL1. This query involves the aggregation phases mentioned in Section 4. We choose COL1 for the group-by because it has the least number of unique values. We cast to double to avoid potential numeric overflow errors.

(c) `SELECT TBLz.COLx FROM TBLy, TBLz WHERE TBLy.ID = TBLz.ID AND TBLy.COLx >= ? AND TBLy.COLx <= ?`. The client joins two target tables on the ID column. A random column is selected to filter and project. This query involves the equi-join steps mentioned in Section 4.

Before each experiment begins, we let clients build a prepared statement for each query they can issue. There are no thinking times. The clients do not fetch results, in order to not let the network transfer dominate. Each TP value in a timeline corresponds to the slope of the achieved queries during the previous 30 seconds.

8.1 Adaptive Data Placement

The first experiment realizes the introductory example of Figure 1. TBL1 and TBL2 are placed on socket S1, TBL3 on S2, and TBL4 is partitioned across S3 and S4. Each of the tables TBL1-3 are targeted by 64 clients executing query (a) with a low selectivity (0.001%) for 5 minutes. Figure 10 shows the timelines of the throughput (TP), the utilization imbalance, and additional performance measurements that include H/W counters as well as our tracked utilization (RUH).

At the beginning, only S1 and S2 execute queries as shown by their RUH. Queries are dominated by the scan's first phase ("IV-Scan"). Tasks are memory-intensive as shown by the task class's memory TP, which is over the stealing threshold. That is why adaptive stealing disallows stealing, and most LLC misses are local. As shown by the tables' RUH, TBL1 and TBL2 share S1, while TBL3 fully utilizes S2.

DP recognizes the imbalance, but does not take action because the TBP are not yet eligible to be moved or partitioned. DP searches the catalog to find TBL4 which is partitioned and cold (thus not shown in Figure 10), and at 16 s starts a background request to merge it. The merge finishes at 64 s, and the single TBP is moved to S4 (a cold socket) at 106 s. The merge and move contribute to the small bump in the CPU load and memory TP of S3 and S4. Another reason for their increased CPU load is that their worker threads attempt to steal tasks from other sockets, but tasks are memory-intensive and cannot be stolen in this experiment. Since S3 and S4 do not process any queries, we do not account this busy CPU load in their RUH.

At 53 s (see markers on the timeline of the tables' RUH graphs), DP examines the pair of S1 and S3. It decides to fix their imbalance by moving TBL2, which has become eligible for moving, to S3. The move completes at 91 s. Overall TP and memory TP are increased.

Next, DP detects that there is still a utilization imbalance because 3 sockets are utilized and S4 is not (as shown by its RUH). At 108 s, DP examines the pair of S2 and S4. It decides to fix their imbalance by partitioning TBL3, which is eligible for partitioning, into two parts. At 174 s, partitioning completes, and the two TBP are moved to S2 and S4 concurrently, which completes at 190 s.

After that point, the imbalance is decreased within our threshold, and there are no more actions. In comparison to the beginning of the experiment, overall memory TP is 2x more, and TP is also 2x more.

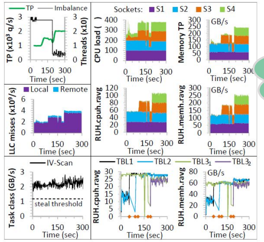


Figure 10: Introductory experiment showing the adaptive data placem... er).

Detailed results

8.2 Adaptive Task Stealing

To show the effect of adaptive task stealing, we use scans of varying selectivity. We place TBL1 on S2 and TBL2 on S4. Adaptive placement is disabled. Each of the tables is targeted by 256 clients executing query (a) with the specified selectivity. Half of the sockets have local tasks, while the other half would need to steal. For each selectivity, we execute 5 min runs of: enabled stealing for all tasks, disabled stealing for all tasks, and adaptive stealing. We report each run's average TP. The results are shown in Figure 11.

