A Framework for Detecting Unnecessary Industrial Data in ETL Processes

Philip Woodall
Department of Engineering
University of Cambridge, UK.
phil.woodall@eng.cam.ac.uk

Torben Jess
Department of Engineering
University of Cambridge, UK.

Mark Harrison
Department of Engineering
University of Cambridge, UK

Duncan McFarlane
Department of Engineering
University of Cambridge, UK.

Amar Shah,
Department of Engineering
University of Cambridge, UK

William Krechel
Boeing, USA

Eric Nicks
Boeing, USA.

***Abstract*—Extract, transform and load (ETL) is a critical process used by industrial organisations to shift data from one database to another, such as from an operational system to a data warehouse. With the increasing amount of data stored by industrial organisations, some ETL processes can take in excess of 12 hours to complete; this can leave decision makers stranded while they wait for the data needed to support their decisions. After designing the ETL processes, inevitably data requirements can change, and much of the data that goes through the ETL process may not ever be used or needed. This paper therefore proposes a framework for dynamically detecting and predicting unnecessary data and preventing it from slowing down ETL processes – either by removing it entirely or deprioritizing it. Other advantages of the framework include being able to prioritise data cleansing tasks and determining what data should be processed first and placed into fast access memory. We show existing example algorithms that can be used for each component of the framework, and present some initial testing results as part of our research to determine whether the framework can help to reduce ETL time.**

Keywords—Extract, transform and load, ETL, Data warehouse, reduce ETL, unnecessary data, data overload, detecting unnecessary data.

I. INTRODUCTION

Data warehouses are currently a critical infrastructure used to underpin the operational, reporting and data analytics capabilities of many leading industrial organisations. In order to populate data warehouses, data goes through a process known as extract, transform and load (ETL). ETL extracts data from various source systems, transforms it as required, and loads it into the data warehouse, or any "target" system. One example use of ETL is to integrate data throughout the supply chain, see for example, [1]. Another example ETL process could be the transfer of a batch of engineering parts, which have been requested by engineers in one system, to the procurement system that is used to select suppliers for the orders of the parts. These orders may then also be transferred by ETL into the order release system, ready to send to the suppliers. Moreover, for reporting, these orders may be sent via ETL to a data warehouse that is used by managers to, for example, scrutinise various orders over the last month.

However, the problem is that ETL is a time-consuming and expensive process, which can leave decision makers in industrial organisations waiting for the critical data they need to make decisions and carry out their operations. In the previous example, if an ETL process does not transfer the required orders to the order release system in time, then the end result could be that there are delays while the assembly line waits for the parts to be available. Moreover, this could cost the business further, if the price for the parts increases in the time it takes to release the orders.

The standard approach is to design the data warehouse and ETL procedures so that they only transfer the necessary data [2]. However, data requirements can change over time, rendering the initial ETL designs obsolete. To avoid the problem of not having transferred the data they need, we observed that industrial organisations often deliberately ETL more data than the initial requirements dictate; the advantage is that the ETL process does not need to be changed every time a decision maker requests new data, which is a problem for companies, because the ETL code can be very difficult to change especially when external consultants were the developers, and managing the ETL code is a research area in itself [3]. The effect of this, however, is that ETL processes can take a long time to complete the data transfer. Although this is still a problem, many organisations prefer to make the whole set of data available to the decision makers, at the expense of the time taken for the ETL process to complete and the risk of receiving data too late.

We therefore investigated the problem of determining whether it is possible to have the necessary data available to decision-makers on-time, while ensuring that they can access all the data they need. In this paper we introduce a framework that can detect unnecessary or low priority data and either prevent it from being transferred into a data warehouse – or arrange for it to be transferred in a lower priority second phase of the ETL process – so that decision makers have only the necessary data, and no unnecessary data has slowed down the ETL process. The framework detects unnecessary data in systems dynamically (i.e. continually and as the systems is used), rather than statically at design time or redesign time. This paper presents our research-in-progress towards developing and evaluating this framework. In this paper, we introduce the framework and various algorithms that can be applied to each part of the framework, thereby showing that it is realisable. Using an industrial example, we also show some initial results that illustrate that it is possible to detect unnecessary data and infer what new data is likely to be unnecessary before it is extracted, transformed and loaded into other systems.

We argue that our approach improves on the current practice of designing ETL procedures that either ETL all data (and slow the ETL) or choose the relevant subset of data to ETL (that force the ETL procedures to be updated if requirements change). With our approach the aim is that industrial organisations can continue to develop ETL processes that include all data, so all data is potentially available to the decision maker, without incurring increased ETL execution time.

