List of Project Topics (Proposals)

Last update: Oct 20, 2024

Most LDE Projects in the Context of Two Open-source Systems

- **DAPHNE EU-project**
  https://github.com/daphne-eu/daphne
  - Focus on integrated data analysis pipelines
  - Project implementation mainly in C++

- **Apache SystemDS**
  https://github.com/apache/systemds
  - Focus on the end-to-end data science lifecycle
  - Project implementation mainly in Java and DML

Topics in DAPHNE

Implementation mainly in C++

#856: Integration of DAPHNE with Database Systems

- **Motivation**
  - When the data to analyze resides in a relational database, it must be exported to a file (e.g., CSV) before it can be read in DAPHNE, which is inefficient
- **Task** (in C++)
  - Extend DAPHNE'S existing sql() function, such that it can process the given query over an existing database inside an existing DBMS instance (e.g., DuckDB, MySQL, PostgreSQL, …)
  - Implement and compare different methods to import the query result in DAPHNE (e.g., ODBC, ConnectorX, …)
- **More information & hints**
  - https://github.com/daphne-eu/daphne/issues/856
  - Contact: Patrick Damme

#857: Speed-up Repeated Read of Data Set Files w/ Storage Budget

- **Motivation**
  - Data science workflows are exploratory by nature; thus, variants of the same workflow are executed repeatedly, although they often access the same data set files
  - When loading a data set file, a lot of information needs to be found out by the system (e.g., column types, separator positions in CSV files, number formats etc.), which makes reading a file more expensive
- **Task** (in C++)
  - Explore different means to retain information gathered about a data set file across independent executions of a DaphneDSL script for future reuse; these could range from reusing the detected column types over creating suitable indexes to replicating the required part of the data in a more I/O-efficient format on storage
  - Think of meaningful schemes to exploit a given storage budget for the most useful information
  - Detect changes to the data set outside of DAPHNE and invalidate the auxiliary information as required
- **More information & hints**
  - https://github.com/daphne-eu/daphne/issues/857
  - Contact: Patrick Damme

#858: Compare Different String Representations

- **Motivation**
  - String data is commonplace in real-world data sets, and DAPHNE supports string-values matrices/frames
  - There are many different ways to represent strings (or a sequence of string) with different trade-offs regarding efficiency (memory footprint, runtime of various operations) and simplicity
- **Task** (in C++)
  - Implement and compare additional, more advanced string representations as new value types in DAPHNE
  - A range of typical operations should be supported on these types (e.g., comparison, transformations, concat, …)
  - Exploit data characteristics to design efficient algorithms on string-valued matrices/frames
  - Examples include: Umbra strings, dictionary encoded strings, prefix-encoded strings, etc.
- **More information & hints**
  - https://github.com/daphne-eu/daphne/issues/858
  - Contact: Patrick Damme

#691: Hybrid Compilation

- **Motivation**
  - By default, DAPHNE lowers domain-specific operations from linear and relational algebra operations to calls to pre-compiled C++ kernels (physical operators).
  - However, being based on MLIR/LLVM, on-the-fly code generation for these operations is an alternative option.
  - Interestingly, both approaches have their pros and cons, so a hybrid approach (choosing pre-compiled kernels for some operations and using code generation for others) is desirable.
- **Task** (in C++)
  - Extend DAPHNE's existing proof-of-concept for a hybrid compilation chain by additional features such as:
    - Support for the Frame data type and relational algebra operations thereon
    - Support for sparse matrix data types (e.g., CSR) and linear algebra operations thereon
    - Support for zero-copy views into row/column segments of matrices and frames
    - Integration of code-generated operations with DAPHNE's vectorized pipelines
    - Automatic decision on whether to use a pre-compiled kernel or to generate code
- **More information & hints**
  - https://github.com/daphne-eu/daphne/issues/691
  - Contact: Philipp Ortner

#693: Codegen-only Compilation Pipeline for LA Operations

- **Motivation**
  - By default, DAPHNE lowers domain-specific operations from linear and relational algebra operations to calls to pre-compiled C++ kernels (physical operators).
  - However, being based on MLIR/LLVM, on-the-fly code generation for these operations is an alternative option, which could, e.g., simplify targeting hardware accelerators through existing MLIR/LLVM backends.
- **Task** (in C++)
  - Implement a proof-of-concept for a compilation pipeline in DAPHNE, which only involves code-generated operations, thereby reducing the compatibility challenges of a hybrid compilation chain.
  - For simplicity, you can focus on linear algebra operations, supporting dense and sparse matrices.
  - The resulting code may target x86 CPUs or, optionally CUDA-enabled GPUs.
- **More information & hints**
  - Working on the GPU aspect is optional and requires a GPU with CUDA support
  - https://github.com/daphne-eu/daphne/issues/693
  - Contact: Philipp Ortner

