Bring Your Language to Your Data with EXASOL

Stefan Mandl[1], Oleksandr Kozachuk[2], Jens Graupmann[3]

Abstract: User Defined Functions (UDF) are an important feature of analytical SQL as they allow processing of data right inside of relational queries. Typically UDFs have to be written in a special language which sometimes diminishes their practical use, especially when required libraries are not available in this language. An alternative approach is to allow users to provide functions as native low-level database extensions; an approach that can be very dangerous. EXASOL now follows a radically different approach by allowing to integrate any programming language with the database without affecting data integrity: UDF language implementations are encapsulated within Linux containers that communicate with the database engine via a straightforward protocol. Using these technologies, users can now make available their own programming language for UDFs in the database. As an example, we show how to provide C++ as a UDF language for EXASOL.

Keywords: SQL, User Defined Function, C++

1 Introduction

User Defined Functions (UDFs) are an important tool for users to extend the standard set of SQL functions. Especially for analytical scenarios, the ability to create custom aggregate functions, analytical functions or table generating functions is crucial.

Furthermore, for parallel and distributed systems with massive amounts of data, copying all data to a single machine and processing it there with a single process is often not an option. Here UDFs that run in parallel on many machines provide the necessary solution, but only if the UDF language is suitable for the task and if the right libraries are available in the database.

With EXASOL (starting in version 6.0.0) it is now possible for users or other 3rd party to add new UDF language implementations, new libraries for existing languages, or complete standalone language containers.

In this paper, we explain the basic concepts behind the new technology and demonstrate how to add C++ as a new UDF language to EXASOL.

2 Concepts

Please note, that in EXASOL, UDFs are usually run in parallel and distributed in a cluster. This means, that for a UDF in a given language, there are typically many instances of the

[1] EXASOL AG, Neumeyerstr. 22–26, 90411 Nuremberg, Germany, stefan.mandl@exasol.com
[2] EXASOL AG, Neumeyerstr. 22–26, 90411 Nuremberg, Germany, oleksandr.kozachuk@exasol.com
[3] EXASOL AG, Neumeyerstr. 22–26, 90411 Nuremberg, Germany, jens.graupmann@exasol.com

	scalar output	tuple output
scalar input	scalar functions (SCALAR-RETURNS)	table generating functions (SCALAR-EMITS)
tuple input	aggregate functions (SET-RETURNS)	general $n : m$ mapping (SET-EMITS)

Tab. 1: Different UDF input-output behaviors in EXASOL. The keywords in parentheses are used as syntax in the CREATE SCRIPT statement of EXASOL.

language implementation running at the same time (More information can be found in the white papers [EX15] and [EX]). UDFs can be invoked in two modes:

- Callback mode: Only a single function in the UDF will be called by the database. This mode is used to implement features like virtual schemas, where the SQL compiler needs to communicate with user defined code.
- Compute mode: The UDF is started in order to process data.

In compute mode, data, which is sent from EXASOL to the UDF and results which are sent back from the UDF to EXASOL can either be scalar or tuple valued, resulting in four different kinds of user defined functions which are shown in Tab. 1.

In this section we describe the individual ingredients which together enable users to add new languages to EXASOL:

- Loose coupling of languages to EXASOL gives us a lever where new languages can be added.
- The public script language implementation protocol allows 3rd party developers to create new languages that behave in the same way as the pre-installed language implementations.
- BucketFS allows to actually install the new language on every machine inside an EXASOL cluster.
- A URL like syntax in SQL for describing the details of how to access a script language implementation to enable EXASOL to actually use it.

2.1 Tight vs. loose coupling of UDF languages

In EXASOL language implementations can be one of two different kinds:

- Tightly coupled: The Lua language and its implementation is specifically designed as extension language. Its interpreter is simple and safe and can be integrated with existing C++ code without compromising other parts of the system. Therefore, in order to avoid copying data to and from the script language implementation, Lua is compiled into the database engine.

- Loosely coupled: Implementations of languages like Java, Python, and R are not suited for direct linking into a database engine as they provide means for low-level memory access, tend to clog memory or crash often. These language implementations simply are not compatible with the safety and security requirements of a database management system. Therefore UDF scripts using these languages are started in a separate operating system process which in turn is encapsulated into a separate Linux container. The language implementation and the database system communicate via ZeroMQ[4] sockets.

2.2 Script Language Protocol

The messages between EXASOL and an UDF script language implementation are encoded using Google's Protocol Buffers[5]. The message types are listed in Tab. 2. After being started by the database, the language implementation takes initiative, it sends a message of type MT_CLIENT to the database, which responds with a message of type MT_INFO which among general information about the EXASOL system and the UDF (like database name, database, the number of cluster nodes, etc.) also contains the actual source code of the UDF. Then the script language implementation requests the meta data of the current UDF: types and names of input and output columns, the call mode and the required input and output iteration behaviors.

Finally, the script language implementation notifies the database system that it is up and running and will be requesting and sending actual data using the MT_RUN message.

