Abstract: In light of continuously growing volumes of data, increasingly complex calculations must be performed in the database. This becomes even more important for larger data volumes and more specific calculations. Particularly in the big data and in-memory environments, complex statistical calculations on data volumes that far exceed conventional main memory volumes are a matter of course. With its EXAPowerlytics advanced analytics environment, EXASOL has created a tool that allows complex calculations to be performed in the database. Thus EXAPowerlytics unites the advantages of a modern in-memory database with those of conventional programming environments: (1) it is integrated entirely into SQL and therefore complies with standards, (2) it allows development in a whole variety of programming languages, and (3) it enables quick and simple development of massively parallel algorithms.
1 In-memory and big data

By the end of 2013, humankind will have compiled several sextillion \(10^{21}\) bytes of data and information [MC13] - a virtually inconceivable volume of data. This is by no means surprising given the Internet’s exponential growth as an inexhaustible data source and the ever-growing number of data uploads in social media. Yet the amount of data is also rocketing within industry, primarily due to machine-to-machine (M2M) communication and the long-term storage of process and measurement data. In recent years, data pools that were a blessing for companies just a few years ago have become a major challenge. This development is reflected in a number of different factors, which are leading to calls for a rethink in the approach to large data volumes: on the one hand, decision-making processes implemented to date can no longer be used, as the algorithms previously applied are only effective with small data volumes. Today, the vast increase in data slows the outdated algorithms, rendering data analyses at short notice impossible. On the other, demand for the analyses performed also increases with the data volume. Thus, while it was previously sufficient to analyse the turnover from one month or one year for financial reports, for example, current data sets must be analysed and evaluated in real time, particularly in the manufacturing industry. It is no longer sufficient to have a comprehensive evaluation of the previous day by the end of the following one. It must be possible to call up production reports and quality assessments on an ad hoc basis. A third factor influencing the change brought about by growing data volumes is the lack of usability of previous systems. Particularly unstructured data (e.g. sensor measurements in production or social media data) cannot be analysed using the standard approach. Today, data such as images and texts, which must first be converted into a data format that is comprehensible to the analysis database in a complex process, is being used in analyses. If this data is linked to structured data, the outcome is a comprehensive database that often comprises several terabytes of information. It is only with new techniques (e.g. in-memory and massively parallel data processing) that these mountains of data can now be evaluated within a short time.

In-memory databases reflect a development that returns databases to the centre of strategic applications. The ever-growing data volumes mean that it is simply no longer possible to load large parts of a data set from the database into an application for complex calculations. One outcome is the in-memory principle: the database retains the data in main memory and can access it more rapidly as a result. This approach yields a great many advantages for simpler calculations in particular, as the data is available to the application (e.g. a BI tool or decision support system) very quickly, and only a few changes to the existing programs are required. However, for more complex calculations, the in-memory connection alone is not fast enough, as the data must still be transferred to the application via the relatively slow database connection. Particularly detailed statistics and
complex business applications require a so-called data locality, i.e. the calculations must be performed where the data is stored. EXAPowerlytics from EXASOL is also based on this principle.

2 Calculations in the database

EXAPowerlytics allows complex algorithms to be performed in the database and therefore for the advantages of an in-memory database to be linked with those of modern programming languages. This takes place through the integration of the Lua [Lua96], Python [Py13] and R [R13] programming languages (see Section 4 for details). The program code of the respective language can be defined directly in SQL.

```
CREATE R SCALAR SCRIPT greet_world (name VARCHAR(50))
EMITS (greeting VARCHAR(100) AS
    run <- function(ctx) {
        ctx$emit(paste("hello world from ",name))
    }
/
```

And it can also be retrieved there as well.

```
SELECT greet_world ('EXASOL') FROM DUAL;
```

This type of script is known as a user-defined function (UDF). By performing parts of the business logic in the database, it is possible to retrieve data very rapidly, enabling data volumes in the big data environment to be processed.

Four different types of user-defined functions (UDFs) can be defined in EXAPowerlytics:

1. SCALAR RETURNS: a function that is assigned a tuple of scalar values, i.e. integer value, and returns a single integer value.
2. SCALAR EMITS: a function assigned a tuple of scalar values that returns a list of scalar values.
3. SET RETURNS: a function assigned a large number of values that returns one single value.
4. SET EMITS: a function assigned a large number of values that returns a large number of values.

Each of these four types has its applications. SCALAR RETURNS is suitable for calculating a complex KPI value from a number of scalar column values, for example. With SCALAR EMITS, a Fibonacci sequence up to the specified value can be determined. SET RETURNS is useful for aggregation functions analogous to
MIN/MAX or MEDIAN operations in which one element should be determined or calculated from a list of values. SET EMITS can be used for filter or statistical operations in which a probability independent of the total volume is attributed to each element in a volume.

