DAWN'19

Workshop on Database Aspects Explored by Wisconsin's New DB Researchers

December 09, 2019 from 8:00AM-9:40AM, and
December 11, 2019 from 8:00AM-9:40AM

in 1257 CS
Madison, WI

Picnic Point at Dawn. Photo by: Jeff Miller, UW-Madison University Communications

Monday December 9, 2019

Time (AM)	Title	Authors	Abstract
8:00-8:10	Ranked Enumeration of Conjunctive Query	Tien-Lung Fu, Pan Wu	For many data processing applications, enumerating query results according to an order by a ranking function is fundamental task. For example, users want to extract the top patterns from an edge-weighted graph, and the rank of each pattern is the sum of the weights of the edges. Ranked enumeration also exists in SQL queries with an ORDER BY clause. Usually, users want to see the first k results as quickly as possible without predetermined value of k. In our work, we investigate the enumeration of top-k answers for conjunctive queries against relational databases. Our main task is to design and implement data structure and algorithm that allow for efficient enumeration after a pre-processing phase. We designed and implemented a novel priority queue-based algorithm with near-optimal delay and non-trivial space guarantees that are output sensitive and depend on structure of the query. Our algorithms are divided into two phases: the pre-processing phase, where the system constructs a data structure that can be used later and the enumeration phase, when the results are generated. All our algorithms aim to minimize the time of the pre-processing phase, and guarantee a logarithmic delay during enumeration.
8:10-8:20	A Study on Query Optimization	Anjali, Sakshi Bansal	Query optimization is the part of the query process in which the database system compares different query strategies and chooses the one with the least expected cost. The query optimizer, which carries out this function, is a key part of the relational database and determines the most efficient way to access data. It makes it possible for the user to request the data without specifying how these data should be retrieved. The cost of accessing a query is a weighted combination of the I/O and processing costs. The I/O cost is the cost of accessing index and data pages from disk. Processing cost is estimated by assigning an instruction count to each step in computing the result of the query. In this project, we study the optimization done by PostgreSQL and SQLite by running the queries present in the SSB benchmark. The results obtained will help us to further investigate the query optimization techniques implemented by the two database systems.
8:20-8:30	Performance analysis of BitWeaving with modern hardware in Rust implementation	Suryadev Sahadevan Rajesh, Muthunagappan Muthuraman	Today, modern processors have improved bandwidth between the CPU and the caches. It also has lower access latency when compared to the previous generation processors. Since BitWeaving/V uses the caches present in the CPU effectively, we expect it to perform better than other algorithms used for scanning in the modern processors as well. In our implementation, we have used Rust. As Rust is memory safe and identifies data races in the compile-time, it avoids potential system crashes during runtime. We also show that the BitWeaving/V performs better in the Rust without any performance degradation.
8:30-8:40	Compression and Bitweaving	Aaron Whitaker	The goal of this project is to study the impact of additional compression on top of the Bitweaving storage formats, and to determine how to modify the Bitweaving protocol to perform predicate evaluations in this compressed space. Column data from the TPC-H benchmark standard are converted into the two storage formats used by the Bitweaving method, Bitweaving/V and Bitweaving/H. Three different compression techniques are then applied to to these codes, basic run length encoding, zlib, and bzip, and the compression ratios of these techniques are analyzed. Finally, methods for evaluating predicates on the compressed codes are analyzed.