Then the UDF and EXASOL iteratively interchange data. Here, the UDF requests new data with MT_NEXT messages, and sends results with MT_EMIT messages.

Please note that in an EXASOL cluster a single instance of a UDF typically only processes a small subset of the data. The only exceptions being UDFs with input-output behavior SET-RETURNS and SET-EMITS. For these kinds of UDFs, a single instance will receive the complete data of a group as defined by the GROUP BY clause (if present) of the query it is used in. This allows for implementing aggregate functions by the user. Processing still is in parallel, but the parallelism is follows the grouping of the data.

2.3 BucketFS

BucketFS is a replicated file system in EXASOL clusters which is accessed via HTTP. Files can be stored in buckets which contain no further hierarchy. More than one BucketsFS can be running in an EXASOL cluster. From the point of view of EXASOL's SQL engine, BucketFS is just another external system with one exception: Inside of script language implementations and therefore in UDFs, files in BucketFS can be read via the file system. They are mounted using directory names of the form `/buckets/<bucketfs name>/<bucket name>` As a

[4] `http://zeromq.org`
[5] `https://github.com/google/protobuf`

MT_CLIENT	The script language implementation is alive and requests more information
MT_INFO	Basic information about the EXASOL system and cluster configuration and the UDF script code
MT_META	Names and data types of the data to send between EXASOL and the script language implementation
MT_CLOSE	Terminates the connection to EXASOL
MT_IMPORT	Request the source code of other scripts or information stored in CONNECTION objects
MT_NEXT	Request more data to be sent from EXASOL to the UDF
MT_RESET	Restart the input data iterator to the beginning
MT_EMIT	Send results from the UDF to EXASOL
MT_RUN	Change status to indicate the start of data transfers
MT_DONE	Indicate that the UDF will send no more results for the current group of data
MT_CLEANUP	Send to indicate that no more groups of data will have to be processed by the script language implementation and that it may stop
MT_FINISHED	Sent when the script language implementation successfully stopped
MT_CALL	Used to call a certain function in the UDF when in Single-Call mode
MT_RETURN	Used to send the result of the Single-Call function call

Tab. 2: Message types related to loosely coupled UDF implementations in EXASOL.

matter of fact, the Linux container which is used to run the pre-installed script languages as well as the actual implementation of the pre-installed script languages is also installed in BucketFS.

In a nutshell, this means that languages that are created and installed by 3rd party can use the exact same technology like the pre-installed languages.

2.4 Defining and Using Script Language Implementations in SQL

In order to start and communicate with a script language implementation, EXASOL basically needs two things:

1. A Linux container in which to start the language implementation
2. The actual binary to run inside the container.

These information are customizable via a session/system parameter. The syntax is like this:

```
ALTER SESSION SET SCRIPT_LANGUAGES=
                  '<LANGUAGE_NAME>=<LANGUAGE_DEFINITION> ...';
```

Language names are simple strings while language definitions are either one of the magic names `builtin_python`, `builtin_java`, `builtin_r` or URL syntax. The magic names in turn are interpreted by the EXASOL compiler so that they expand to URL syntax which point to the script languages that come pre-installed with EXASOL. This way, users can be sure that their Python script always uses the Python that comes with the current version of EXASOL, while still being able to modify which implementations to use for other languages.

The URL syntax for script language implementations has the standard form:

```
scheme:[//host[:port]][/]path[?query][#fragment]
```

The parts are interpreted in the following way:

- **scheme** - *localzmq+protobuf* - this means that EXASOL and the script language implementation communicate via ZeroMQ and encode their message with Protocol Buffers. In the future, other schemes could be used.
- **host:port** - currently, this value is empty as script language containers are started on the same machines as EXASOL runs on.
- **path** - this component has the form

  ```
  <bucketfs_name>/<bucket_name>/...<path_to_linux_container>
  ```

  It describes how to access the Linux container which will be started to run the UDF script language implementation in.
- **query** - The value of query is passed as parameter to the actual script language implementation. For the built-in languages, we only have a single binary, which can start all the built-in languages and uses the value of query to determine which language to use.
- **fragment** - this is the path to the binary which implements the language inside the Linux container which is defined in the path component.

For EXASOL 6.0, the pre-installed script language `PYTHON` has the following URL:

```
localzmq+protobuf:///bfsdefault/default/EXAClusterOS/
ScriptLanguages-6.0.0/?lang=python#
buckets/bfsdefault/default/EXASolution-6.0.0/exaudfclient
```

(without newlines).

Once the script language is defined via the variable SCRIPT_LANGUAGES, it can be used when defining new UDFs.

In addition, the language definition can be changed during SQL sessions and UDFs will then use the updated languages.

3 Use Case: Providing C++ as a UDF script language

Up until version 6.0, EXASOL only supported the UDF languages Python, Java, R, and Lua. These are also the languages that come pre-installed with EXASOL 6.0

As an example for the new technology, we show how to add C++ as a UDF language.

3.1 Which Linux container to use?

For new language implementations, user are free to upload new Linux containers into BucketFS. As an alternative they can also use the container which is already provided by EXASOL, which is a good default strategy as it saves disk space.