#859: Advanced Test Infrastructure

- **Motivation**
  - Any complex system requires a good test coverage to detect bugs and performance regressions
  - To achieve this, writing test cases should be easy and their execution automated as much as possible
  - DAPHNE already has a lot of unit and script-level tests, however, some important aspects are still missing
- **Task** (in C++, DaphneDSL, bash, Python)
  - Introduce an infrastructure for automated performance regression tests, that run certain benchmark DaphneDSL scripts automatically, compare the runtime performance to previous runs, and report regressions
  - DSL fuzzing: Automatically generate random, simple and complex, equivalent scripts in DaphneDSL/DaphneLib and a baseline (e.g., Python, Julia, R) and compare their outputs to each other to find bugs in DAPHNE.
  - Introduce a means to easily write a high number of script-level test cases that are executed independently of each other
- **More information & hints**
  - https://github.com/daphne-eu/daphne/issues/859
  - The focus of this task is on improving the test infrastructure, not on adding the actual test cases
  - Contact: Patrick Damme

#690 IDE/Tooling Support for DaphneDSL and DaphneIR (new)

- **Motivation**
  - DaphneDSL is DAPHNE's domain-specific language for integrated data analysis pipelines.
  - Its syntax is inspired by languages like Python, R, and C.
  - DaphneDSL can be written in any text editor, but support in an integrated development environment would increase user productivity.
- **Task**
  - Implement support for DaphneDSL in a widely-used IDE (preferably VS Code), including a LSP and TreeSitter.
  - The tool should be connected to the DAPHNE compiler, especially to its features for type/shape/property inference in order to augment the DaphneDSL code with additional information
- **More information & hints**
  - https://github.com/daphne-eu/daphne/issues/690
  - Contact: Philipp Ortner

Topics in Apache SystemDS

Implementation mainly in Java and DML (SystemDS's R-like domain-specific language)

#3777: Built-in Function for ADASYN

- **Task** (in Java and DML)
  - Add a new script-level built-in function adasyn() for oversampling the minor class in imbalanced training data sets via the ADASYN method
  - SystemDS already implements a built-in function for SMOTE, which can be used as an example
  - In detail this task should
    - Start by implementing the corresponding junit tests (for local and spark operations)
    - Register the new built-in function
    - Implement the ADASYN built-in function
    - Evaluate the new method on a number of real datasets
- **More information & hints**
  - https://issues.apache.org/jira/browse/SYSTEMDS-3777
  - Contact: Matthias Boehm

#3778: Determinant Computation Primitives

- **Task** (in Java and DML)
  - Add a natively supported built-in function **`s=det(A)`** for computing the determinant of a squared matrix
  - Internally, these operation can either be implemented through dedicated kernels or a rotation of the matrix and subsequent row and column sums
  - In detail, the task should be approached in 3 subsequent PRs (where the 1st PR comprises tests and local ops)
    - Tests and built-in function registration
    - Local in-memory operations
    - Distributed Spark operations
    - Two or three simplification rewrites
      (e.g., **`det(t(A)) --> det(A)`**, and **`det(A%*%B) --> det(A)*det(B)`**)
- **More information & hints**
  - https://issues.apache.org/jira/browse/SYSTEMDS-3778
  - Contact: Matthias Boehm

#3556: Counter-based Random Number Generation

- **Motivation**
  - SystemDS's random number generation is based on a sequential number generation of individual blocks of BlockSize x BlockSize
  - This makes the generation of matrices parallelized at maximum at the number of blocks
  - This technique is inefficient on GPUs
- **Task** (in Java and DML)
  - Implement a counter-based random generation of blocks that work well on both CPU and GPU
  - The ideal implementation produces equivalent results to cuRAND (https://docs.nvidia.com/cuda/curand/index.html)
  - Generate values based on the block ID associated to make the generation compatible with Spark, but inside each block, the values should be generated based on the counter-based number generator
- **More information & hints**
  - https://issues.apache.org/jira/browse/SYSTEMDS-3556
  - Contact: Sebastian Baunsgaard

Additional Compression Formats for Matrix Column Groups

- **#3539: Delta Encoding** (in Java)
  - Reader for uncompressed matrices to be encoded into compression statistics as if delta-encoded
  - Transforming operations such as cumulative sum that have to be wired to transform existing column groups into a delta-encoded column group
  - Compression taking an uncompressed matrix and encoding a delta-encoded column group from it without materializing the delta encoded version of the input matrix
- **#3543: Piece-wise Linear Compression** (in Java)
  - Implement a new column group for piece-wise linear compression that is based on a target loss
  - The technique compresses a column of values, into smaller line segments
  - A naive implementation of this in the extreme cases would potentially be 0 target loss, with full allocation of input, and 100% target loss containing only the average of all values
  - Other than this, the implementation moves from a lossless array into a lossless piece-wise linear representation via dynamic programming.
- **#3779: LZW Compression** (in Java)
  - Add a LZW compressed column group, that builds on top of the DDC group, only compressing the mapping data with LZW compression
  - Use a modified LZW algorithm that returns equal size codes for each code compressed
  - It should be possible to allocate a temporary mapping for the index to enable efficient processing directly on the compressed representation
- **More information & hints**
  - https://issues.apache.org/jira/browse/SYSTEMDS-3539, https://issues.apache.org/jira/browse/SYSTEMDS-3543, https://issues.apache.org/jira/browse/SYSTEMDS-3779
  - Contact: Sebastian Baunsgaard

