Msc Projects 2017-2018

CWI has a world-class data management research group, which has created multiple advanced systems for data analysis. Its core expertise are high-performance query evaluation techniques that connect the worlds of databases, compilers and computer architecture.

Some of these projects are with Databricks. The expertise of CWI recently has led Databricks to open a R&D office in Amsterdam. Databricks’ main focus is the development of the Apache Spark framework, and the Databricks Amsterdam R&D office is focused on Spark performance in all its facets. The office is expanding, and Databricks is looking at MSc projects as a possible avenue for recruitment. Expect a selective US tech-company style interview process when applying for a Databricks topic...

All below projects represent cutting edge data management research projects, that if successfully executed might also lead to a research publication.

Adaptive Partial Multi-dimensional indexing
Adaptive partial indexing techniques like Database Cracking have emerged in recent years and proven useful and successful in scenarios where a-priori construction of complete indexes is unfeasible due to lack of resources of lack of a-priori workload knowledge. While intensely studied for single-attribute indexes / single-dimensional data, adaptive indexing techniques for multi-dimensional data have not been proposed or studied, yet. On "obstacle" is the fact that existing adaptive indexing techniques, just like classical single-dimensional indexing techniques assume a total sorting order, which is not given in multi-dimensional data. A first attempt towards adaptive partial indexing for multi-dimensional data could be to adopt the classical technique of reducing multi-dimensional data to a single dimension using space-filling curves (e.g., Hilbert-curve or Z-ordering), and apply, say, database cracking on top of that. In addition to exploring the opportunities and challenges of this route, other projects could explore the space of devising adaptive partial indexing techniques inspired by multi-dimensional tree structures (R-Trees, kd-trees) or gridding techniques.

You will be advised by Stefan Manegold of CWI.

Automatically Detecting Database Performance Regressions
Measuring database performance is an important quality assurance tool for development. Over time, bugs are fixed and performance is hopefully improved. However, manually running a set of benchmarks every time a change is made is prohibitively expensive when done manually. Along the lines of Continuous Integration (CI), we propose a system to perform Continuous Benchmarking. For every version control check-in, a set of benchmarks is to be run and results (e.g. runtime, disk IO etc.) recorded.

However, this creates a huge amount of numbers that are difficult to interpret. The main challenge of this thesis is to automatically and statistically accurately detect performance regressions or improvements according to a certain confidence level and the standard deviation of previous runs on same hardware, configuration, dataset size etc. The topic will also encompass a usable real-world implementation and integration of the new system, in particular in the MonetDB/MonetDBLite project.

You will be advised by Hannes Muehleisen of CWI+VU.
**Time-Travel for interactive analysis**

Interactive data analysis in frameworks like R and SciPy is highly popular. However, a common issue is the modification of potentially large datasets within an interactive session. Unintentional modification is frequent, and the common solution is to re-run the steps that were required to get from a data file to the point in question. This reduces the usability of the analysis tools, makes “what-if” exploration difficult, and creates a lot of unnecessary overhead for either manually saving state or re-running scripts to recreate it. The proposed project will investigate the use of relational Multi-Version Concurrency Control (MVCC) for these interactive workloads. Is it possible to adapt these well-known concepts to data analysis beyond declarative SQL? The topic will also encompass a real-world implementation to showcase its potential, for example in the R environment.

You will be advised by Hannes Muehleisen of CWI+VU.

**DSL Optimisation for Data Analysis**

Recent years have seen astonishing progress in the programming language field regarding the automatic optimisation of Domain-Specific Languages (DSL). However, one of the most well-known DSLs, the relational query language SQL still requires highly complex and specific software components for its optimisation. This is partly due to historic reasons, and partly due to the dependence of SQL execution efficiency to data cardinalities and distributions. The proposed project will investigate whether state-of-the-art metaprogramming systems such as Rascal can be used to express SQL and its static and dynamic optimisation rules. A proof-of-concept implementation will be used to determine the feasibility of the approach, comparing with common data management systems’ optimisers.