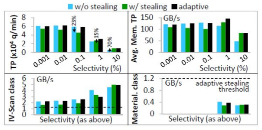


Figure 11: Experiment showing how adaptive task stealing disallows stealing of memory-intensive classes (4-socket server).

For low selectivities, the scan's first phase ("IV-scan") dominates. Tasks are memory-intensive and stealing hurts TP by up to 23% for the case of 0.1% selectivity. As selectivity increases, the scan's second phase (materialization) dominates and is parallelized. The fewer IV-scan tasks can utilize more memory bandwidth on their socket. The dominating materialization tasks are CPU-intensive, due to their random accesses to the dictionary, and thus stealing helps improve TP by up to 70% for the case of 10% selectivity. Adaptive stealing achieves the best TP of either stealing or not stealing in all cases of selectivity. It can also, e.g., for the case of 1% selectivity, further improve performance by 15%. This is due to disallowing stealing of IV-scan tasks, and allowing stealing of materialization tasks, instead of taking a static strategy for all task classes.

Takeaways

8.3 Partitioning Overhead

Here, we show how our adaptive data placement can avoid the overhead of unnecessary partitioning. We focus on aggregations, but the impl... ing are similar for joins ... across all sockets ... onsecutive 5min ... executing query ... has no activity. ... shows the results. ... erged. During the third p... appened to be merged on the same socket to balance ... TP reaches 1.7x of the TP of the first phase, because there is no partitioning overhead, and the server can be saturated with non-partitioned tables. This is also shown by the improved memory TP and local LLC misses.

No red+green. Print in gray

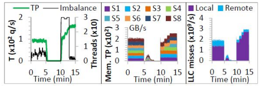
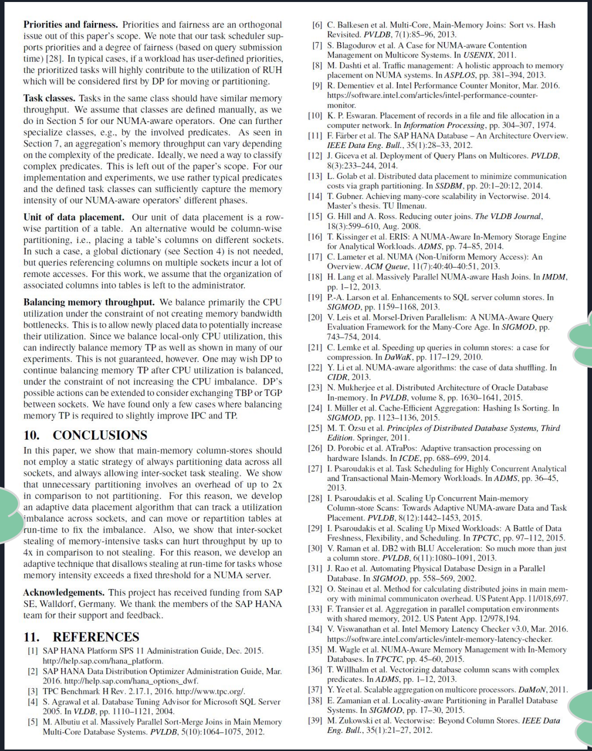


Figure 12: The overhead of partitioning (8-socket server).

Other types of papers exist (e.g., vision)

Conclusion and references



Priorities and fairness. Priorities and fairness are an orthogonal issue out of this paper's scope. We note that our task scheduler supports priorities and a degree of fairness (based on query submission time) [28]. In typical cases, if a workload has user-defined priorities, the prioritized tasks will highly contribute to the utilization of RUH which will be considered first by DP for moving or partitioning.

Task classes. Tasks in the same class should have similar memory throughput. We assume that classes are defined manually, as we do in Section 5 for our NUMA-aware operators. One can further specialize classes, e.g., by the involved predicates. As seen in Section 7, an aggregation's memory throughput can vary depending on the complexity of the predicate. Ideally, we need a way to classify complex predicates. This is left out of the paper's scope. For our implementation and experiments, we use rather typical predicates and the defined task classes can sufficiently capture the memory intensity of our NUMA-aware operators' different phases.

