YeSQL: Extending SQL with Rich and Performant User-Defined Functions

Importance of integrating user-defined functions (UDFs) with SQL in relational databases

Modern data processing involves handling diverse data sources and complex processing tasks on large volumes of data. Relational databases have emerged as powerful engines for data storage and processing. However, many tasks cannot be efficiently expressed in SQL alone and require additional expressive power.

Overview of the paper's focus on "YeSQL" and its comprehensive study

Existing solutions, such as MonetDB and PostgreSQL, lack certain functionalities like dynamic typing, polymorphism, and Just-In-Time (JIT) compilation. This paper introduces YeSQL, an SQL extension and its implementation that enhances the usability, expressiveness, and performance of Python UDFs.

Related Works

Prior works on optimizing Python code such as compilers and transpilers, and data processing systems with the support of Python UDFs.

- Compiler and Transpiler Approaches
- Data Processing Systems

Compiler and Transpiler Approaches

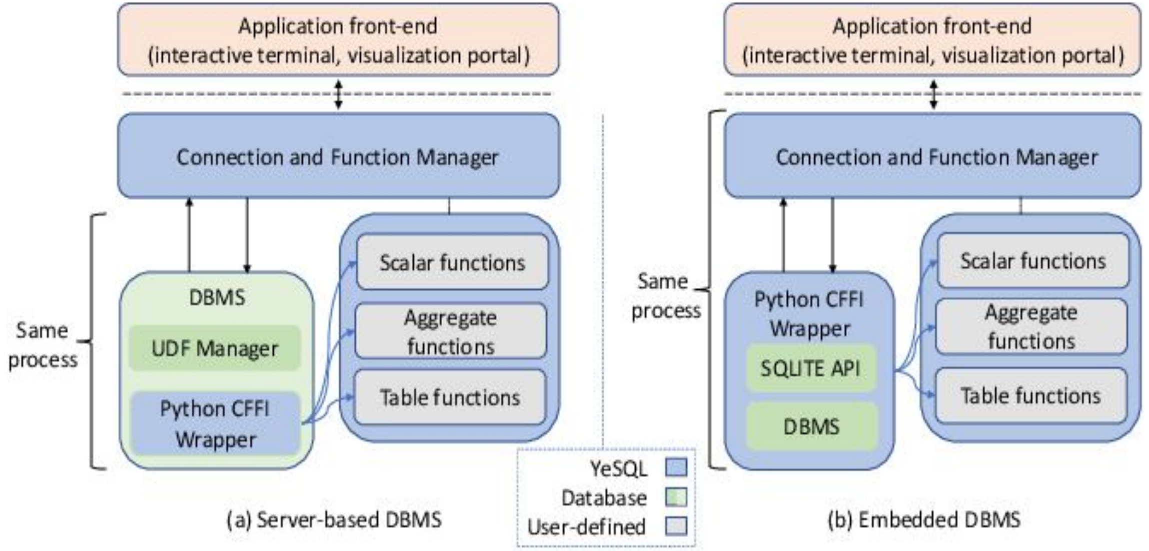
- GraalPython (Focuses on optimizing pure Python code without C extensions, particularly when integrated with Java.)
- Numba (Applies JIT compilation to Python code by inferring data types and targeting array-structured data from libraries like NumPy and it lacks support for dynamically typed Python and importing external libraries.)
- Cython (Translates Python code into C but requires explicit declaration of data types)
- Nuitka (Compiles Python code to C++, but its slow compilation time for UDFs)

Data Processing Systems

- JIT Compilation for Python UDFs (Tuplex:not support exception handling, limiting its practicality in real-world applications.)
- Compilation Frameworks and Optimization Techniques for Python UDFs
- Translation of Python Code into SQL
- Abstractions for Python UDFs in Data Processing Systems
- Translation of UDFs into SQL and Optimization Techniques
- UDF Fusion Techniques But:However, all these techniques support only specific libraries (e.g., matlab, numpy) as they require the libraries to be rewritten in the intermediate representation. In contrast, YeSQL fully supports Python constructs and libraries without any limitation or need to translate the code into an intermediate representation.(GOLAP, Weld, HorsePower)

YeSQL System Architecture:

YeSQL is a novel approach to extending SQL with rich and highly performant user-defined functions (UDFs) in relational databases. YeSQL is a versatile solution that can be integrated with server-based DBMSs, such as MonetDB. Integration with Server-Based DBMSs and Integration with Embedded DBMSs. CFM layer three essential components: the function manager, parser, and code generator.