Reducing unnecessary data in systems has additional benefits besides reducing ETL time and ensuring that decision-makers receive data on-time: such as, reducing unnecessary data storage costs, and research from Gartner argues that companies are spending 20% more than necessary by backing up superfluous data [4]. Moreover, with only the necessary data, any data cleansing tasks will be focussed on correcting only the most important data.

This paper first describes the relevant definitions and assumptions for the research in section two before reporting the related work that comes closest to identifying unnecessary data and reducing ETL time in section three. Sections four and five describe our framework and some initial testing results of an implementation of the framework using fictitious data based on an industrial example. Finally, we describe the conclusions of our paper.

II. Definitions and Assumptions

We define unnecessary data as data that is not needed by the users of an information system for any of their tasks/decisions. To make this concrete, for this paper we consider any database field and/or record that does not affect the result of a user's query (e.g. SQL query) to be unnecessary for that user. This extends to: any field and/or record that does not affect any user queries (for all users and all queries) is unnecessary. Note that a record may not affect a query due to errors in the record's values, and we discuss this later in the paper.

An important assumption for this work concerns the use of data warehouses (or the target system): we assume that the target system has a static set of queries that do not change (i.e. are not ad hoc queries). This is the case for reporting systems that issue the same queries to the data over time, and only the underlying data changes rather than the queries. Real examples of this are when engineering parts orders are transferred to a reporting system and the manager always views a report of the most expensive parts ordered over the last month.

III. Related Work

The area of information overload is related to this work as it aims to reduce the amount of information the decision maker is exposed to [5]. Suggestions in this area include using publish and subscribe ("push" technologies) to reduce unnecessary data by allowing users to choose what data they subscribe to. Our approach builds on this work by looking at the problem at a finer level of granularity, in particular, at the actual records of data and whether these are relevant or not.

Record linkage merge/purge solutions [6] could also be seen as ways of removing unnecessary data. However, it is not only duplicate records that can be unnecessary, and therefore our approach extends the scope of this work.

Incremental loading [7] is one technique that could also reduce the time it takes to ETL data. Our framework can be used to support incremental load techniques for data warehouses, because it reports, for each item of data, how many reports will need the data. Incremental load techniques predict when data needs to be transferred and our technique predicts what data needs to be transferred.

Other approaches that attempt to reduce the time to ETL data are related to the management of the ETL code [3]. These attempt to make it easy to edit and maintain the ETL procedures; therefore, they also make it easier to modify the ETL to remove the unnecessary data transmitted each time new data requirements appear.

IV. A Framework For Detecting And Removing Unnecessary Data

The proposed approach to detecting and predicting unnecessary data consists of six stages: Detect, Trace, Document, Model, Predict and Apply (as shown in Figure 1). Each of these stages is described in the following subsections.

A. Detect

Using the current set of data in the target system and the queries that are applied to the data, the first stage detects which fields and records are necessary. To detect unnecessary records, data lineage methods [8] can be used for this stage if we consider that data is unnecessary if the user never sees the data as part of their view of the system. For example, the user may run three queries against the data and see a view (the query result) for each query. Any data that does not contribute to all views is considered unnecessary, and lineage methods are designed to detect these "contributing" records. Another method to detect what records contribute to a query result is to use a sensitivity analysis: in testing our approach, we implemented a sensitivity analysis for this stage that iteratively removed individual records from the target and tested to see if

the query result was affected in any way. If a record was removed, and the result was unaffected, then this record was considered to be unnecessary. Any record that was unnecessary for all queries on the target was considered to be unnecessary for the user. Note that to detect different "witnesses" [8] where individually a record does not affect the result, but together with other records it does, it is possible to apply an n-order sensitivity analysis ($n > 1$) to detect each case. The sensitivity analysis was implemented because it is a simple method to develop. However, in practice, the lineage methods are more likely to be appropriate in most operational environments. Note that although an n-order sensitivity analysis is inefficient, it has the advantage that it does not need an SQL query to operate (which many of the lineage methods require). Provided that it is possible to observe the result and change the input, the sensitivity analysis can detect unnecessary data. This approach is therefore suitable for systems that employ SPARQL queries and application specific queries or filtering, such as data mining programs that filter the records via means other than SQL queries.

