# The ETLMR MapReduce-Based ETL Framework

Xiufeng Liu, Christian Thomsen, and Torben Bach Pedersen

Dept. of Computer Science, Aalborg University
{xiliu,chr,tbp}@cs.aau.dk

**Abstract.** This paper presents *ETLMR*, a parallel Extract–Transform–Load (ETL) programming framework based on MapReduce. It has built-in support for high-level ETL-specific constructs including star schemas, snowflake schemas, and slowly changing dimensions (SCDs). ETLMR gives both high programming productivity and high ETL scalability.

There is an ever-increasing demand for ETL tools to process very large amounts of data efficiently. Parallelization is a key technology to achieve the needed performance. Therefore, the "cloud computing" technology *MapReduce* [2] offering flexibility and scalability is interesting to apply to ETL parallelization. However, MapReduce is a general framework and lacks direct support for high-level ETL-specific constructs such as star schemas, snowflake schemas and SCDs. It is thus still very complex to implement a parallel ETL program with MapReduce and the ETL programmer's productivity is low. This paper presents the parallel ETL framework *ETLMR* [1] which directly supports high-level ETL constructs on MapReduce. The complexity of MapReduce is hidden from the user who only has to specify the transformations to apply and declarations of source data and destination tables. This makes it very easy to develop a highly scalable ETL program as only few lines of code and configuration are required.

ETLMR uses and extends the framework pygrametl [3] for code-based ETL and further uses Disco [4] as MapReduce platform. Fig 1 shows the architecture. An ETL flow in ETLMR comprises two sequential phases: dimension processing and fact processing, each of which runs as a separate job on the MapReduce platform. A job consists of a number of MapReduce instances (or tasks) running in parallel on several nodes in a cluster. An instance reads data from input files in a distributed file system (DFS) and processes the data and inserts it into dimension tables and/or fact tables in the DW.

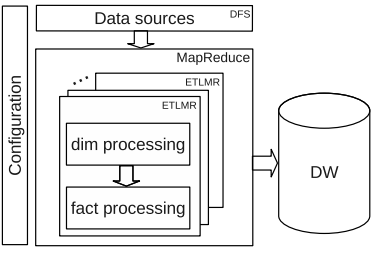


**Fig. 1.** The ETLMR framework

**Configuration.** ETLMR facilitates parallel ETL implementation by using a configuration file which defines dimension tables and fact tables. An object is

J.B. Cushing, J. French, and S. Bowers (Eds.): SSDBM 2011, LNCS 6809, pp. 586–588, 2011.

created for each of the target tables in a single statement where table name, attribute names, etc. are given. ETLMR supports different types of dimensions including slowly changing dimensions and snowflaked (i.e., normalized) dimensions. All the dimension classes have a common interface offering dimension operations such as *insert*, *lookup*, etc. In addition, the configuration file specifies the transformations to apply to the data and the number of map and reduce tasks to use (i.e., the level of parallelization). It is thus extremely easy to scale up/scale down an ETLMR-based program by only changing these numbers.

**Dimension processing.** During the dimension processing, ETLMR uses the (parallel instances of) MapReduce's *map* function to apply user-defined transformations to the source data. Further, the map function is used to find (i.e., project) the attributes that are relevant for the different dimensions. These subsets of the data are then processed by the (parallel instances of) MapReduce's *reduce* function to be inserted into the dimension tables. ETLMR offers several methods for doing this. The simplest method is *one dimension one task (ODOT)*. With ODOT, there is one, and only one, reduce instance for each dimension table. This makes it easy to avoid duplicated data, duplicated key values, etc. as all insertions to a given dimension table is done by one instance. Obviously, this method does, however, not scale well. Another supported method is *one dimension all tasks (ODAT)* in which all reduce instances (of which there can be any number) process data for all dimensions. This can, however, lead to problems with duplicated values (one reducer cannot see what another is about to insert) and duplicated key values. To remedy this, ETLMR, features the novel technique *post-fixing* where such problematic data autmatically is corrected when all data has been processed. Further, ETLMR offers dimension processing methods specialized for snowflaked dimensions. By processing the participating dimension tables in safe orders which respect foreign key relationships between tables, post-fixing can be avoided with these methods. Finally, ETLMR offers "offline" dimensions. With these, ETLMR does not communicate directly with the DW DBMS but instead stores data locally in the nodes. This leads to better performance.

**Fact processing.** In the fact processing, ETLMR looks up dimension keys in the (now processed) dimensions, does aggregation of fact data if needed, and bulk-loads the fact data into the DW. Several tasks do this in parallel.

**Scalability.** We have evaluated the scalability of the different processing methods on realistically sized datasets. The experiments [1] show that ETLMR achieves a nearly linear speedup in the number of tasks and compares favourably with other MapReduce data warehousing tools. To load a simple snowflake schema required 14 statements in ETLMR. In the MapReduce data processing frameworks Hive and Pig, the same schema required 23 and 40 statements, respectively. For a more complex schema with SCDs, ETLMR still only requires 14 statements, while Pig and Hive are not able to handle SCDs at all.

## References

1. http://www.cs.aau.dk/~xiliu/etlmr/ as of (April 13 ,2011)
2. Dean, J., Ghemawat, S.: MapReduce: A Flexible Data Processing Tool. CACM 53(1), 72–77 (2010)
3. Thomsen, C., Pedersen, T.B.: pygrametl: A Powerful Programming Framework for Extract-Transform-Load Programmers. In: Proc. of DOLAP, pp. 49–56 (2009)
4. http://www.discoproject.org as of (April 13 ,2011)