YeSQL's Advantages

- **Usability and expressiveness:** YeSQL extends SQL with an alternative syntax that enables compact expressions of relational queries and facilitates the composition of complex UDF pipelines. This reduces the time required to develop new algorithms and pipelines.
- **Performance:** YeSQL improves Python UDF performance by addressing the main bottlenecks associated with data conversions, copies, and running complex analysis on the Python interpreter. It achieves this through techniques such as seamless data exchange, JIT compilation, UDF parallelization, stateful UDFs, and UDF fusion.
- **Portability:** YeSQL is designed to work synergistically with existing systems. It is implemented on top of the SQLITE API and is compatible with embedded DBMSs like BerkeleyDB, DuckDB, and MonetDB. It also integrates with other works, such as Hustle, to enhance query acceleration for analytics.
- **Deployment:** YeSQL has been successfully deployed in production by OpenAIRE, a technical infrastructure utilized by numerous European universities and research centers. OpenAIRE data scientists rely on YeSQL for various tasks, including research output harvesting, classification, text mining, and extraction of valuable information. This demonstrates YeSQL's practicality and scalability in real-world scenarios.

YeSQL's Advantages:Rich support for polymorphic Python UDFs

- **Scalar functions:**Programmable scalar functions work in a similar way as standard SQL scalar functions such as *abs()*, *lower()* and *upper()*. Their implementation is done in Python, and they are able to use all Python facilities and libraries.
- **Polymorphic table functions:** Polymorphic tables are actually functions that take parameters and output table like data. They can be used in the SQL syntax wherever a regular table would be placed.
- **Aggregate functions:** Programmable aggregate functions work in a similar way as standard SQL aggregate functions such as *sum()*, *min()* and *max()*. Their implementation is done in Python, and as is also the case for scalar functions, all Python facilities and libraries are available.

YeSQL's Advantages:Rich support for polymorphic Python UDFs

```
yesql> select detectlang('Il en est des livres comme du feu de nos foyers');
french
```

```
% python yesql.py
yesql> select * from file('./demo/continents.tsv') limit 2;
Asia|AF
Europe|AL
```

```
yesql> select "term1+term2" as a UNION select "term2 term3" as a;
term1+term2
term2 term3
```

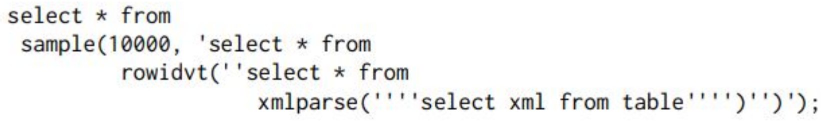
YeSQL's Advantages:UDF fusion for optimized execution

YeSQL enables the direct execution of multiple UDFs on their outputs, creating a fused function at runtime to minimize context switching and data conversion, especially with the use of a tracing JIT like PyPy.

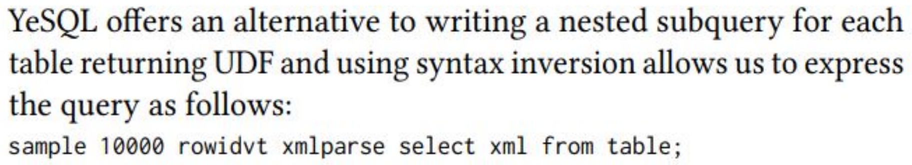
YeSQL's Advantages:Syntax inversion for convenient UDF composition:

YeSQL supports table function chaining, which is known as syntactic inversion in this paper.YeSQL supports a functional-style syntax inversion for composing UDFs conveniently, allowing for the seamless use and combination of UDFs in queries.