#3541: Exploratory Workload-aware Compression on Intermediates

- **Task** (in Java)
  - This project is to experiment with enabling compression on intermediate values.
  - Currently, the project supports compression of arbitrary intermediate values.
  - The goal of this project is to enable this feature, experiment with it across a number of algorithms, and report results, bugs and interesting findings.
  - In this process, if regressions or limitations are found, solutions are proposed and implemented.
- **More information & hints**
  - https://issues.apache.org/jira/browse/SYSTEMDS-3541
  - Contact: Sebastian Baunsgaard

#3780: Compression Fused Quantization

- **Task** (in Java)
  - Add a new instruction.
  - Assuming M is an matrix and S is a scaling constant, vector, or matrix; then implement a rewrite and instruction that fuse the following sequence of instructions:
    - ```
      M2 = floor(M * S)
      C = compress(M2)
      ```
  - The result should be a CLA compressed quantized matrix according to the scaling values of S
  - In a passing implementation S is a scalar constant value
- **More information & hints**
  - https://issues.apache.org/jira/browse/SYSTEMDS-3780
  - Contact: Sebastian Baunsgaard
      ```

#3548: Optimize I/O Path of SystemDS Python Interface

- **Task** (in Java and Python)
  - This task is to optimize the data transfer between SystemDS and Python
  - Two starting points are string transfer of pandas data frame data both to and from SystemDS and boolean transfer from SystemDS that could be bit packed
  - The task includes benchmarking the interface to know how the performance is currently and what the limiting factors are
  - There is also an opportunity to try out parallel data transfer between the environments
  - There are multiple low hanging fruits for this task:
    - String transfer – This is currently done by transferring a single string at a time, making it unbearably slow because it calls Py4J once per cell.
    - Bit packed transfer – This is currently done by unpacking bits from a long into individual bytes, making the transfer 8x larger than it is supposed to be.
- **More Information & Hints**
  - https://issues.apache.org/jira/browse/SYSTEMDS-3548
  - Contact: Sebastian Baunsgaard

More Topics in Apache SystemDS

- **Loop Vectorization** (in Java)
- **Extended Common Subexpression Elimination** (in Java)
- **Extended I/O Framework: Readers/Writers for More File Formats (NetCDF, HDF5, Arrow)** (in Java)
- **#3650 I/O Support for Cloud-optimized GeoTIFF (COG)** (in Java)
- **Various Model Debugging or Data Preprocessing Strategies** (in DML)

- **See also the full list of available student projects in SystemDS:**
  https://issues.apache.org/jira/secure/Dashboard.jspa?selectPageId=12335852#Filter-Results/12365413

- **More information & hints**
  - Contact: Matthias Boehm

Stand-alone and Cross-cutting Topics

Benchmarking Polynomial Approximation for Time Series Compression

- **Task** (in your language of choice: Zig, C/C++, Java, Python, or Rust)
  - Compare different methods for compressing time series data
  - Implement lossless compression methods for comparison, e.g.
    - Optimal Piecewise Linear Approximation (https://doi.org/10.1109/TSP.2006.875394)
    - Facebook's Gorilla Lossless Compression (https://www.vldb.org/pvldb/vol8/p1816-teller.pdf)
    - Mix-Piece Linear Approximation (https://link.springer.com/article/10.1007/s00778-024-00862-z)
- **More information & hints**
  - Many algorithms are already implemented in TerseTS (a library for time series compression) which we will reuse
  - The implemented algorithms shall be added to the open-source TerseTS library (written in Zig)
  - Depending on the team size, other algorithms can be added for implementation
  - Contact: Carlos E. Muniz Cuza

Alternative: Propose Your Own Topic Idea

- **We are open to additional topic proposals**
  - In the context of data engineering, data management, and machine learning systems
  - If you are passionate about your idea
  - More topics in SystemsDS and DAPHNE or other open-source systems possible,
    but contributions might be more difficult to get accepted
  - **If you would like to propose your own topic**, **approach me by email by Oct 31, 23:59 CET**;
    in any case, **also fill in the poll regarding the topic selection with your preferred topics from the list above**