Lazy ETL in Action: ETL Technology Dates Scientific Data

Yağız Kargın
CWI Amsterdam
Yagiz.Kargin@cwi.nl

Milena Ivanova
Netherlands eScience Center
M.Ivanova@esciencecenter.nl

Ying Zhang
CWI Amsterdam
Ying.Zhang@cwi.nl

Stefan Manegold
CWI Amsterdam
Stefan.Manegold@cwi.nl

Martin Kersten
CWI Amsterdam
Martin.Kersten@cwi.nl

ABSTRACT

Both scientific data and business data have analytical needs. Analysis takes place after a scientific data warehouse is eagerly filled with all data from external data sources (repositories). This is similar to the initial loading stage of Extract, Transform, and Load (ETL) processes that drive business intelligence. ETL can also help scientific data analysis. However, the initial loading is a time and resource consuming operation. It might not be entirely necessary, e.g. if the user is interested in only a subset of the data.

We propose to demonstrate *Lazy ETL*, a technique to lower costs for initial loading. With it, ETL is integrated into the query processing of the scientific data warehouse. For a query, only the required data items are extracted, transformed, and loaded transparently on-the-fly.

The demo is built around concrete implementations of Lazy ETL for seismic data analysis. The seismic data warehouse is ready for query processing, without waiting for long initial loading. The audience fires analytical queries to observe the internal mechanisms and modifications that realize each of the steps; lazy extraction, transformation, and loading.

1. INTRODUCTION

Nowadays scientists receive increasingly large volumes of data. This data (e.g. images, time-series, sequences, etc.) is accompanied by self-descriptive information, called metadata (e.g. properties, parameters, description of data structures, etc.). Both data and associated metadata are collected in domain-specific files and file repositories. Efficient management and analysis of these high-volume scientific repositories have become pivotal for advancement in science. Currently, scientists use their legacy tools that mostly apply file-at-a-time data analysis. But these tools do not scale to the increasing dataset sizes [6]. Hence, there is a clear need of a data management tool that opens up these file repositories and performs analysis steps near-instantly.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Articles from this volume were invited to present their results at The 39th International Conference on Very Large Data Bases, August 26th - 30th 2013, Riva del Garda, Trento, Italy.
Proceedings of the VLDB Endowment, Vol. 6, No. 12
Copyright 2013 VLDB Endowment 2150-8097/13/10... $ 10.00.

In business intelligence, data warehouses provide the means for analytical tasks. The data from external source datastores is extracted, transformed, and loaded (ETL) into the data warehouse using ETL processes. These processes facilitate the initial loading, that enables analysis of business data within the data warehouse. They also facilitate the periodic refreshment to keep the data up-to-date [18]. In its essence, this setting addresses the need that is similar to that of scientific data analysis. In addition, a file repository is one of the kinds of source datastores that ETL processes deal with [19]. It is, therefore, natural to consider the techniques that helped meeting the requirements of business intelligence to help scientific data analysis.

Traditional ETL relies on eagerly filling the data warehouse with data. This requirement, however, is a high initial investment of time. It makes the first step of analysis far from near-instant performance. This has been recognized as one of the major obstacles in adopting database solutions for scientific data [17, 7]. To overcome this shortcoming without losing the merits of traditional ETL, the Lazy ETL approach was introduced in [12] at BIRTE 2012. Here, we demonstrate Lazy ETL, a query-driven, on-demand ETL system, that mitigates the burden of initial loading.

Initial loading is reduced to loading only metadata to achieve the necessary scalability. Consequently, the metadata is used to identify the actual data required by a query. At query time, the actual data is extracted from the files that contain the required data. It is transformed and loaded transparently. Our approach reduces the cost for bootstrapping a scientific data warehouse for a new file repository. It also makes updating and extending a warehouse with modified and additional files more efficient. It can be also considered as a step forward in the 'near real-time ETL' vision put by Dayal et. al [5].

We implemented our approach in MonetDB [2] (an analytical column-store database with support of a scientific declarative query language, SciQL [20]). We propose to demonstrate Lazy ETL through seismic data analysis. The source datastore is a repository containing files in mSEED format [1]. The system serves the need of mining interesting seismic events. With the initial loading of only metadata, the data warehouse is instantly ready for analysis queries. This presents a significant reduction of the overall time from source data availability to query answer. The overall system provides easy browsing of metadata and navigation in the data. At the same time, the demo shows the internal technical details behind the scene.