Example: The query fragment "*sample 100 file 'publications.json*'" reads a file containing publications and retrieves a random sample of 100 rows.



```
select * from
 sample(10000, 'select * from
        rowidvt(''select * from
                 xmlparse(''''select xml from table'''')'')');
```



YeSQL offers an alternative to writing a nested subquery for each table returning UDF and using syntax inversion allows us to express the query as follows:

```
sample 10000 rowidvt xmlparse select xml from table;
```

Boosting Performance in YeSQL

The major purpose of YeSQL speed improvements is to correct the imbalance of impedance between relational (SQL) evaluation and procedural (Python) execution. To eliminate these costs, used the following boosting approaches one at a time:

- Tracing Just-In-Time (JIT) Compilation: PyPy
- UDF Fusion
- Parallelism
- Stateful UDF
- Seamless Integration with DBMS

Boosting Performance in YeSQL: Tracing Just-In-Time PyPy compiler

JIT compilation enhances programme efficiency by dynamically compiling programme parts into machine code during runtime. Tracing JIT, as opposed to method-based JIT, optimises frequently executed loops, making it suitable for UDFs involving repetitive calculations on table tuples.

The authors' solution made use of PyPy, a dynamic Python compiler. YeSQL query compilation acts as a pre-optimization process, leveraging PyPy's ability to optimise larger Python code segments. PyPy's trace compiler creates optimised machine code based on frequently executed instruction sequences, allowing for better Python performance efficiency.

Boosting Performance in YeSQL: Tracing Just-In-Time PyPy compiler

- (a) Support for the standard Python library and popular packages, with new packages added on a regular basis.
- (b) Optimisation of dynamically typed Python programmes and packages without previous compilation or tweaking.
- (c) Exception handling capabilities akin to CPython, providing smooth exception handling within YeSQL UDFs.
- (d) A foreign function interface, or CFFI, allows for efficient interaction with C code. It also supports CPython, allowing UDFs that require packages not yet supported by PyPy to use a CPython interpreter, and PyPy's interface with DBMS is easy.

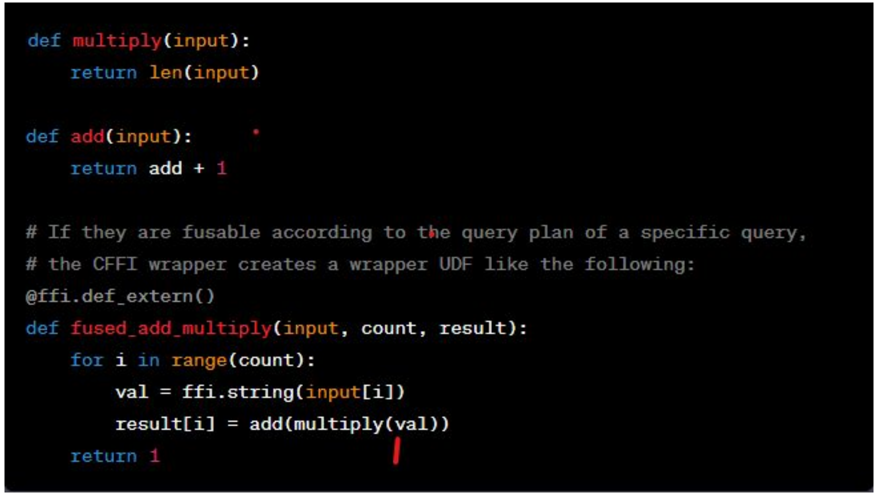
Boosting Performance in YeSQL:Seamless Integration with DBMS

YeSQL provides a seamless interface with DBMS:Integrated CFFI is used to package UDFs in this manner. Throughout the execution of a UDF, data is given to CFFI as pointers to cdata objects, with no data copies.