Unit of data placement. Our unit of data placement is a row-wise partition of a table. An alternative would be column-wise partitioning, i.e., placing a table's columns on different sockets. In such a case, a global dictionary (see Section 4) is not needed, but queries referencing columns on multiple sockets incur a lot of remote accesses. For this work, we assume that the organization of associated columns into tables is left to the administrator.

Balancing memory throughput. We balance primarily the CPU utilization under the constraint of not creating memory bandwidth bottlenecks. This is to allow newly placed data to potentially increase their utilization. Since we balance local-only CPU utilization, this can indirectly balance memory TP as well as shown in many of our experiments. This is not guaranteed, however. One may wish DP to continue balancing memory TP after CPU utilization is balanced, under the constraint of not increasing the CPU imbalance. DP's possible actions can be extended to consider exchanging TBP or TGP between sockets. We have found only a few cases where balancing memory TP is required to slightly improve IPC and TP.

10. CONCLUSIONS

In this paper, we show that main-memory column-stores should not employ a static strategy of always partitioning data across all sockets, and always allowing inter-socket task stealing. We show that unnecessary partitioning involves an overhead of up to 2x in comparison to not partitioning. For this reason, we develop an adaptive data placement algorithm that can track a utilization imbalance across sockets, and can move or repartition tables at run-time to fix the imbalance. Also, we show that inter-socket stealing of memory-intensive tasks can hurt throughput by up to 4x in comparison to not stealing. For this reason, we develop an adaptive technique that disallows stealing at run-time for tasks whose memory intensity exceeds a fixed threshold for a NUMA server.

Acknowledgements. This project has received funding from SAP SE, Walldorf, Germany. We thank the members of the SAP HANA team for their support and feedback.

11. REFERENCES

[1] SAP HANA Platform SPS 11 Administration Guide, Dec. 2015. http://help.sap.com/hana_platform.

[2] SAP HANA Data Distribution Optimizer Administration Guide, Mar. 2016. http://help.sap.com/hana_options_dwf.

[3] TPC Benchmark H Rev. 2.17.1, 2016. http://www.tpc.org/.

[4] S. Agrawal et al. Database Tuning Advisor for Microsoft SQL Server 2005. In *VLDB*, pp. 1110–1121, 2004.

[5] M. Albutiu et al. Massively Parallel Sort-Merge Joins in Main Memory Multi-Core Database Systems. *PVLDB*, 5(10):1064–1075, 2012.

[6] C. Balkesen et al. Multi-Core, Main-Memory Joins: Sort vs. Hash Revisited. *PVLDB*, 7(1):85–96, 2013.

[7] S. Blagodurov et al. A Case for NUMA-aware Contention Management on Multicore Systems. In *USENIX*, 2011.

[8] M. Dashti et al. Traffic management: A holistic approach to memory placement on NUMA systems. In *ASPLOS*, pp. 381–394, 2013.

[9] R. Dementiev et al. Intel Performance Counter Monitor, Mar. 2016. https://software.intel.com/articles/intel-performance-counter-monitor.

[10] K. P. Eswaran. Placement of records in a file and file allocation in a computer network. In *Information Processing*, pp. 304–307, 1974.

[11] F. Färber et al. The SAP HANA Database – An Architecture Overview. *IEEE Data Eng. Bull.*, 35(1):28–33, 2012.

[12] J. Giceva et al. Deployment of Query Plans on Multicores. *PVLDB*, 8(3):233–244, 2014.

[13] L. Golab et al. Distributed data placement to minimize communication costs via graph partitioning. In *SSDBM*, pp. 20:1–20:12, 2014.

[14] T. Gubner. Achieving many-core scalability in Vectorwise. 2014. Master's thesis. TU Ilmenau.

[15] G. Hill and A. Ross. Reducing outer joins. *The VLDB Journal*, 18(3):599–610, Aug. 2008.