2. RELATED WORK

The data ingestion needs of scientists are similar to those of business analysts. We describe the state of the art of data ingestion with business ETL and scientific data ingestion.

ETL Research. ETL research is mostly focused on sources of Online Transaction Processing (OLTP) datastores. The most popular setup is the following [4]: a row oriented operational DBMS for the OLTP-load and a data warehouse for the analytical needs. After the initial loading, new data that enters the operational system is loaded into the warehouse in intervals using Extract, Transform & Load (ETL) processes [9]. However, the data in the data warehouse might become stale, depending on the update window [14]. Moreover, all data is stored redundantly, making the system space-inefficient [9]. The ETL research addresses some of these problems from a different angle than we do. They focus on update intervals [11], incremental updates [10], bounds on staleness [13], etc.

Scientific Data Ingestion. The problem of scientific data ingestion can be considered as loading data from external files. This has received some attention from the database community. The SQL/MED (Management of External Data) standard [16] offers an integration of external files into a relational schema. However, the file content straightforwardly maps to tables, and the burden of managing the files is still on the user. There are also well-established techniques like external tables in major commercial systems. They enable access to data in external sources as if it were in a table in the data warehouse. However, they require every query to access the entire dataset, because they are actually intended for loading a file's content. To address that, DBMS query processing over flat data files is presented as NoDB [3]. They provide query-driven on-demand loading as we do. However, they cannot handle complex file formats that are common in science applications like MiniSEED (mSEED) or Geo Tagged Image File Format (GeoTIFF), etc. These files contain complex schemas of different types of objects and even pointers that should be resolved to foreign key constraints when loading. Plus, they do not exploit the metadata for selectively loading during query processing.

3. LAZY ETL

Traditional ETL processes load and pre-aggregate all data that can possibly be queried (i.e. *eager ETL*). This is because it is unclear which data subset is interesting without workload knowledge. However, the data warehouse may only need a subset of the available data to answer a user's (ad-hoc) query. Nevertheless, the user still has to wait for the possibly lengthy *eager* loading of all the input data.

Our goal is to mitigate the burden of lengthy initial loading. For that purpose, we consider ETL as part of the query execution, breaking with the traditional paradigm. Actual data access is left to query-time. That is, we run ETL only when required and only for the data that is required by a query. This approach is called *Lazy ETL* [12]. Initial loading covers only metadata. The actual data accessed lazily is kept in a cache of limited size. If there is any update in the repository, refreshments are handled in the cache when the data warehouse is queried.

Metadata vs. Actual Data. Without any knowledge about the input data files, all available files have to be considered "relevant" for a given query. *Metadata* provides a means to understand the data. Thus it is central to scientific data access [6]. Metadata is data that provides insight into the content of a record, a file or the repository. In science, this might cover data such as the starting time of measurements, the sampling rate, the position of a sensor, etc. Analytical queries usually refer to a subset of metadata in order to define the subset of data they are interested in. In most cases, traditional metadata is smaller in size and cheaper to acquire than actual data. It is even cheaper if metadata is encoded in the filename. In that case, the file does not even need to be read. Therefore, it is our conscious preference to load the metadata eagerly to exploit it for selective access to the actual data. The definition of metadata might, however, differ with each file format, use case and system performance. Nevertheless, it is relatively straightforward in most of the cases to understand which subset of the data describes the rest of the data. We call all other data the *actual data*. In science domains, actual data items usually describe data points, e.g., a timestamp and one or many measured values. In practice, the majority of the data is actual data.

In the following we illustrate how to replace a traditional ETL process with our approach. We visit each step of the lazy ETL process to explain the internals of the system that we will demonstrate.

3.1 Lazy Extraction

The first goal of extraction is identifying the correct subset of source data that has to be submitted to the subsequent transformation and loading. In a typical ETL process, this is usually performed in a bulk fashion. However, in Lazy ETL extraction is performed per-query basis since we use metadata to identify the required subset of the source (files). The extraction might also include the decompression if necessary.

In the worst case, the required subset of actual data that is subject to lazy ETL is the entire repository. In that case, it involves extracting the complete source as during initial loading of the traditional ETL. In the best case, the required subset of actual data is in the cache and up-to-date. Then no ETL process needs to be performed.