CFFI - C Foreign Function Interface for Python. Interact with almost any C code from Python, based on C-like declarations that you can often copy-paste from header files or documentation.

Boosting Performance in YeSQL: UDF Fusion

Producing tiny, reusable UDFs boosts efficiency, particularly in text mining operations. By reducing conversions and exposing larger instruction sequences, UDF fusion enhances speed. When two UDFs are fusible, a new merged UDF is constructed for efficient pipeline execution utilising a CFFI wrapper.



```
def multiply(input):
    return len(input)

def add(input):
    return add + 1

# If they are fusable according to the query plan of a specific query,
# the CFFI wrapper creates a wrapper UDF like the following:
@ffi.def_extern()
def fused_add_multiply(input, count, result):
    for i in range(count):
        val = ffi.string(input[i])
        result[i] = add(multiply(val))
    return 1
```

Boosting Performance in YeSQL: Parallelism

In general, parallelism improves programme speed and scalability. When Python UDFs are used in parallel execution within DBMSs, their speed is limited by the **Global Interpreter Lock (GIL)**. Because the GIL only permits one thread to control the Python interpreter at a time, CPU-bound and multi-threaded programming suffers significantly. The GIL is also active during Python object creation, which adds extra cost. Python object creation is quicker in PyPy, and optimised GIL releasing and acquiring are enabled, providing better performance than standard Python. GIL is released slowly and without synchronisation in CFFI, increasing efficiency .

Boosting Performance in YeSQL: Stateful UDF

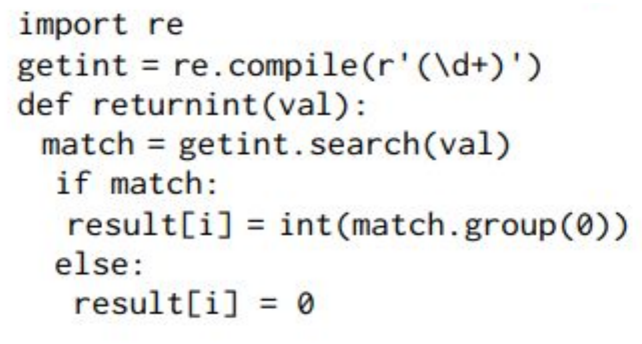
A stateful UDF refers to a user-defined function that can maintain and utilize internal state or context across multiple invocations. This means that the UDF can retain information or data between function calls, allowing it to make use of that stored state in subsequent invocations. Another example is precompiling a pattern instead of compiling it once per each row.

Most data processing systems that accept Python UDFs primarily provide stateless UDFs, in which only the output remains after the UDF has been executed. Stateful UDFs, on the other hand, provide up prospects for algorithm creation in data analysis and data science, as well as specialised performance optimisations. UDFs are often stateful by default in embedded DBMSs and can access external states. States are available across rows and UDFs in server-based DBMSs such as MonetDB. The UDF developer distributes their UDF as a Python module, allowing for global configuration and package import. The **Function Manager** wraps these functions in a CFFI embedded function, which provides performance benefits such as conducting expensive actions once at the global scope, such as downloading nltk data or assigning precompiled regex patterns.

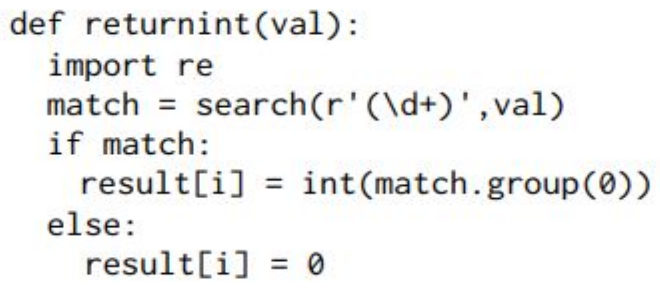
Boosting Performance in YeSQL: Stateful UDF

Stateful UDFs can provide performance benefits by avoiding redundant computations or initialization steps, as they can reuse previously computed results or data.