On the other hand, if your language requires many additional libraries in special versions, a additional container is a good idea.

For implementing C++, we suggest using the container that is already installed in EXASOL. Then only the binary implementing the UDF script language protocol needs to be created by the user.

3.2 Basic strategy for executing C++ in UDFs

As described above, the contents of UDF scripts is sent as source code to the UDF language implementation. Therefore, in order to implement C++, we suggest the following strategy:

- Compile the UDF's code on the fly (using `gcc`).
- Dynamically link and call the generated code.

This strategy has the advantage that it allows to program UDFs in C++ directly inside the SQL client. The disadvantage is that it compiles the C++ code for every query, which in particular for C++ can be very costly.

An alternative strategy is to pre-compile the UDF's C++ code, load it into BucketFS to link and call it from the UDF without invoking the C++ compiler. Then the actual code in the UDF would only specify which libraries to link and which functions to call. As we feel that this is a different (but also interesting) language altogether, we stick with the first alternative.

3.3 Creating the EXASOL Protocol and the Language Implementation

Here comes the hardest part when providing a new language, as the protocol which has been outlined above has to be implemented for every new script language. On the other hand, we provide the source code of the C++ language implementation described here on `https://github.com/EXASOL/script-languages` which should provide a good starting point for anyone creating new language implementations.

When creating the language implementation, it is important to use the Linux container that later will be used when running the implementation.

Here docker[6] provides a nice tool chain to simplify the process:

- Linux containers can directly be imported from BucketFS and started with shared folders.
- Inside the shared folder, the language implementation can be compiled.
- The content of the shared folder can be packaged and uploaded into BucketFS.

3.4 Creating and Using C++ UDFs in SQL

Once the UDF language implementation is created and loaded into BucketFS, it can be added to the UDF languages known to your EXASOL by adding a definition like the following to the SCRIPT_LANGUAGES variable:

```
CPP=localzmq+protobuf:///bfsdefault/default/EXAClusterOS/
ScriptLanguages-6.0.0#buckets/bfsdefault/cpp/cppclient/cppclient
```

Here we assumed, that `cpplient` implements the UDF language and is stored in a bucket with the name `bfsdefault` and accessible via the path `buckets/bfsdefault/cpp/cppclient/cppclient`.

Now new UDF scripts, using C++, can be created by the user like this:

```
CREATE cpp SCALAR SCRIPT duplicateAndScale(n INT, x DOUBLE,
                                           y DOUBLE, z DOUBLE)
EMITS (x DOUBLE, y DOUBLE, z DOUBLE) AS

%compilerflags -lblas;
#include <cblas.h>

using namespace UDFClient;
```

[6] `https://www.docker.com`

```
void run_cpp(Metadata* meta, InputTable* in, OutputTable* out) {
  for (size_t n = 0; n < in->getInt64(0); ++n) {
    double x[] = {in->getDouble(1), in->getDouble(2),
                                    in->getDouble(3)};
    cblas_dscal(3, 4.323, x, 1);
    for (size_t i = 0; i < 3; ++i)
        out->setDouble(i,x[i]);
    out->next();
  }
}
```

Please note that we added the `%compilerflags` directive to our version of C++ UDFs which can be used for several purposes like linking additional libraries to the C++ code of the UDF. The classes `Metadata`, `InputTable`, `OutputTable` and the convention to use the function name `run_cpp` for the main loop of our UDF are all features of our particular implementation of C++ UDFs and are not related to EXASOL's UDF protocol.

Finally we are able to seamlessly use C++ UDFs in SQL:

```
SQL_EXA> SELECT duplicateAndScale(3,1,2,3);
EXA: SELECT duplicateAndScale(3,1,2,3);

X                 Y                 Z
----------------- ----------------- -----------------
            4.323 8.646000000000001            12.969
            4.323 8.646000000000001            12.969
            4.323 8.646000000000001            12.969

3 rows in resultset.
```

4 Conclusion

User Defined Functions in EXASOL are a powerful feature that allows processing massive amounts of data in parallel and distributed in a cluster.

They can be used to create map-reduce style dataflows, create new aggregate functions, and support many other use cases.

Except for the language Lua, UDF language implementations are not compiled into EXASOL but run in an isolated Linux container (per UDF instance) for maximum safety and security. These implementations communicate with the database via a straightforward protocol.

By making these these internals accessible for users and other 3rd party, anyone can now add new UDF languages to EXASOL by following the three simple steps of creating a new language implementation, making it available in the EXASOL cluster via BucketFS and informing the SQL compiler about the new language.

Using these new facilities, a proof-of-concept implementation of C++ as new UDF language has been straightforward and we eagerly expect new languages to be made available by customers, consultants and data geeks.

References

[EX] A Drill-Down into EXASOL. `http://info.exasol.com/technical-whitepaper-exasol-2.html`.

[EX15] EXASOL – A Peek under the Hood. `http://info.exasol.com/technical-whitepaper-exasol-1.html`.