2.1 A production line approach

SQL queries are processed in EXASolution, the database management system from EXASOL, in the form of overlapping work steps. This procedure is known as pipelining [GUW08], as similar to the stages in a production line - the data moves through a series of stations and is processed accordingly. One of these stations is a filter set by the WHERE clause of an SQL query, for example, which only allows the data elements meeting the query through.

![Figure 1: Pipelining in the database. The data is fed through the process one data item at a time. In the process, the work steps are applied to the individual data sets, which makes temporary saving of the entire data set unnecessary.](image)

One station in the production line processing is the execution of a UDF. This means that a function must always only consider a small portion of the entire data set, as appropriate. This is particularly advantageous for very large data volumes, as not the entire data set but only the current one is loaded into main memory. The function itself also processes the data like in a production line: the respective current data item is always read, processed and issued in a loop. This process is repeated until all the data items have been processed.

```lua
CREATE LUA SET SCRIPT pipeline (x DOUBLE) EMITS (x DOUBLE) AS
  function run(data)
    local current_result = 0
    Repeat
```

---

© 2013 EXASOL AG - White paper Advanced analytics
-- perform operation on current data item
   current_result = do_something(data.x)

-- write result of operation back 2 pipeline
   data.emit(current_result)

-- fetch next data item and iterate
   until not data.next()
end

A UDF is therefore able to process several hundred data sets without having to load the entire data set. This is a particularly valuable asset in the big data environment.

3 Massively parallel calculations in the database

Executing operations on a data set is just a small part of the power of EXAPowerlytics. The environment’s special capability is reflected in the parallel execution of scripts on several nodes.

3.1 The structure of EXASolution

EXASolution is conceived as a cluster system, i.e. the database system distributes the data to all nodes in the cluster so that each individual node only has to manage part of the data. When the system receives an SQL query, the system accesses all nodes and distributes the query’s processing across the entire cluster. EXASolution automatically distributes the data in the cluster, meaning that the user does not need to take care of the distribution or access.

Figure 2: Parallel execution of a query in EXASolution. The database distributes the query to all nodes, and each node processes the query independently of the others.
This principle of distributing the processing to the individual nodes is not only appropriate for responding to queries; it is also particularly well suited to the performance of complex calculations. For example, in a simple mean value calculation, the mean value is calculated for the individual subsets independently of and in parallel to one another in the first step, with the mean value for the mean values of the subsets then determined in the second step. This approach of (1) distributing the calculation to a series of computers and (2) merging individual results to obtain an overall result is known as MapReduce (MR) [DG04]. MR is primarily popular due to its simplicity, as in contrast to other concepts for the distributed computing of calculations, which mostly come from scientific computing [KK02] - its uses are extremely versatile.

Thanks to EXAPOWERlytics, the development of distributed algorithms is now even easier. The database takes care of the distribution work, leaving the developer to take care of just the calculation.
This example shows parallel calculation of the mean value using the built-in AVG function in the map step and aggregation of the values with the help of the Lua GROUP_MAX script in the Reduce step. The database distributes the groups arising from the GROUP BY statement to the individual nodes, and calculation of the mean value is then determined in parallel on the respective nodes independently of the mean values of the other groups. Calculation of the maximum of all group mean values ultimately takes place on a node sequentially. This form of parallel execution yields significant speed advantages in complex map steps and simple Reduce steps in particular.

One example for such an application would be the selection of derivatives based on the underlying share price. The table values would be the historic values of the different shares. Each share then corresponds to a group that is then aggregated. A pricing algorithm is used as the map function (countless packages are available for this in R), and in the Reduce function, the derivatives with the most favourable characteristics at the time can be selected. Given that there are a great many shares and calculation of a fair price is very time-consuming, parallelisation leads to an increase in speed, which increases linearly with the number of computers involved.

4 Integration of external programs

The selection of programming languages that can be used to develop UDFs is often decisive to the acceptance of a database solution. EXASolution allows for programs developed in the programming language of your choice to be integrated as UDFs. The programs run as independent processes, with communication with the
database taking place via ØMQ [H13]. Given that ØMQ also enables access to non-local processes, which allows processing on a system with only network-linked database nodes, your imagination needs to have virtually no boundaries. From local Java programs through to access via legacy software running on host systems, almost anything is possible.

![Diagram](image)

**Figure 3:** Linking external processes to EXASolution. With EXASolution, linkage is possible via ØMQ. Thus, it is irrelevant whether the process runs on the local nodes or a remote system.