Lazy extraction is implemented as two steps of query plan modification. First, the query plan generated by the data warehouse is modified at compile-time. The plan is reorganized so that the selection predicates on the metadata are applied first. Second, once this part of the plan is executed, a plan rewriting operator is executed. This operator uses plan introspection provided by the MonetDB system and performs a plan modification at run time. It injects operators that either access data from the cache or extract data only from the necessary files. Internally these operators use external scientific library calls to extract the data from the specific file formats. This allows us to seamlessly integrate lazy extraction with query evaluation.

3.2 Lazy Transformation

An ETL process might include diverse transformations. Examples are schema-level transformations, one-to-many mappings, etc. [18] [19]. Eager ETL processes apply these transformations on the entire input data once and then on the possible updates later. Most of these transformations can, however, be expressed as relational algebra operators (i.e. using projections, joins, etc.). So, we express them in SQL for lazy ETL. We implement all necessary transformations as

non-materialized views. Thus, the transformations are performed at query time, because view definitions are simply expanded into the query. After the required actual data is extracted, the transformations are done using relational operators as specified in the query plan of the data warehouse. This means that only the minimal amount of data items needed from the sources are transformed and the transformations benefit from query optimization. In addition, some transformations that are performed on a fine granularity (e.g., record-level and value-level transformations, and data cleaning) are added to the end of extraction phase.

Lazy transformation is provided with a run-time modification of the query plan. After the rewriting operator injects the operators for lazy extraction, the references to relational tables that contain actual data are removed from the query plan. The operators for transformation are then assigned to each of the intermediates resulting from operators for lazy extraction.

3.3 Lazy Loading

The extracted and transformed data is materialized into the internal data structures of the data warehouse. This operation is called loading in traditional ETL. This is usually done by bulk loading data through a DBMS-specific utility. For us, materialization of the extracted and transformed data is simply caching the result of a view definition (i.e. some of the intermediate results). Fortunately, MonetDB already supports such caching in the form of *intermediate result recycling* [8].

Lazy loading is provided through integration of Lazy ETL with intermediate result recycling. Usually, the end result of a view is saved in the cache. A least recently used (LRU) policy is used for cache maintenance. In the ETL setting, this cache also has to take care of the updates. We do that lazily, too. The cache makes use of required files' last modified timestamp, and compares that with the admission timestamp of that data to the cache. If the required data in the cache is outdated or it is not in the cache, then it is extracted from updated files during lazy extraction. Otherwise the cached data is used. We adjust the cache size according to the input dataset size, but it is not larger than the size of system's main memory.

4. DEMONSTRATION

We demonstrate Lazy ETL based on seismic data analysis. In seismology, SEED [1] is the most widely used standard file format to exchange waveform data among seismograph networks. A SEED volume mainly consists of the waveform time series, which are highly compressed. For example, a SEED repository requires up to 10 times the original storage size when loaded into a database [12]. Additionally, a SEED volume has several ASCII control headers. The control headers contain the metadata. In this demo we use Mini-SEED (mSEED) variant, which reduces the SEED metadata to the most widely used subset. The sizes of an mSEED file commonly vary from 4 KB to several MBs. Millions of them are stored in remote file repositories with direct FTP access, e.g., [15].

Each mSEED file contains multiple mSEED records. An mSEED record represents the sensor readings over a consecutive time interval, i.e., a time series. The normalized data warehouse schema, as proposed in [12], includes three tables, that are straightforwardly derived from the mSEED format.

```
SELECT AVG(D.sample_value)
FROM mseed.dataview
WHERE F.station = 'ISK'
  AND F.channel = 'BHE'
  AND R.start_time > '2010-01-12T00:00:00.000'
  AND R.start_time < '2010-01-12T23:59:59.999'
  AND D.sample_time > '2010-01-12T22:15:00.000'
  AND D.sample_time < '2010-01-12T22:15:02.000';

SELECT F.station,
  MIN(D.sample_value), MAX(D.sample_value)
FROM mseed.dataview
WHERE F.network = 'NL'
  AND F.channel = 'BHZ'
GROUP BY F.station;
```

Figure 1: Two sample queries.

Two metadata tables F and R hold metadata per mSEED file and mSEED record, respectively. Whereas one actual data table D stores all the data points (i.e. tuples of sample time and sample value from all files and records). Each mSEED file is identified by its URI. Each record is identified by its (record) sequence number (unique per file). These identifiers form also the foreign key relations between the three tables. We define a (non-materialized) view `dataview` that joins all three tables into a (de-normalized) "universal table".

