List of Project Topics (Proposals)

Last update: Apr 25, 2025

LDE Projects in the Context of Our 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++ and DaphneDSL, some also in Python

https://github.com/daphne-eu/daphne/issues?q=is%3Aissue%20state%3Aopen%20label%3A%22LDE%20summer%202025%22

#955 Locality Sensitive Hashing (LSH)

- **Motivation**
  - LSH is a powerful technique for efficient approximate nearest neighbor search in high-dimensional spaces.
  - Unlike traditional hash functions that aim to minimize collisions, LSH is designed so that similar items have a higher probability of colliding.
  - This property makes LSH invaluable for many data-intensive applications including recommendation systems, duplicate detection, and clustering large datasets.
- **Task** (in C++ and DaphneDSL)
  - Design and implement a LSH algorithm in DaphneDSL.
  - Explore different variants of LSH (e.g., MinHash, SimHash, random projections)
  - Create user-friendly functions that integrate well with DAPHNE's existing APIs.
  - Demonstrate the effectiveness of LSH through example applications.
- **More Information & Hints**
  - https://github.com/daphne-eu/daphne/issues/955
  - Contact: Philipp Ortner

#956 Large Language Model (GPT-2) Inference

- **Motivation**
  - Large Language Models (LLMs) are enabling hitherto unknown applications.
  - GPT-2 is a transformer-based language model that can generate coherent text based on a given prompt.
- **Task** (in C++, DaphneDSL/Python)
  - Implement GPT-2 inference (forward pass) in DAPHNE, to this end:
    - Understand the architecture and operation of transformer-based language models.
    - Implement required and missing operators in the DAPHNE system (C++).
    - Implement required and missing library functions (either Python+DaphneLib or DaphneDSL).
    - Performance experiments may investigate, e.g., the generated tokens/second.
  - Groups may implement additional optimizations beyond the basic functionality and add more models.
- **More Information & Hints**
  - https://github.com/daphne-eu/daphne/issues/956
  - Contact: Philipp Ortner

#957 Efficient Sparse Data Transfer between DAPHNE and Python Libraries

- **Motivation**
  - Sparse matrices containing mostly zeros is commonplace in many applications (e.g., graph processing).
  - Various efficient physical representations for sparse data (e.g., CSR, CSC, COO)
  - DAPHNE needs to integrate with the existing Python-based data science ecosystem to achieve user adoption.
  - Efficient data transfer is key and DAPHNE partly supports it, but sparse representations are still missing.
- **Task** (in C++ and Python)
  - Implement efficient (ideally zero-copy) bidirectional data transfer of sparse data between DAPHNE and famous Python libraries like SciPy, TensorFlow, and PyTorch.
  - Extend the existing data transfer for dense numpy arrays by support sparse data structures.
  - Efficiently handle data and value type conversions between different physical matrix representations.
  - Showcase the efficient data transfer by offloading computations from Python to DAPHNE
- **More Information & Hints**
  - https://github.com/daphne-eu/daphne/issues/957
  - Contact: Patrick Damme

#512 Simplification Rewrites for Linear and Relational Algebra

- **Motivation**
  - User programs/queries are typically parsed into an initial, unoptimized internal representation (IR).
  - This initial IR usually offers large potential for performance optimization through simplification rewrites, which reorder, remove, or replace operations when certain patterns are detected (e.g., `t(t(X)) -> X`).
  - The smart application of simplification rewrites can increase performance and reduce memory consumption by orders of magnitude; but so far, DAPHNE has only limited support for simplification rewrites.
- **Task** (in C++)
  - Design a simplification rewrite system as a part of DAPHNE's MLIR-based optimizing compiler.
  - Implement a good set of useful static and dynamic rewrites for linear and relational algebra.
- **More Information & Hints**
  - https://github.com/daphne-eu/daphne/issues/512
  - Contact: Patrick Damme

#511 Efficient Parallel Hash-Join Operator for Speeding up the SSB

- **Motivation**
  - The Star Schema Benchmark (SSB) is a well-known benchmark for analytical query processing.
  - The runtime of most SSB queries is dominated by PK-FK joins and semi-joins.
  - An efficient join implementation is crucial for achieving good results in this benchmark.
- **Task** (in C++)
  - Implement an efficient parallel hash-join (separate build and probe) operator on columnar data.
  - Devise an efficient hash table implementation and support it for intermediate results in DaphneIR.
  - Parallelism should be achieved through multiple threads (MIMD), and optionally also through SIMD operations.
  - In a first step, multi-threading could be applied inside the operator; based upon that, an integration with DAPHNE's vectorized engine could be tackled.
- **More Information & Hints**
  - https://github.com/daphne-eu/daphne/issues/511
  - Contact: Patrick Damme