To define an external UDF (eUDF), it is therefore sufficient to enter the function signature - its name, the associated transfer and return parameters and the network address (IP address and TCP port) where the service can be found.

```sql
CREATE EXTERNAL SCALAR SCRIPT external_script(param DOUBLE)
EMITS (num INT) AS
# redirector tcp://192.168.1.1:4050
/
```

The signature shown above indicates that an external UDF can be found at the network address 192.168.1.1:4050. The corresponding program, shown in section 4.1, of the external process then implements a service that waits for queries at the address indicated and executes them. While implementation of such a service is complex, it is also extremely flexible. Not only can this approach be used to develop UDFs in virtually any programming language, it is also easy to connect to external services such as web services or existing programs. In such cases, the eUDF would merely serve as a wrapper for the actual service.
4.1 Hadoop and the world of unstructured data

One special type of external process is undoubtedly the Hadoop MapReduce job. It enables simple processing of unstructured data using the like-named open source framework, and has therefore gained enormously in popularity in recent years. With the EXAPowerlytics Hadoop Integration Service, Hadoop jobs can be integrated into SQL processing by EXASolution extremely easily. This service expands the eUDF framework with a management service, the redirector service, which manages the assignment of Hadoop nodes to EXASolution nodes and vice versa. Figure 4 shows the structure of the Hadoop integration services. Using an SQL command, a reference is applied to an existing Hadoop job that can then be called up directly from the SQL processing.

```
CREATE EXTERNAL SCALAR SCRIPT external_script(param DOUBLE) 
EMITS (num INT) AS 
  # redirector tcp://192.168.1.1:4050
  exasol.method = input
  exasol.input.splits = 1
  mapred.output.dir = file:/resultdir
  mapreduce.job.jar = /nfs/home/my_user/my_exa_classes.jar
  mapreduce.job.name = hd_job
  mapreduce.job.map.class = EXAHdJobMapper
  mapreduce.job.inputformat.class = exasol.hadoop.EXAIOfomat$Input
  mapreduce.job.output.key.class = org.apache.hadoop.io.IntWritable
  mapreduce.job.output.value.class = org.apache.hadoop.io.Text
  mapreduce.job.output.fileoutputformat.outputdir = file:/resultdir
  mapreduce.job.outputformat.class = \
    org.apache.hadoop.mapreduce.lib.output.TextOutputFormat
  mapreduce.job.maxtaskfailures.per.tracker = 0
  mapreduce.job.maps = 16
  mapreduce.map.maxattempts = 1
 /
```

Besides defining the Hadoop job in EXASolution, the actual Hadoop job must naturally also be programmed. A Hadoop job for EXASolution differs from a standard Hadoop job in just a few points. The key difference lies in the way in which the data is exchanged between the Java classes and the database. The EXAPowerlytics Hadoop integration service makes a range of auxiliary classes available to this end. The following source code section shows a mapper framework for the integration of both services.
public class EXAHdJobMapper {

    public static class InputMapper
        extends Mapper<EXAIOFormat.Key,
                       EXAIOFormat.Value,
                       LongWritable, Text> {
          /* ... */

        public void map(EXAIOFormat.Key key, 
                        EXAIOFormat.Value value, 
                        Context context) 
            throws IOException, InterruptedException {

            Double input;

            for (boolean rowsLeft=true; rowsLeft; 
                 rowsLeft=value.readNextRow(false)) {
                input = value.getDouble(0);
                value.newResultRow().
                              setInteger(Math.round(input)).emit();
            } 
        } 
    }
}

Through the cooperation between EXASolution and Hadoop, it is possible to process unstructured and structured data together. This enables linking the most varied of data sources with company data. For example, data from social media or clickstreams can be linked with customer data, customer interaction data can be linked with sales figures, and image data can be linked with quality assurance data. The underlying added value is ultimately tapped by interlocking different data sources.
Figure 4: The integration of EXASolution and Hadoop. EXASolution accesses Hadoop via the redirector service, whereby the redirector is only charged with mediation. The actual communication and data transfer takes place directly between the EXASolution and Hadoop nodes.

5 Summary

A growing number of complex analytical calculations are being performed in the database. This is due, on the one hand, to the fact that the volume of data has grown to a level that exceeds the main memory of even the largest of computers and, on the other, to the increasing analysis options of modern database systems. In this context, the capabilities of the EXASolution analytical database and EXAPowerlytics advanced analytics framework offer a comprehensive range of options to perform complex calculations in the database. This includes the implementation of scripts written in Lua, R or Python, massively parallel distribution of these scripts, and the linking of external processes such as web services and host systems. Moreover, the integration of Hadoop provides the opportunity to analyse structured and unstructured data together.

EXASolution is a modern analytical in-memory database management system that allows complex calculations to be performed in the database. This enables the execution of business-relevant analyses on large data volumes comprising hundreds of terabytes, and thus questions to be answered that could not be answered in the past.
Bibliography