8:40-8:50	Evaluating the performance of BitWeaving with AVX-512 SIMD instructions	Sanchit Jain	Column stores are widely used in analytics systems because their memory-layout is usually a lot more cache-efficient than that of row-stores, resulting in faster aggregations and comparisons. These column-stores often have dictionary-encoded values, which are smaller than the values they were decoded from. BitWeaving, a novel extension of bit-slicing, is an approach to speed-up analytics queries involving main-memory column-scalar scans. Supplementing it with SIMD instructions achieves a higher magnitude of speed-up. One of its variants, BitWeaving/V is poised to achieve speedup with SIMD instructions, than does BitWeaving/H. We implement BitWeaving in Apache Arrow, that runs on the Gandiva execution engine, and evaluate its performance sans the use of SIMD instructions. Modern hardware supports SIMD instructions of 512 bits, and we find that they entail in faster performance.
8:50-9:00	Accelerating Joins with Filters	Nicholas Corrado, Xiating Ouyang	In query optimization on star schemas, lookahead information passing (LIP) is a strategy exploiting the efficiency of probing succinct filters to eliminate practically all facts that do not appear in the final join results. Assuming data independency across all columns in the fact table, LIP achieves efficient and robust query optimization. We present some variants of LIP that can achieve empirically efficient query execution on fact table with correlated and even adversarial data columns, experimented on a skeleton database on top of Apache Arrow. We also analyze the performance of each variant of LIP using the notion of competitive ratio in online algorithms.
9:00-9:10	Multi-Query Optimization for Streaming Data	Elena Milkai	Continuous queries in streaming data allow users to obtain new results without having to issue the same query repeatedly. However, for a continuous query system to be useful must be capable of supporting efficiently a big amount of queries at each time. Moreover, the system should be able to handle new continuous queries that are dynamically added and removed. So, I propose an incremental group optimization approach in which queries are grouped according to their signatures. When a new query arrives, the existing groups are considered as possible optimization choices instead of re-grouping all the queries in the system. The new query is merged into existing groups whose signatures match that of the query. The implementation of the dynamic multiple query optimizer in streaming data is done in Apache Calcite.
9:10-:9:20	Scheduler on Hustle for Concurrent Query Processing	Lichengxi Huang	The project aims at building a scheduler for concurrent query processing on Hustle. A scheduler schedules the execution of operators which are extracted from the physical plan generated by the optimizer. The execution of a query is essentially the execution of a sequence of work orders. To realize the parallel execution of multiple queries and pipelining between the execution of operators, multiple workers are created, with each associated with a CPU core. The scheduling policy is independent of the mechanism and can be changed to other more sophisticated ones.
9:20-9:30	Evaluating MySQL, PostgreSQL and SparkSQL database systems using TPCC & YCSB Benchmarks	Sangeetha Sampathkumar, Shebin Roy Yesudhas	Our project aims at comparing database systems such as MySQL, PostgreSQL and SparkSQL using two benchmarks namely TPC-C and YCSB. In some scenarios, the use case of application just requires databases that could fit on a single machine. Given that we have different type of database offerings both on SQL and NoSQL paradigm that could be run on a single machine, benchmarking is crucial. We plan to compare the mentioned database systems by measuring the performance of each of them in terms of throughput and latency using OLTP benchmark framework which emulates TPC-C and YCSB benchmark workloads.

Wednesday December 11, 2019			
8:00-8:10	Reducing Lock Contention through a New Controlled Lock Violation Protocol for Main-memory Database	Zhihan Guo	Inspired by the idea of controlled lock violation, we proposed a new protocol based on 2PL to help reduce lock contention for main-memory databases. The original controlled lock violation protocol focuses on reduce the waiting time for hardening, while ours aims to reduce the waiting time for acquiring the lock, which addresses the new bottleneck in main-memory databases in contrast to traditional databases. We evaluated our methods against the basic 2PL over TPCC benchmark to see how we improve the OLTP application performance under high contention.
8:10-8:20	Latch Free Concurrent Hashing Data Structure Performance Analysis	Madan Raj Hari, Raghavan Vellore Muneeswaran	The goal of the project is to implement Cuckoo Hashing, a concurrent latch free hash table and study its performance as the key size and data size changes. A Hash Table is a data structure which allows us to perform rapid storage and retrieval operations. Concurrent Hash tables are widely used in many computer systems and applications. A data structure is latch-free (lock-free) if it is nonblocking and guarantees global progress. There must always be one thread finishing its operation in a finite number of steps. Latch Free solutions are better than lock based solutions when contention is high and machine architecture is unknown. However, latch free solutions are difficult to get correctness in relocation operation, memory reclamation and ABA problem.
8:20-8:30	Concurrency Control for Cache-Sensitive B+-Trees	Arjun Balasubramanian, Vinith Venkatesan	B+-Trees have now become commonplace in large database systems for fast indexing. With main memories becoming larger, variants such as CSB+-Trees have been proposed so as to improve the utilization of a cache line and thereby give better performance. However, concurrency control remains unaddressed in these structures. In this work, we explore two concurrency control options for CSB+-Trees and we evaluate the performance implications of these two approaches.
8:30-8:40	Lock-free Concurrent Linear Probing Hash Table in Rust	Kaiwei Tu	The concurrent hash table has wide applications in a modern in-memory computer system. Especially, a lock-free concurrent hash table that benefits from low-overhead hardware atomicity is crucial to get higher concurrency and better performance. However, how to efficiently support complex key and value space under compare-and-swap (CAS) instruction stills needs more experiments. This project aims at prototyping a lock-free concurrent linear probing hash table in Rust. Based on this prototyping system, we will conduct several experiments on the overhead induced by supporting complex key-value space and cost of changing key and value size.
8:40-8:50	Analyzing the Scalability of Latch-Free Hash Tables in Rust	John Truskowski, Sapan Gupta	Hash tables are a quintessential data structure for many applications due to their O(1) insertion and lookup times (in the best case). However, due to the increasing scale of data, the need for concurrent reads and writes to these hash tables grows ever more important. A critical data structure for this is the latch-free hash table, as traditional methods for controlling concurrent access (locks) introduce significant overhead and degrade the performance. Latch-free hash tables guarantee that when one or more active threads are performing an operation on data structure, at least one thread will be able to complete within a finite number of steps, agnostic of other threads. These hash tables must perform well in the presence of contention, deletion and growing input sizes. We explore how a Rust implementation of latch-free hash tables performs when varying input size and key size for a variety of common database workloads.