Detecting fields is considerably easier because it is possible to check for the presence of the name of the field in each query. Any field that was not present in all queries is considered to be unnecessary for the user. Note that no data need be present in the target system to detect unnecessary fields; only the set of queries is needed.

B. *Trace*

The trace step involves finding the relevant fields and records in the source that were extracted, transformed and loaded to the target system. ETL transforms may change the name of a field; for example, "name" in the source could be converted to "surname" in the target, and the trace step involves discovering the name of the relevant field in the source. Note that, depending on the transform, a source field might contribute both to a necessary field and an unnecessary field in the target. So the requirement is to determine what fields in the source contribute only to the unnecessary fields in the target.

Similarly, the records in the target will have been derived from records in the source. Cui and Widom [9] define three different ways in which target records can be derived from source records, which equate to the following: one-to-many, many-to-one, and many-to-many. The trace step also involves discovering which records in the source only contribute to the necessary records in the target.

Note that determining the field mappings does not always have to be done dynamically if the fields in the target stay the same; it can be defined by the user once and recorded in a mapping document. The trace stage can simply use this document to map the target field names into source field names. Tracing the relevant records is, however, more complicated and needs to be done dynamically because the records in most systems change daily. Cui and Widom [9], as part of their lineage research also developed methods to trace the lineage of records that have been through a general transform such as an ETL transform, and therefore their methods can be used to determine the lineage of the unnecessary records. In our initial test implementation, we do not perform any data transforms, and hence do not implement the trace stage; this is planned as future work.

C. *Document*

From the results of the detect and tracing stages, the unnecessary fields and records in the source (and each source, if there is more than one) should be documented. In our test implementation, we simply document the unnecessary fields in a text file and append an extra field to the source system that indicates whether the record is unnecessary or not. In an operational system, appending fields may not be possible and therefore storing the identities of the unnecessary records externally is an alternative approach.

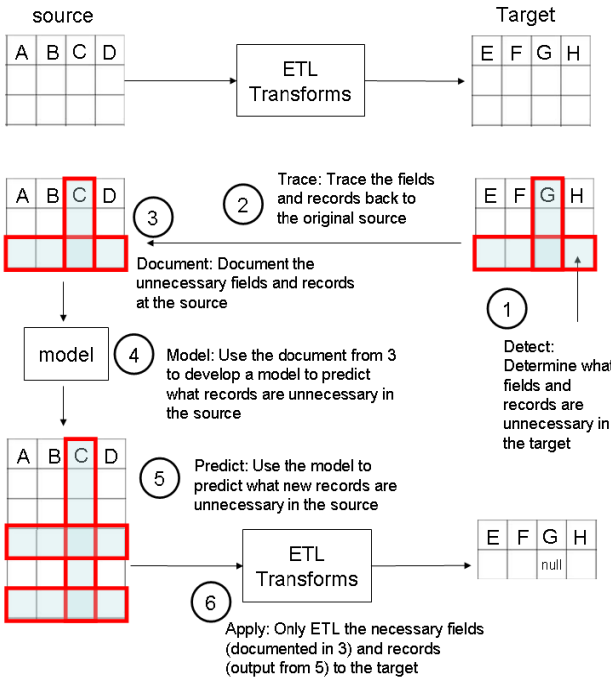

Fig. 1. Stages of the framework to detect and predict unnecessary data

D. *Model*

The model stage uses the documentation of the unnecessary records (note: the unnecessary fields are not needed) to build a classification model that can be used to predict which other source records are likely to be unnecessary in the target. In our implementation, we used a data mining classification algorithm to build this model. The training set for the model is the set of records in the source system with the relevant classification label (necessary/unnecessary) appended to each record.

E. *Predict*

When new records are added to the source system as part of its daily operation (such as, new orders, point-of-sale transactions, new customers, new suppliers etc.), the model from the previous stage can be used to determine which new records are likely to be unnecessary in the target system. The

input to the model is the set of new records in the source system and the classification model should output the predicted label for each record (i.e. whether it is necessary/unnecessary).

F. *Apply*

Once the unnecessary records have been predicted and the unnecessary fields have been documented, it is possible to apply some countermeasure to mitigate the effects of the unnecessary data. Such countermeasures include preventing the data from being extracted, transformed and loaded into the target system (for example, by pointing the ETL to a view of the source that excludes the unnecessary records and fields), or prioritising the necessary data in the ETL process over the unnecessary data etc.