For example, the following UDFs are equivalent but the left one imports the package and pre-compiles the pattern at global scope:



```
import re
getint = re.compile(r'(\d+)')
def returnint(val):
  match = getint.search(val)
   if match:
    result[i] = int(match.group(0))
   else:
    result[i] = 0
```



```
def returnint(val):
  import re
  match = search(r'(\d+)',val)
  if match:
    result[i] = int(match.group(0))
  else:
    result[i] = 0
```

Evaluation

The YeSQL codebase is made up of around 66K lines of Python and C++. This comprises 18.5K lines devoted to the code definitions of over 150 Python UDFs supported. To examine YeSQL, used three sample data science workflows and run micro-benchmarks targeting key design characteristics.

Data science workflows:

- Zillow consists of listings from the Boston, MA area, with variations in size: 1GB/5.6M rows, 5GB/28.6M rows, and 10GB/56M rows.
- Flights originally from a Trifacta tutorial, was expanded by the Tuplex team through the inclusion of additional airport and airline data from various sources . It is available in three size variations: 1.6GB/5M rows, 3.2GB/10M rows, and 6.4GB/20M rows.
- The text-mining pipeline is derived from a practical application called OpenAIRE.

Results of pipeline: Zillow

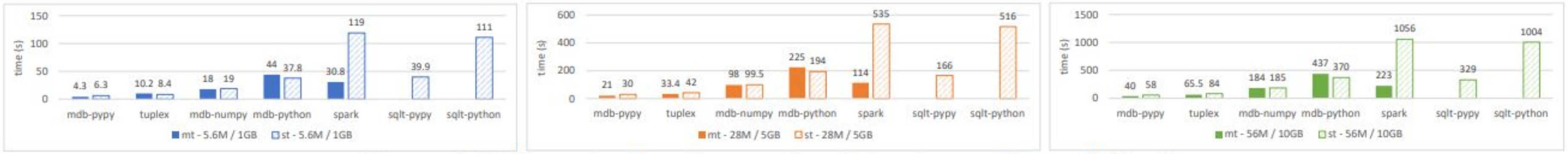


Figure 4: Zillow pipeline for varying data sizes and parallelization

- mdb-pypy -> YeSQL with tracing JIT on MonetDB
- mdb-python -> MonetDB with CPython with GIL

Results of pipeline: Flights

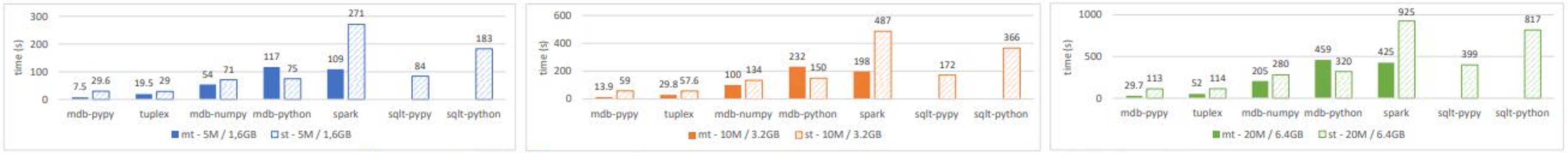


Figure 5: Flights pipeline for varying data sizes and parallelization

- mdb-pypy -> YeSQL with tracing JIT on MonetDB
- mdb-python -> MonetDB with CPython

Results of pipeline: Text mining

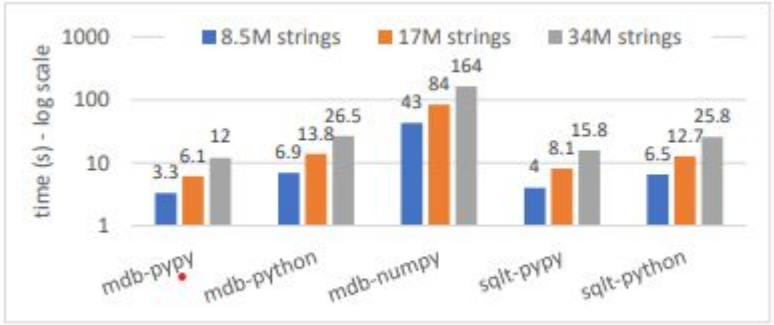


Figure 6: Text mining pipeline (time in log scale)