8:50-9:00	Enhancing concurrent cuckoo hashmap using optimistic locking and incremental migration	Sandhya Kannan, and Sri Harshal Parimi	We aim to improve the performance of concurrent cuckoo hashmap implementation using the following techniques: Instead of acquiring locks to perform consistent reads, we adopt a fine-grained optimistic locking approach using version counters. Since operations on the cuckoo hashmap are generally stalled when the entire table is locked for migration, we implement an incremental migration of entries from the older hashmap to the newer hashmap. We plan to compare performance with the existing cuckoo hashing implementation and derive insights that generalize to the implementation of concurrent data structures.
9:00-9:10	SQL-Only Neural Networks	Naazish Sheikh, Samuel Erickson	This project explores the performance of a neural network in PostgreSQL in an attempt to expand the capabilities of SQL-only neural networks as well as improve their efficiency. The primary goal of this project is to create a SQL-only neural network implementation that performs more efficiently than a traditional neural network that simply pulls data from the database. For our project we selected a Major League Baseball pitch data set on which we trained our neural networks to predict pitch type. This data set contains numerous mathematical attributes such as pitch speed and trajectory as well as pitch type, e.g. curveball. We stored our data set in a cloud database through ElephantSQL, which is a PostgreSQL-as-a-service hosting framework. This provides us with all of the advantages of a modern DBMS (concurrency control, recovery, etc.) in a fully-managed, scalable environment. Using a connection through pgAdmin 4, database tables were created and the MLB pitch data set was imported into said tables from CSV files. We started by creating a baseline against which we could compare our optimizations. We did so by implementing a neural network in Python using scikit-learn, from which we gathered performance metrics on three different sized subsets of our data set (small, medium, and large). The performance metrics we are primarily concerned about are overall training speed for the neural network and the accuracy of the predictions made from the neural network. To make sure the prediction accuracy of each neural network implementation was reasonable to compare, our neural networks in both implementations are multi-layer perceptron classifiers that use back propagation.
9:10-:9:20	Reliable ML Based Selectivity Estimation	Ankur Goswami, Zifan Liu, and Yunjia Zhang	
9:20-9:30	Using Neural Networks for predicting query execution time	Na Li and Askar Safipour Afshar	Query execution time prediction is an important and challenging problem in database systems. Especially for applications that handle large amounts of data or where time loss and deadlocks are hardly tolerated, it is very useful to predict the query execution times before actual execution. In this project we aim to predict query execution times automatically using neural network-based approaches, and compares these approaches in terms of training time and accuracy. We implemented neural networks methods, and compared these methods using the TPC-DS benchmark database. The results of this project show that multilayer perceptron with back-propagation presents accurate results in predicting query execution times within acceptable training times.

If you are looking for the CS 764 course home page, click here.