#521 Efficient Matrix Multiplication for Generic Value Types

- **Motivation**
  - Matrix multiplication is a central operation in many machine learning and data analysis algorithms.
  - Various libraries (e.g., BLAS) provide highly optimized implementations, but are typically limited to certain data and value types, especially dense matrices of single/double-precision floating point values.
  - DAPHNE strives to be extensible w.r.t. to data and value types, thus it needs efficient matrix multiplications for other types, too.
- **Task** (in C++)
  - Implement efficient matrix multiplication for various combinations of dense/sparse inputs/output, different values types (e.g., integers, bool), and input shapes (e.g., matrix-matrix and matrix-vector).
  - On the one hand, write hand-tuned kernels for a handful of cases
  - On the other hand, devise a generic implementation that comes as close as possible to these specialized ones.
  - Showcase the benefit of your kernels on ML algorithms dominated by matrix multiplications.
- **More Information & Hints**
  - https://github.com/daphne-eu/daphne/issues/521
  - Contact: Patrick Damme

#690 IDE/Tooling Support for DaphneDSL and DaphneIR

- **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

https://issues.apache.org/jira/secure/Dashboard.jspa?selectPageId=12335852#Filter-Results/12365413

#3862 & #3859 Relational Query Processing in SystemDS

- **#3859 Improved Relational Algebra Built-in Functions**
  - SystemDS already has DML-bodied built-in functions for relational algebra operations which were inspired by prior work (https://www.vldb.org/pvldb/vol15/p2811-he.pdf). This task aims to improve these operations by first running systematic benchmarks and subsequently improving their runtime by script-level changes or system-internal rewrites and improved runtime kernels.
- **#3862 SSB Benchmark Implementation**
  - Implement the queries of the Star Schema Benchmark (SSB) via the existing relational algebra operators `raSelect()`, `raJoin()`, and `raGroupBy()` as DML scripts and prepare shell scripts that use the SSB data generator and these newly created scripts to run the benchmark at different scale factors.
- **More Information & Hints**
  - https://issues.apache.org/jira/browse/SYSTEMDS-3859
  - https://issues.apache.org/jira/browse/SYSTEMDS-3862
  - Contact: Matthias Boehm

#3855 & 3856 Java Vector API: SIMD Operations in Java

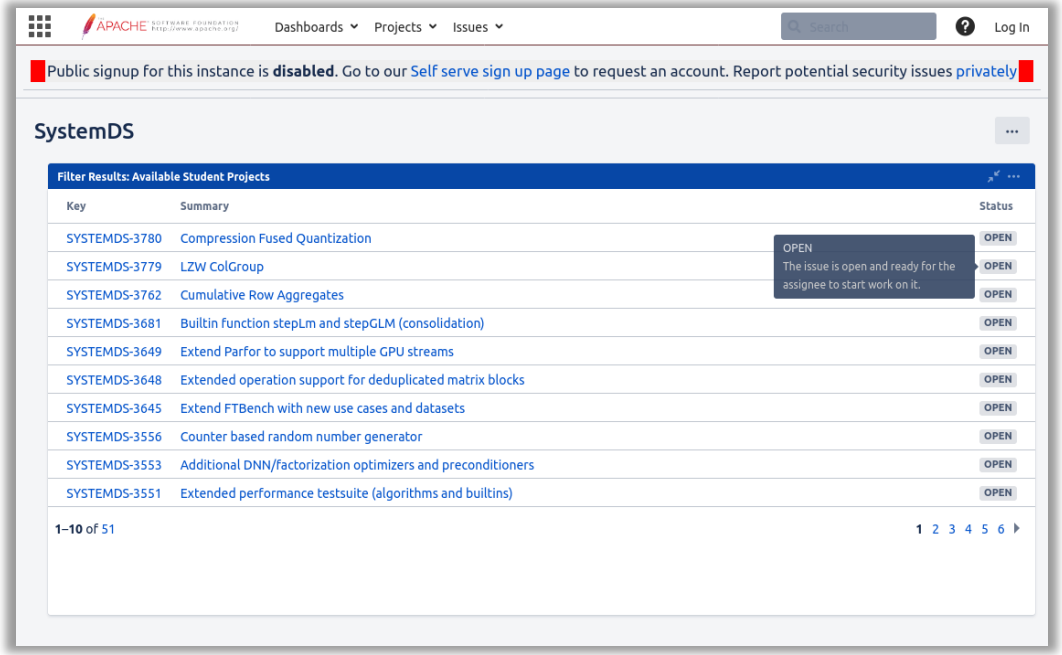
- **Motivation**
  - SIMD (Single Instruction Multiple Data) instructions can process multiple data elements at once and promise good speed-ups compared to ordinary scalar instructions
  - Since recently, SIMD instructions can also be used in Java through the new Vector API
- **Task**
  - Explore the benefit of the Vector API in SystemDS for
    - Dense Matrix Multiplications
    - Quaternary Operations
- **More Information & Hints**
  - https://issues.apache.org/jira/browse/SYSTEMDS-3855
  - https://issues.apache.org/jira/browse/SYSTEMDS-3856
  - Contact: Matthias Boehm