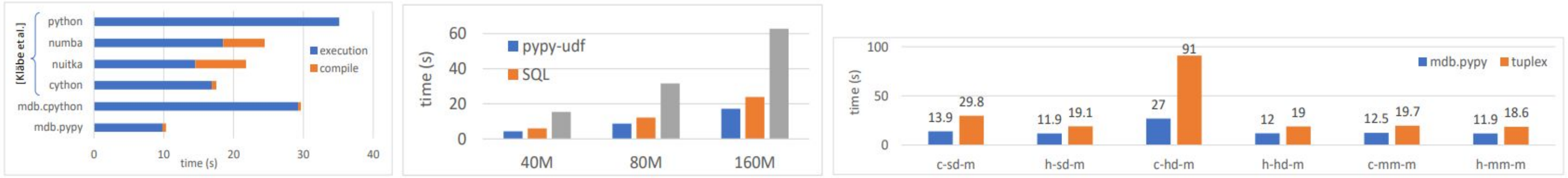
- mdb-pypy -> YeSQL with tracing JIT on MonetDB
- mdb-python -> MonetDB with CPython

Micro-benchmarks: Evaluating the performance of a single UDF, seamless integration, and various setups

- Tracing JIT: The effect of tracing JIT (specifically PyPy) on UDF execution in YeSQL is investigated and compared with other systems.

- Seamless integration with a DBMS:Experiment 1: A comparison is made between a SQL query and its Python UDF counterpart. The UDF execution using tracing JIT, CFFI, and seamless integration in MonetDB achieves similar or better performance compared to native SQL queries. Numpy UDFs are slower due to string-to-PyObject conversions.Experiment 2: The overhead of transferring string columns using ffi.string is evaluated. The first UDF, which directly processes data objects in C, is faster on average by 1.9x compared to the second UDF, which converts data to Python strings.Experiment 3: Different setups (cold/hot caches, SSD/HDD/shared memory, parallelization) are analyzed. YeSQL outperforms tuplex in multi-threaded execution across various setups, with performance factors ranging from 1.56x to 3.37x. Tuplex performance varies based on the execution setup, achieving the best results with hot caches.Experiment 4: A comparison is conducted on different DBMSs supporting Python UDFs. YeSQL's extensions to MonetDB with either CPython or PyPy outperform alternatives like PostgreSQL, Spark, dbX, Pandas, Tuplex, and NumPy. YeSQL on SQLite remains competitive, highlighting its versatility for lightweight architectures and various applications.

Micro-benchmarks: Evaluating the performance of a single UDF, seamless integration, and various setups



(a) Comparison of a single UDF (b) Seamless integration (c) Cold cache vs. Hot cache on SDD/HDD/tmpfs (multi-threaded)

Figure 7: Evaluating the performance of a single UDF, seamless integration, and various setups