Seismic data analysis contains tasks that help hunt for interesting seismic events. Such tasks include finding extreme values over *Short Term Averaging* (*STA*, typically over an interval of 2 seconds) and *Long Term Averaging* (*LTA*, typically over an interval of 15 seconds), retrieving the data of an entire record for visual analysis, etc. Two of the sample analysis queries are given in Figure 1. This is to make attendees already familiar with the kind of queries that can be fired. The first query computes a short term average over the data generated at Kandilli Observatory in Istanbul (`ISK`) via a specific channel (`BHE`). The second query calculates both minimum and maximum values (amplitudes) for a channel (`BHZ`) per station in the Netherlands (`NL`) without restricting the time period. For more detailed information about the data, schema, and queries please refer to [12].

The demonstration scenario allows to zoom into the details of the Lazy ETL system and functioning. To illustrate, a snapshot of the GUI is shown in Figure 2. The numbers in the figure comply with the numbers on the list below. In particular, the attendees can experiment with (1) initial loading of only metadata from an mSEED repository of any size, (2) browsing the metadata and navigating through the data with their own queries, (3) comparing the performance to the eager ETL approach, (4) observing the query plans and changes on them while realizing lazy extraction of the actual data, (5) observing the files containing required actual data that is lazily extracted for the query. (6) observing the plans generated on the fly for lazy transformation, (7) observing the contents of the cache and updates to it happening if needed (i.e. lazy loading), (8) looking through the log to see what operations are performed and in which order. In addition, there are also predefined queries for demonstration purposes that can be used when looking for interesting seismic events.

5. ACKNOWLEDGMENTS

The work reported here is supported by the EU-FP7-ICT project TELEIOS and the Dutch national project COMMIT. We wish to thank our colleagues at KNMI and Holger