V. Initial Testing Of The Framework Using An Industrial Scenario

So far, the Detect, Document, Model and Predict stages of the framework have been implemented. This is convenient because it is possible to determine which data is unnecessary and predict which new data is unnecessary without the complexity of dealing with ETL transformations. Using a real industrial case example from a large manufacturing organisation as a start point, we generated fictitious data that could be used to test our framework. We worked with professionals of the organisation to ensure that the data and the queries used are relevant and realistic to a real-world scenario.

A. *Experimental Setup*

We tested the Detect, Document, Model and Predict stages on an automatically generated dataset that contains "goods received" data, which is a list of parts delivered to an organisation (see the table in Figure 2, note that the "contributes" field has been generated by the Detect stage and is not part of the original dataset). We used random number generators to generate the unitSize, noOfUnits (quantity) and unitCost fields. The list of part numbers and part descriptions were obtained from a publically available online catalogue of light-aircraft parts, and itemIndex is the unique identifier for each delivered part. The deliveryNo field indicates when the parts were delivered (for instance, deliveryNo "1" was the first delivery, and deliveryNo "2" was the second delivery etc.). Note that in the case of record 4, it has a zero noOfUnits, which indicates that it was expected to be delivered in deliveryNo 1, but was actually delivered in deliveryNo 2.

This dataset (containing 86 records) was partitioned into two equal parts, where the second partition acted as the "new" data that is in the source system, but has yet to be extracted, transformed and loaded into the target. The following two queries, that represent two typical reports that a user of the system would want to observe each day, were issued to the first partition.

1. SELECT partNo, sum(unitCost*noOfUnits) as total FROM table GROUP BY partNo;
2. SELECT partNo, unitCost FROM table WHERE unitCost > '25';

B. *Results For the Industrial Scenario*

After these queries were issued, the first partition has therefore been "used" and the Detection stage can be applied to detect unnecessary data for these two queries. Figure 2 shows a screenshot of an application that implements the sensitivity analysis algorithm. It has appended the "contributes" field to the dataset; this column indicates the number of query results each record contributes to. It shows that record 4 is unnecessary. Intuitively, this is the case because the noOfUnits is zero and hence it will not affect query 1 and its unitCost is less than 25, so it will not feature in the results for query 2. Record 1 features in both query results (hence the "2" in the contributes field), and record 7 features in the results for query 2 only (hence the "1" in the contributes field). The unused fields (attributes) are also listed at the bottom of the application in Figure 2 after having checked for the presence of each field in the two queries issued to the dataset.

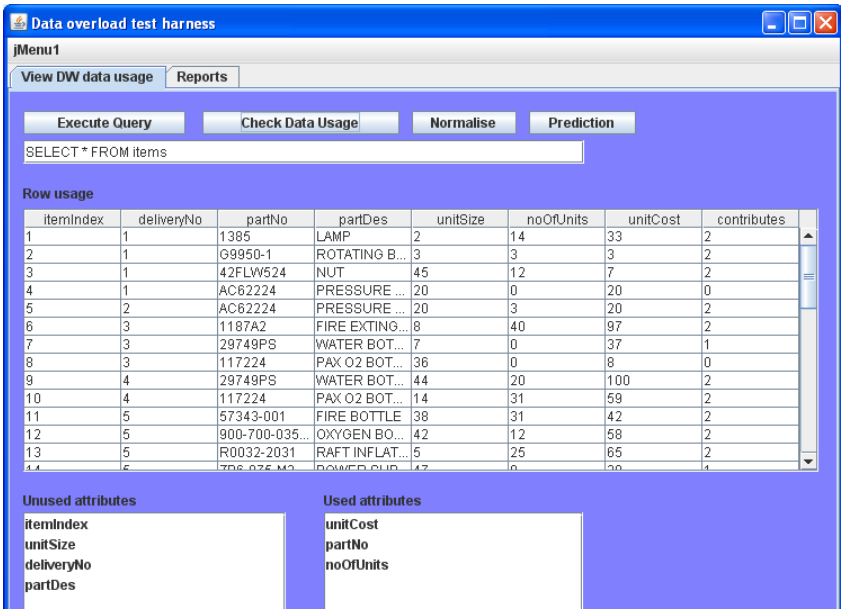

Fig. 2. A screenshot of the unnecessary data tool

In order to develop the model for predicting what data in the second partition is useful, the table shown in Figure 2 (only 13 records are shown in this table because of space limitations) was used as a training set to build a data mining classification model (with the "contributes" field being the classification label). RapidMiner (http://rapidminer.com/) version 5.2.008 was used to build this model using a decision tree that operates in a similar manner to the C4.5 algorithm [10]. The resulting decision tree is shown in Figure 3.