[16] T. Kissinger et al. ERIS: A NUMA-Aware In-Memory Storage Engine for Analytical Workloads. *ADMS*, pp. 74–85, 2014.

[17] C. Lameter et al. NUMA (Non-Uniform Memory Access): An Overview. *ACM Queue*, 11(7):40:40–40:51, 2013.

[18] H. Lang et al. Massively Parallel NUMA-aware Hash Joins. In *IMDM*, pp. 1–12, 2013.

[19] P.-A. Larson et al. Enhancements to SQL server column stores. In *SIGMOD*, pp. 1159–1168, 2013.

[20] V. Leis et al. Morsel-Driven Parallelism: A NUMA-Aware Query Evaluation Framework for the Many-Core Age. In *SIGMOD*, pp. 743–754, 2014.

[21] C. Lemke et al. Speeding up queries in column stores: a case for compression. In *DaWaK*, pp. 117–129, 2010.

[22] Y. Li et al. NUMA-aware algorithms: the case of data shuffling. In *CIDR*, 2013.

[23] N. Mukherjee et al. Distributed Architecture of Oracle Database In-memory. In *PVLDB*, volume 8, pp. 1630–1641, 2015.

[24] I. Müller et al. Cache-Efficient Aggregation: Hashing Is Sorting. In *SIGMOD*, pp. 1123–1136, 2015.

[25] M. T. Özsu et al. *Principles of Distributed Database Systems, Third Edition*. Springer, 2011.

[26] D. Porobic et al. ATraPos: Adaptive transaction processing on hardware Islands. In *ICDE*, pp. 688–699, 2014.

[27] I. Psaroudakis et al. Task Scheduling for Highly Concurrent Analytical and Transactional Main-Memory Workloads. In *ADMS*, pp. 36–45, 2013.

[28] I. Psaroudakis et al. Scaling Up Concurrent Main-memory Column-store Scans: Towards Adaptive NUMA-aware Data and Task Placement. *PVLDB*, 8(12):1442–1453, 2015.

[29] I. Psaroudakis et al. Scaling Up Mixed Workloads: A Battle of Data Freshness, Flexibility, and Scheduling. In *TPCTC*, pp. 97–112, 2015.

[30] V. Raman et al. DB2 with BLU Acceleration: So much more than just a column store. *PVLDB*, 6(11):1080–1091, 2013.

[31] J. Rao et al. Automating Physical Database Design in a Parallel Database. In *SIGMOD*, pp. 558–569, 2002.

[32] O. Steinau et al. Method for calculating distributed joins in main memory with minimal communicaton overhead. US Patent App. 11/018,697.

[33] F. Transier et al. Aggregation in parallel computation environments with shared memory, 2012. US Patent App. 12/978,194.

[34] V. Viswanathan et al. Intel Memory Latency Checker v3.0, Mar. 2016. https://software.intel.com/articles/intelr-memory-latency-checker.

[35] M. Wagle et al. NUMA-Aware Memory Management with In-Memory Databases. In *TPCTC*, pp. 45–60, 2015.

[36] T. Willhalm et al. Vectorizing database column scans with complex predicates. In *ADMS*, pp. 1–12, 2013.

[37] Y. Ye et al. Scalable aggregation on multicore processors. *DaMoN*, 2011.

[38] E. Zamanian et al. Locality-aware Partitioning in Parallel Database Systems. In *SIGMOD*, pp. 17–30, 2015.

[39] M. Zukowski et al. Vectorwise: Beyond Column Stores. *IEEE Data Eng. Bull.*, 35(1):21–27, 2012.

Final summary and takeaway

References!

Strict page limit. We squeeze stuff!

Submission and reviews

Typical timeline:

- Prepare paper for months ahead of deadline
- Submit by deadline (e.g., Microsoft CMT)
- Submit conflicts of interest of authors
- Get (blind) reviews a couple of months later
- Possible rebuttal phase
- Acceptance/rejection notification
- If accepted, camera-ready deadline
- Copyright form
- At least 1 presented live in the conference
- Prepare presentation & do dry-runs
- Travel typically paid by academic institution
- Present & possible poster session

VLDB 2025: Important Dates

All deadlines below are 5 p.m. PT, unless otherwise noted.