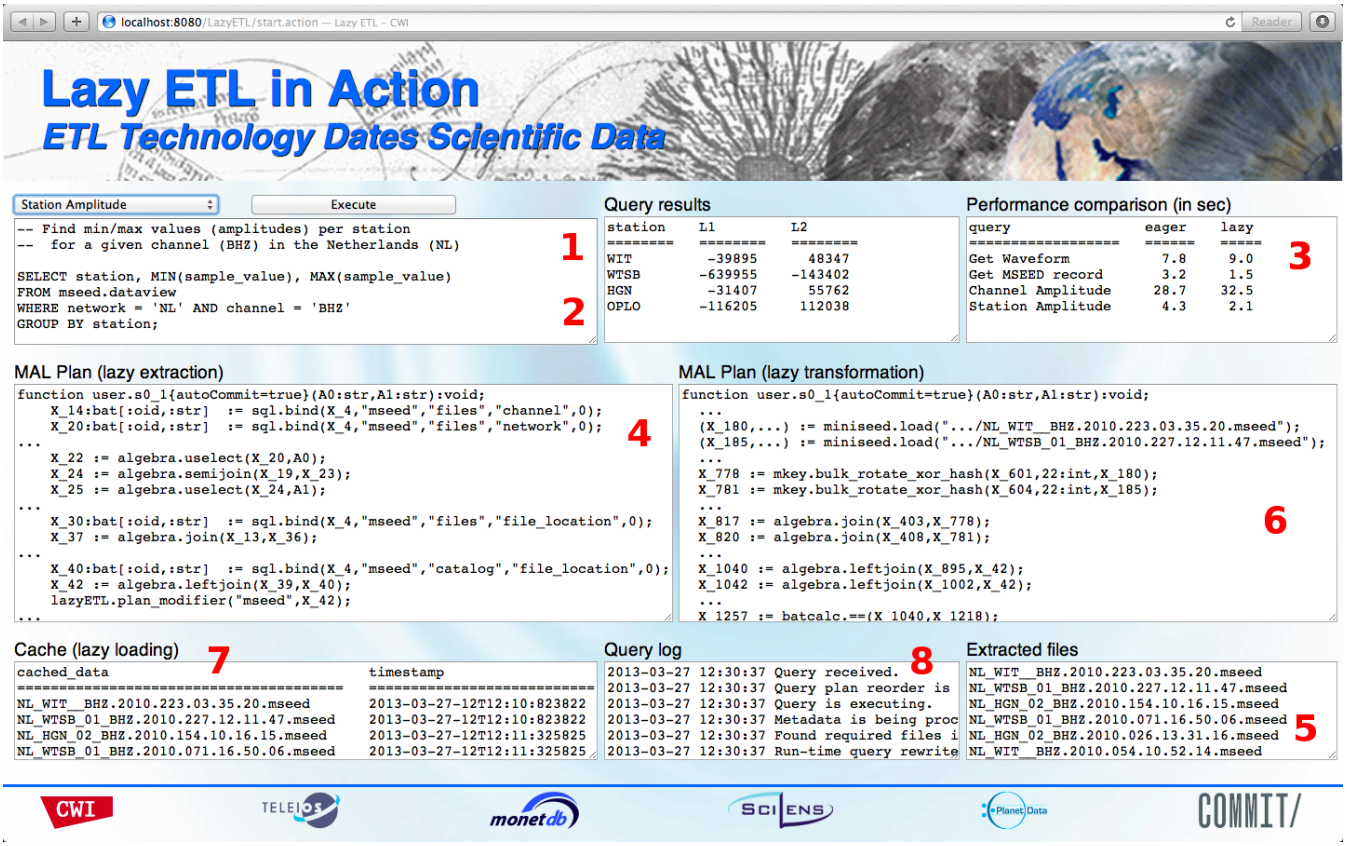

Figure 2: GUI of Lazy ETL.

Pirk, for constructive guidance on the functionality and implementation of Lazy ETL.

6. REFERENCES

[1] *Standard for the Exchange of Earthquake Data.* Incorporated Research Institutions for Seismology, February 1988.

[2] MonetDB, Column-store Pioneers. www.monetdb.org, 2013.

[3] I. Alagiannis et al. NoDB: Efficient Query Execution on Raw Data Files. In *SIGMOD*, 2012.

[4] S. Chaudhuri and U. Dayal. An overview of data warehousing and OLAP technology. *ACM Sigmod record*, 26(1):65–74, 1997.

[5] U. Dayal et al. Data integration flows for business intelligence. In *EDBT*, pages 1–11. ACM, 2009.

[6] J. Gray et al. Scientific Data Management in the Coming Decade. *SIGMOD Record*, 34(4), 2005.

[7] S. Idreos et al. Here are my data files. here are my queries. where are my results? In *CIDR 2011*.

[8] M. Ivanova et al. An Architecture for Recycling Intermediates in a Column-store. In *SIGMOD Conference*, pages 309–320, 2009.

[9] M. Jarke, M. Lenzerini, Y. Vassiliou, and P. Vassiliadis. *Fundamentals of data warehouses*. Springer Verlag, 2003.

[10] T. Jörg and S. Deßloch. Towards generating etl processes for incremental loading. In *IDEAS*, pages 101–110. ACM, 2008.

[11] T. Jörg and S. Dessloch. Near real-time data warehousing using state-of-the-art etl tools. *Enabling Real-Time Business Intelligence*, pages 100–117, 2010.

[12] Y. Kargin et al. Instant-On Scientific Data Warehouses — Lazy ETL for Data-Intensive Research. In *BIRTE*, 2012.

[13] J. Kiviniemi, A. Wolski, A. Pesonen, and J. Arminen. Lazy aggregates for real-time OLAP. *Lecture notes in computer science*, pages 165–172, 1999.

[14] W. Labio, R. Yerneni, and H. Garcia-Molina. Shrinking the Warehouse Update Window. In *In Proceedings of SIGMOD*, pages 383–394, 1998.

[15] ORFEUS. Seismology Event Data (1988 - now). ftp://www.orfeus-eu.org/pub/data/pond/, 2013.

[16] SQL/MED. ISO/IEC 9075-9:2008 Information technology - Database languages - SQL - Part 9: Management of External Data (SQL/MED).

[17] M. Stonebraker et al. Requirements for Science Data Bases and SciDB. In *CIDR*, 2009.

[18] P. Vassiliadis. A survey of extract–transform–load technology. *International Journal of Data Warehousing and Mining (IJDWM)*, 5(3):1–27, 2009.

[19] P. Vassiliadis and A. Simitsis. Extraction, transformation, and loading. *Encyclopedia of Database Systems*, pages 1095–1101, 2009.

[20] Y. Zhang et al. SciQL: bridging the gap between science and relational DBMS. IDEAS '11, pages 124–133. ACM.