You will be advised by Hannes Muehleisen of CWI+VU.

**NVM storage in MonetDB for tables and graphs**

MonetDB is a relational columnar and SQL-compatible DBMS, designed towards analytical workloads. In the runtime, data is stored in contiguous memory chunks, like arrays. If the contiguous data should hold an index, or rather, a (semi) sorted sequence of values that support sub-linear lookups by value, the challenge is how to handle updates at sub-linear cost as well. Moreover, we would like this data structure work well on non-volatile memory (NVM), such as 3D-XPoint. A final consideration for the data structure should be its ability to provide isolation during transaction processing, also at sub-linear cost.

NVM memory is a bit slower than RAM, but persistent. The difficulty in handling it is in ensuring consistency, because writes may be reordered and in case of a crash there are no guarantees on what changes were flushed back already to RAM, and which not (though there are operations to explicitly enforce this). The data structure could somehow be akin to Packed Memory Arrays (PMAs), a cache-oblivious data structure well-known in theoretical computer science, but increase the locality of writes to fit NVM well.

A stretch-goal of the project is to use the data structure for storing updatable graphs. A graph consists of two tables \((V,E)\), where E is stored in source-vertex order and each V stores the offset its first edge; such that it becomes very cheap to discover all edges starting in one particular vertex. Now, an edge insert may cause all subsequent edge offsets in the V table to change, yet the data structure should support updates at sub-linear cost (and be NVM friendly and support isolation).

You would be advised by Dean de Leo at CWI, and co-advised by Peter Boncz of CWI+VU.
**MonetDB-Weld Compiler**

Database and compiler technologies are rapidly converging, and recent research has proposed various Domain Specific Languages (DSLs) that facilitate compiling high-level queries (such as SQL or Spark scripts) into low-level machine code. Weld is such a DSL proposed by MIT and Stanford, and it is able to generate highly performant low-level code both for CPUs and GPUs.

In this project we would like to see how efficient Weld can be for relational databases and for that purpose would like you to generate a SQL-to-Weld translation for the MonetDB open-source column-store database system of CWI.

You would be advised by Peter Boncz of CWI+VU, with occasional contacts with Matei Zaharia and the Weld team at Stanford.

**Weld Vectorized Interpreter**

Database and compiler technologies are rapidly converging, and recent research has proposed various Domain Specific Languages (DSLs) that facilitate compiling high-level queries (such as SQL or Spark scripts) into low-level machine code (e.g. using LLVM). Weld is such a DSL proposed by MIT and Stanford, and it is able to generate highly performant low-level code both for CPUs and GPUs.

However, an alternative approach to modern high-performance query execution is “vectorized query processing”, as realized in the VectorWise systems and some of its successors. There are numerous advantages to vectorized processing, such as better suitability for SIMD execution (e.g. AVX512 instructions) as well as better profiling and debugging opportunities. For this purpose we would like you to create a (vectorized) interpreter for Weld scripts, and compare the performance of this interpreter with the existing low-level code generation backend of Weld.

You would be advised by Peter Boncz of CWI+VU, with occasional contacts with Matei Zaharia and the Weld team at Stanford.

**Updatable Spark GraphFrames**

One of the most challenging data management workloads is support for complex queries on continuously changing graphs, such as social networks. This complexity arises on the one hand from the need to use graph indexes and parallel execution in order to make these graph queries fast on large data; and on the other hand the need to continuously update these distributed graph indexes, since the data is continuously changing.

The scope for this project is to support the Spark GraphFrames API on top of updatable graphs. A basic building block for this project is already present, namely a new kind of Spark DataFrame called the IndexedDataFrame. An IndexedDataFrame is a DataFrame with a built-in index, that allow to quickly find data by a key value, that can be kept inside the Spark in-memory cache.

Thus, the main task is to rewrite the GraphFrames API to use IndexedDataFrame. Other tasks include benchmarking this solution on the Social Network Benchmark of LDBC.