	Chairs	Submissions Open	Submissions Deadline	Notifications	Camera-Ready Submissions
Research Track	Themis Palpanas University Paris Cite Nesime Tatbul Intel Labs and MIT	20th of every month starting March 2024	1st of every month until March 2025	15th of the next month following the deadline	Proceedings chairs will contact the authors with CR instructions by the 5th of the next month following the acceptance notification
Industrial Track	Surajit Chaudhuri Microsoft Research, USA Nikos Ntarmos Huawei, United Kingdom Jingren Zhou Alibaba Group, China	TBD	March 17, 2025	May 19, 2025	July 14, 2025
Tutorials	Hakan Ferhatosmanoglu, University of Warwick and AWS Madelon Hulsebos, CWI	TBD	April 15, 2025	May 30, 2025	Camera-ready: July 1, 2025 Slides availability: August 20, 2025
Demonstrations	Sourav S Bhowmick, NTU, Singapore Philippe Bonnet, University of Copenhagen, Denmark	TBD	March 30, 2025	May 27, 2025	TBD
Panels	Jana Giceva, Imperial College London Alexandra Meliou, University of Massachusetts, Amherst	TBD	May 15, 2025	May 30, 2025	June 31, 2025
Workshop Proposals	John Paparrizos, Ohio State University Norman Paton, University of Manchester	December 1, 2024	January 17, 2025	January 31, 2025	N/A
PhD Workshop Track	Sonia Bergamaschi, University of Modena and Reggio Emilia Raul Castro Fernandez, The University of Chicago	TBD	May 22, 2025	June 30, 2025	July 22, 2025