#3865, #3866 & #3867: Frame Operations

- **#3865 Ragged Array Metadata**
  - Integrate the Ragged Array for metadata handling of transform encode metadata.
  - We currently have a partially tested new `RaggedArray`.
- **#3866 Frame Parallel Binary Operations**
  - Move the binary operations from the `FrameBlock` into a lib and add support to push down the binary operations directly to the underlying arrays with cache conscious parallel processing.
- **#3867 Safe Fast Frame Copy**
  - We have added safety synchronization on all copies of frames to avoid race conditions in boolean arrays.
  - Improve the copy logic, to avoid synchronization barriers in generic cases, to improve the copy mechanic.
  - As part of this task, it is expected that we need to verify calling instances to this copy are tested as well.
- **More Information & Hints**
  - https://issues.apache.org/jira/browse/SYSTEMDS-3865
  - https://issues.apache.org/jira/browse/SYSTEMDS-3866
  - https://issues.apache.org/jira/browse/SYSTEMDS-3867
  - Contact: Sebastian Baunsgaard

Topics on Apache SystemDS

- **See the Full List of Available Student Projects:**
  - https://issues.apache.org/jira/secure/Dashboard.jspa?selectPageId=12335852#Filter-Results/12365413



- **More Examples**
  - #3854 Improved HDF5 I/O support
  - #3863 New robust scaling built-in function
  - …
- **More Information & Hints**
  - Contact: Matthias Boehm

Topics in TerseTS

Implementation mainly in Zig
https://github.com/cmcuza/TerseTS

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 using polynomial approximations.
  - Implement the following algorithms as well as lossless compression methods for comparison:
    - Optimal Piecewise Linear Approximation (https://doi.org/10.1109/TSP.2006.875394)
      Finds the best way to break time series data into straight-line segments.
    - Facebook's Gorilla Lossless Compression (https://www.vldb.org/pvldb/vol8/p1816-teller.pdf)
      A method for compressing time series data without losing any information.
    - Mix-Piece Linear Approximation (https://link.springer.com/article/10.1007/s00778-024-00862-z)
      Combines multiple linear segments for a more accurate data approximation.
  - Test and compare them using a few time series datasets.
  - Measure how well each method compresses the data and how much error is introduced.
- **More Information & Hints**
  - Many algorithms are already implemented in TerseTS (a library for time series compression) which we will reuse.
  - Depending on the team size, other algorithms can be added for implementation.
  - Contact: Carlos E. Muniz Cuza

Exploring Optimization Strategies for Storing Timestamps of Irregular and Lossy Compressed Time Series

- **Motivation**
  - Efficient storage of time series data is a fundamental problem in systems dealing with large-scale temporal data, especially when dealing with irregular sampling or lossy compression schemes.
- **Task** (in Zig)
  - Systematically explore, implement, and benchmark various timestamp encoding techniques in scenarios where time intervals are non-uniform or the result of a lossy compression technique.
  - We will investigate a range of encoding methods including:
    - 1. Delta Encoding: storing differences between successive timestamps.
    - 2. Delta-of-Delta Encoding: further compressing delta-encoded values, particularly useful when second-order differences are small.
    - 3. Run-Length Encoding (RLE): effective when many timestamps are repeated or have consistent intervals.
    - 4. Byte Packing and Bit Packing: fixed-width representations optimized for modern CPU.
    - 5. Elias–Fano Encoding: a succinct data structure suitable for monotonic sequences, offering fast random access and space efficiency.
    - 6. Variable-byte and Rice/Golomb Encoding: for more entropy-based compression depending on value distributions.
  - These methods will be evaluated not only on compression ratio, but also on encoding/decoding speed, random access support, and integration with lossy compression techniques. A central focus will be on how different timestamp characteristics (e.g., irregular gaps) affect the performance of each method. Ultimately, the goal of the project is to derive practical guidelines for selecting timestamp encoding strategies and implement these encoding techniques as part of the open-source library TerseTS.
- **More Information & Hints**
  - 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 May 02, 23:59 CEST**;
    in any case, **also fill in the poll regarding the topic selection with your preferred topics from the list above**