You would be advised by Peter Boncz of CWI+VU and Alex Uta of VU; with regular contacts to the Databricks Amsterdam R&D office.
Databricks-CWI Msc Projects 2017-2018

Databricks Workload Analysis
Databricks operates a cloud-based Spark-as-a-service offering with many customers storing lots of data and executing Spark workloads on this data. For Databricks and for researchers in general, it would be highly beneficial to better understand the characteristics of the data and workloads of these data science users, as such knowledge can expose problems and bottlenecks to focus new research on.

This project involves studying the user logs of Databricks as well as the generation of more performance profiling information in the Databricks infrastructure.

You would be advised by Peter Boncz of CWI+VU and Alex Uta of VU; with regular contacts to the Databricks Amsterdam R&D office.

A profiler for Catalyst query plans
The Tungsten project in Spark introduced on-the-fly compilation of Spark code into Java code, which significantly enhanced its query execution performance. This advanced compilation method fuses all non-materializing operators in a pipeline into a single for-loop. A negative consequence, however, is that it has become more difficult to understand the performance of Spark scripts, which both affects end-users of Spark as well as the developers (who are interested in which operators to improve).

In this project, we would like you to add profiling and debugging information to the generated java code, such that a profiler can relate lines of Java code to the relational operators in the Spark query plan, as generated by its Catalyst optimizer. Thus, it should become possible to relate (an estimate) of execution time to each individual operator (e.g. scans, aggregations, joins) in the plan. A stretch goal is to also allow debugging of such generated code, where we would like to be able to step through query plans and see not only which operator generated a java statement, but also which line of Spark query generator generated the line.

You would be advised by a member of the Databricks Amsterdam office in conjunction with Peter Boncz of CWI+VU.

Late Decoding of Parquet Dictionaries
Parquet files are an often-used data format in Spark workloads. Parquet compresses data in multiple ways, and in case of string data this often uses dictionary compression.

Access by Spark SQL (or any Spark scripts on DataFrames) could be made much more efficient if predicates on strings could directly operate on the compressed data (dictionary-compressed strings). To make this possible, we propose an infrastructure that will allow late decompression of frequently occurring dictionary strings in an in-memory data structure that keeps track of the most frequently used dictionary codes. Often used operations, such as comparing and hashing strings could be translated into operations on integers and significantly accelerate Spark while reduce memory usage.

You would be advised by a member of the Databricks Amsterdam office in conjunction with Peter Boncz of CWI+VU.
Compact hash tables in Tungsten
Some of the most important operations on DataFrames involve storing DataFrames in in-memory hash tables (or hash-maps). The current Tungsten backend of Spark, however, makes little effort to compactly organize these hash tables. Making these data structures smaller is important because this reduces the memory needs of Spark workloads, but also the pressure on the CPU they generate, and therefore directly affects the speed of operations.

In this project, you will improve the layout of hash-tables and reduce their size by dynamic data compression, and measure the effect of these optimizations on commonly used Spark workloads.

You would be advised by a member of the Databricks Amsterdam office in conjunction with Peter Boncz of CWI+VU.

Predicate Pushdown in Parquet Scans
Given its wide use, Databricks is interested in accelerating the performance of scans on Parquet files. Parquet is a compressed columnar open-source data format. The Scan operator reads all data from a Parquet file and decompresses it. We would like you to work on so-called predicate-pushdown functionality, which means the execution of certain simple (but common) filter predicates on the Parquet data before it is decompressed. Typically, this concerns equality or range comparisons of a column against constant values.

This project may also involve executing parts of this workloads not in Java but in native (C++ or C) code, possibly also involving SIMD instructions.

A stretch goal of the project would be to also support join optimizations, only decompressing tuples from the Parquet files whose key is inside a bloom filter created by an upstream join.

You would be advised by a member of the Databricks Amsterdam office in conjunction with Peter Boncz of CWI+VU.