Presentation tips

1.5' / slide
20' incl. Q&A

Talk through visual cues

Keep interest

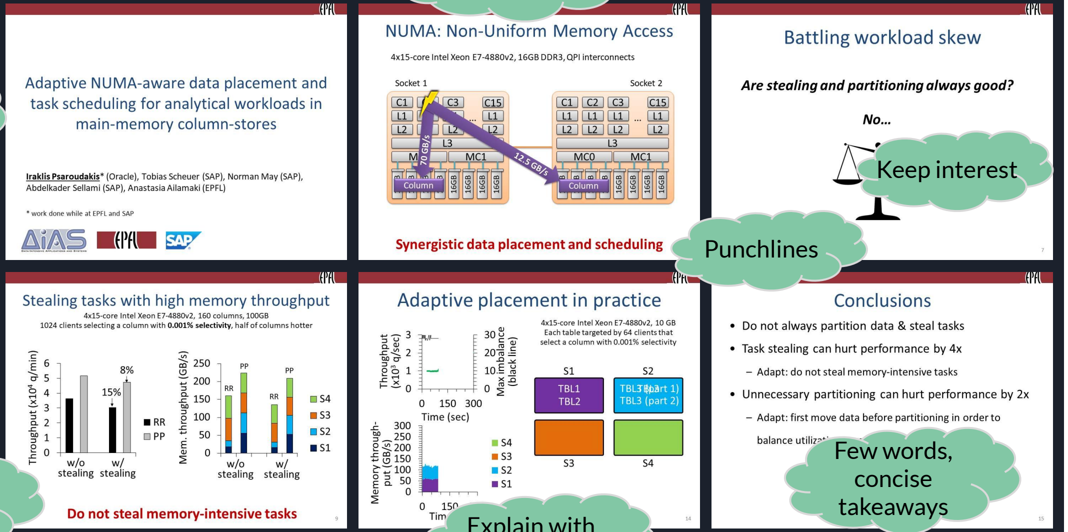
Punchlines

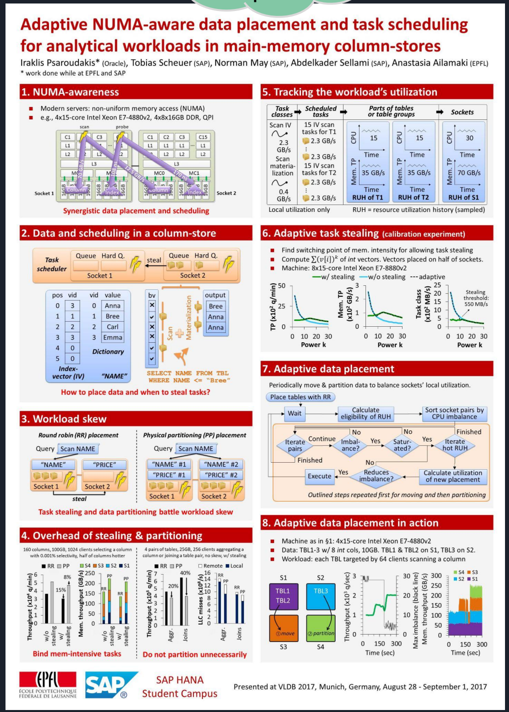
Core experiments

Explain with timeline

Few words, concise takeaways

You tell the story over the poster



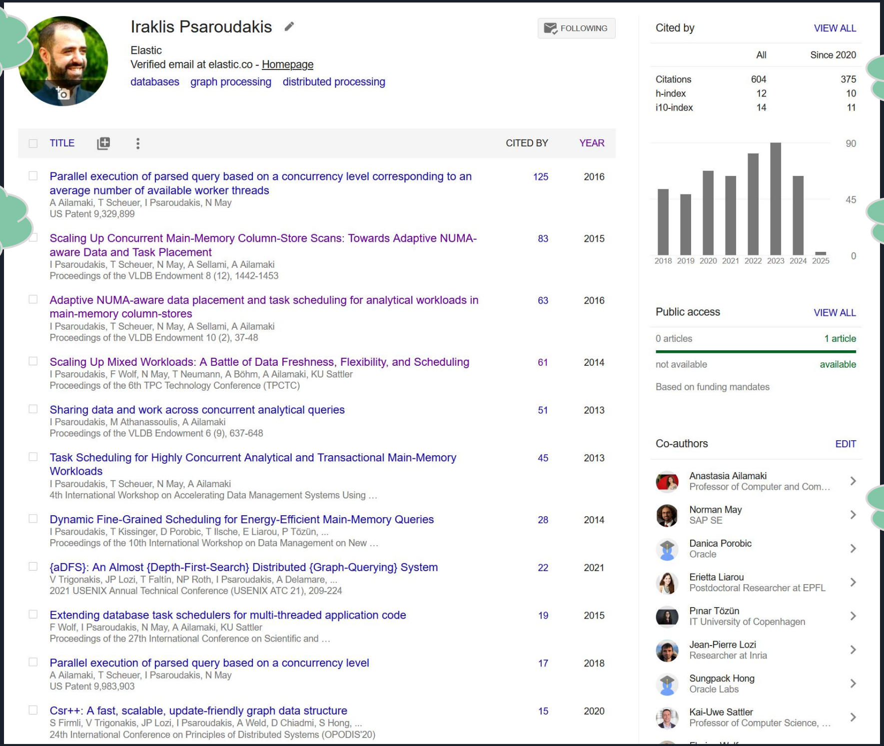


Scholar profile

https://scholar.google.com/citations?user=TUcmoosAAAAJ&hl=en

Important for research jobs

Most cited on top



Citation indices

Consistent citations

Popular co-authors

Conclusion

- Software developer life in academia & research involves
  - 35% coding
  - 35% experimenting, writing
  - 15% reading related work
  - 15% talking, presenting, getting feedback
- More flexibility and challenging topics than industry
- Less life-work balance than industry
- Publish or perish
- Experimental evaluation is key
- Presenting in a succinct manner is key
- PhD necessary for academia, wished for in R&D, not necessary for industry

Thank you!

Questions? www.kingherc.com