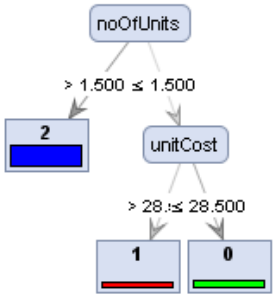

Fig. 3. The decision tree classification model

TABLE I. PARTIAL RESULTS SHOWING THE INCORRECT PREDICTIONS

partDesc	noOf Units	unit Cost	contributes	prediction
POWER SUPPLY	10	12	2	2
CIRCUIT BREAKER	28	25	2	2
PRESSURE SWITCH	26	6	2	2
CIRCUIT BREAKER	1	93	2	1
FUEL DRAIN VALVE	10	27	2	2
ELEVATOR ACTUATOR	24	17	2	2
O2 REGULATOR	29	79	2	2
PRESSURE SWITCH	19	83	2	2
VALVE ASSY	13	13	2	2
VALVE	4	66	2	2
OH SWITCH	1	71	2	1
CENTRIFUGAL SWITCH	12	3	2	2
TEMP SENSOR	40	84	2	2

Note that the first query does not fetch records with a noOfUnits equal to 0, and the second query only fetches records with a unit cost greater than 25. The decision tree has only been trained on a small sample of data (43 records), and so one can expect that the thresholds used to predict the "contributes" values will converge on the correct thresholds as more data is used to build the model. However, for this sample of data, the prediction is incorrect, for records where the noOfUnits is "1" or the unitCost is between 25 and 28.5 exclusive. In our second dataset partition (i.e. the new data), this amounted to two prediction errors in 45 records. Table I shows a sample of the results where the second circuit breaker and the "OH switch" records do form part of the query answer for both queries, but the prediction incorrectly states that they are only needed for one query.

VI. CONCLUSION

Predicting what data is unnecessary for industrial organisations is becoming increasingly critical with the increase in the amount of data companies must sift through. We presented a framework to reduce this burden and help to optimise ETL data processing tasks by focussing them only on the necessary data. Our framework has numerous potential benefits including: being able to reduce ETL processing time and therefore get data to decision makers faster, and reducing the need to store unnecessary data and therefore the costs of memory and speed of queries. In addition to improving the ETL process, our framework can give an indication of data that users do not know exists. For example, a data quality problem could be causing a record to not appear in the query results, and the users will therefore never know about this data until the data quality problem is corrected. Hence, our approach provides a way to target data quality improvement initiatives as well as discover data that could be useful but is never being seen. It could also give an indication of how much information an organization gains from a particular dataset.

REFERENCES

[1] N. Moalla, A. Bouras, and Y. Ouzrout, "Production Data Integration in the Vaccine Industry," in 2007 5th IEEE International Conference on Industrial Informatics, 2007, vol. 1, pp. 603–608.

[2] R. Kimball and J. Caserta, The Data WarehouseETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data, 1st edition. Wiley, 2004.

[3] A. Albrecht and F. Naumann, "Managing ETL Processes.," NTII, vol. 8, pp. 12–15, 2008.

[4] B. Goodwin, "Businesses wasting billions storing unnecessary data," Computerweekly.com, January 22nd, 2013.

[5] M. J. Eppler and J. Mengis, "The Concept of Information Overload: A Review of Literature from Organization Science, Accounting, Marketing, MIS, and Related Disciplines," The Information Society, vol. 20, no. 5, pp. 325–344, 2004.

[6] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios, "Duplicate Record Detection: A Survey," IEEE Transactions on Knowledge and Data Engineering, vol. 19, no. 1, pp. 1 –16, Jan. 2007.

[7] T. Jörg and S. Deßloch, "Towards Generating ETL Processes for Incremental Loading," in Proceedings of the 2008 International Symposium on Database Engineering; Applications, New York, NY, USA, 2008, pp. 101–110.

[8] J. Cheney, L. Chiticariu, and W.-C. Tan, Provenance in Databases: Why, How, and Where. Now Publishers Inc, 2009.

[9] Y. Cui and J. Widom, "Lineage tracing for general data warehouse transformations," The VLDB Journal, vol. 12, no. 1, pp. 41–58, May 2003.

[10] R. Quinlan, C4.5: Programs for Machine Learning (Morgan Kaufmann Series in Machine Learning). Morgan Kaufmann, 1992.