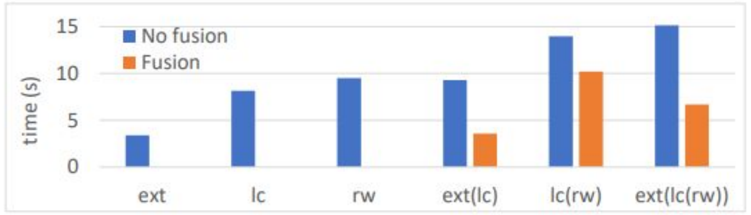


Figure 8: UDF fusion

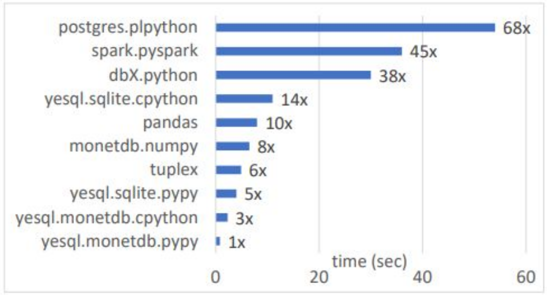
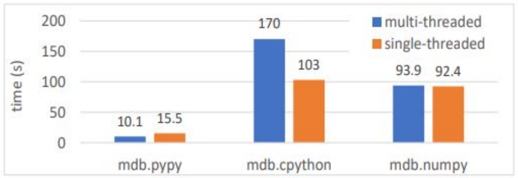


Figure 1: Performance of a scalar UDF on a string column

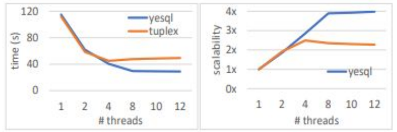
Micro-benchmarks: Evaluating UDF fusion, parallelism, and resource utilization

- UDF fusion: UDF fusion was evaluated in an experiment using 10 million text snippets from a text mining dataset.The query involved multiple UDFs, including removing small words, converting to lowercase, and extracting integer numbers using regular expressions. The execution times of the query with and without UDF fusion were compared.
- Parallelism:In an experiment, the execution times of 9 UDFs from the flights pipeline were tested using PyPy and CPython UDF implementations, as well as MonetDB's default Numpy UDFs. The UDFs were executed in both single-threaded and multi-threaded modes.
- Resource usage:Resource usage of YeSQL with PyPy, Tuplex, and PySpark was profiled, monitoring metrics such as CPU, memory, and disk utilization. The experiment utilized the flight dataset (10M rows) with multi-threaded execution and cold/hot caches.

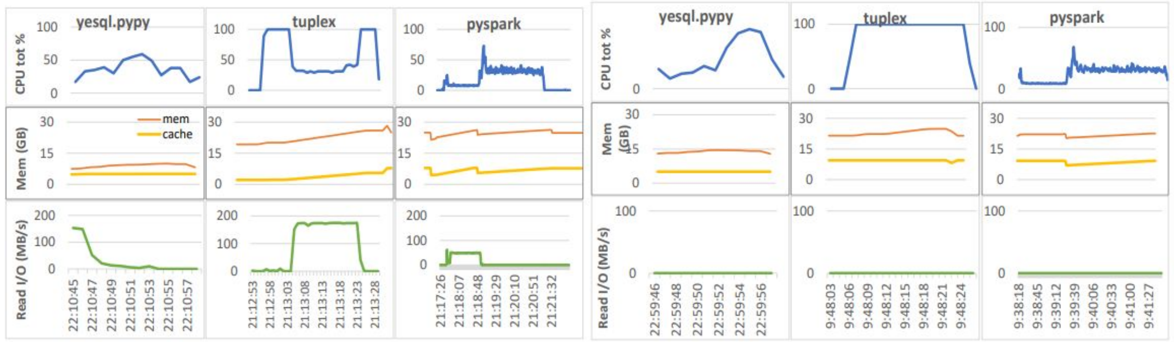
Micro-benchmarks:Evaluating UDF fusion, parallelism, and resource utilization



(a) Multi-threaded vs. single-threaded



(b) Scaling degree of parallelism



(c) Resource usage: YeSQL, Tuplex, PySpark with cold and hot caches

Figure 9: Evaluating UDF fusion, parallelism, and resource utilization

Conclusion

In conclusion, YeSQL offers a powerful solution for extending SQL with rich and highly performant UDFs, providing users with advanced functionality and improved performance within the familiar SQL environment. Its contributions to the field of SQL UDF integration, along with its practical implications and future research opportunities, make it a valuable asset for researchers, practitioners, and database professionals.

Thank you for your time and attention 🙂