AScale: Simple and fast ETL+Q scaling for small and big data

Pedro Martins, Maryam Abbasi, Pedro Furtado
University of Coimbra
Department of Informatics
Coimbra, Portugal
email: {pmom, maryam, pnf}@dei.uc.pt

***Abstract*—In this paper, we investigate the problem of providing scalability (out and in) to Extraction, Transformation, Load (ETL) and Querying (Q) (ETL+Q) process of data warehouses. In general, data loading, transformation, and integration are heavy tasks that are performed only periodically, instead of row by row. Parallel architectures and mechanisms are able to optimize the ETL process by speeding up each part of the pipeline process as more performance is needed. We propose parallelization solutions for each part of the ETL+Q, which we integrate into a framework, that is, an approach that enables the automatic scalability and freshness of any data warehouse and ETL+Q process. Our results show that the proposed system algorithms can handle scalability to provide the desired processing speed in big-data and small-data scenarios.**

Keywords-Algorithms; architecture; Scalability; ETL; freshness; high-rate; performance; scale; parallel processing.

I. INTRODUCTION

ETL tools are special purpose software used to populate a data warehouse with up-to-date, clean records from one or more sources. The majority of current ETL tools organize such operations as a workflow. At the logical level, the E (extraction) can be considered as a capture of data flow from the sources, normally more than one with high-rate throughput. Then, we have T representing transformation and cleansing of data. This corresponds to modifying data so that it will conform to an analysis schema. The L (load) represents loading the data into the data warehouse, where the data is stored to be queried and analyzed. When implementing these types of systems, besides the necessity to create all these steps, the user is required to be aware of scalability requirements that the ETL+Q (ETL and queries) might raise for this specific scenario.

When defining the ETL+Q the user must have in mind the existence of data sources, where and how the data is extracted to be transformed (e.g., completed, cleaned, validated), the loading into the data warehouse, and finally the data warehouse schema, each of these steps requires different processing capacities, resources, and data treatment. However, in some applications scenarios, (e.g., near-real-time monitoring of telecom, energy distribution or stock market) ETL can be demanding in terms of performance. Most of the time because the data volume is too large and one single, extraction, transform, loading or querying node is not sufficient. Thus, more nodes must be added to extract the data and extraction policies from the sources must be created (e.g., round-robin OR on-demand). The other phases, transformation, and load must also be scaled.

After extraction, data must be re-directed and distributed across the available transformation nodes. Again since transformation involves heavy duty tasks (heavier than extraction), more than one node should be necessary to assure acceptable execution/transformation times.

After the data is transformed and ready to be loaded, the load period must be scheduled (e.g., every night, every hour, every minute) and load time controlled (e.g., maximum load time = 5 hours). This means that, between the transformation and load process, the data must be held somewhere.

Regarding the data warehouse, in some application scenarios the entire data will not fit into a single node, and if it fits, it will not be possible to execute queries within acceptable time ranges. Thus, more than one data warehouse node is necessary with a specific schema which allows distributing, replicate, and finally query the data within an acceptable time frame.

In this paper, we study how to provide ETL+Q scalability with ingress high-data-rate in big and small data warehouses. We propose a set of mechanisms and algorithms, to parallelize and scale each part of the entire ETL+Q process, which is included in an auto-scale (in and out) ETL+Q framework. This framework is based on time bounds for the parts of the ETL+Q and/or the global ETL process, automatically scaling, to assure the desired time bounds.

The presented results prove that the proposed monitoring mechanisms and detection algorithms are able to scale-out when necessary.

In Section II, we present relevant related work in the field. Section III, we describe the architecture of the proposed system. Section IV explains the main algorithms which allow to scale-out when necessary. Section V shows the experimental results obtained when testing the proposed system. Finally, Section VI concludes the paper and discusses future work.

II. RELATED WORK

Works in the area of ETL scheduling include efforts towards the optimization of the entire ETL workflow [6] and of individual operators in terms of algebraic optimization

 ISBN: 978-1-61208-457-2

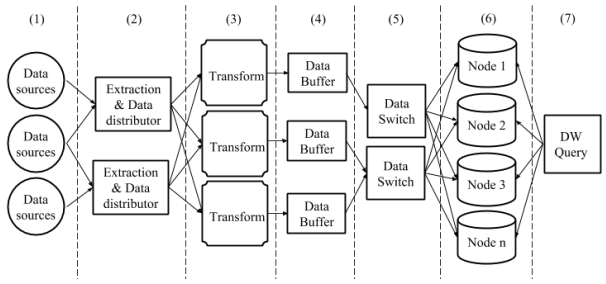

Figure 1. Total automatic ETL+Q scalability

(e.g., joins or data sort operations). The work [3] deals with the problem of scheduling ETL workflow at the data level and in particular scheduling protocols and software architecture for an ETL engine in order to minimize the execution time and the allocated memory needed for a given ETL workflow. The second aspect in ETL execution that the authors address is how to schedule flow execution at the operations level (blocking, non-parallelizable operations may exist in the flow) and how we can improve this with pipeline parallelization [2].

The work [4] focuses on finding approaches for the automatic code generation of ETL processes which is aligning the modeling of ETL processes in the data warehouse with Model Driven Architecture (MDA) by formally defining a set of QVT (Query, View, Transformation) transformations.

Related problems studied in the past include the scheduling of concurrent updates and queries in real-time warehousing and the scheduling of operators in data streams management systems. However, we argue that a fresher look is needed in the context of ETL technology. The issue is no longer the scalability cost/price, but rather the complexity it adds to the system. Previews presented recent works in the field do not address in detail how to scale each part of the ETL+Q and do not regard the automatic scalability to make ETL scalability easy and automatic. The authors focus on mechanisms to improve scheduling algorithms and optimizing workflow and memory usage. In our work, we assume that scalability in a number of machines and quantity of memory is not the issue. We focus on offering scalability for each part of the ETL pipeline process, without the nightmare of operators relocation and complex execution plans. Thus, in our work, we focus on scalability based on generic ETL process to provide the users desired performance with minimum complexity and implementations. In addition, we also support queries execution.

III. Architecture

In this section, we describe the main components of the proposed architecture for ETL+Q scalability.

Figure 1 depicts the main processes needed to support total ETL+Q scalability with specific time bounds.

(1) Represents the data sources from where data is extracted from the system.
(2) The data distributor(s) is responsible for forwarding or replicating the raw data to the transformer nodes. The distribution algorithm to be used is configured and enforced at this stage. The data distributors (2) should also be parallelizable if needed, for scalability reasons.
(3) In the transformation nodes the data is cleaned and transformed to be loaded into the data warehouse. This might involve data look-ups to in-memory or disk tables and further computation tasks. In Figure 1 the transformation (3) is parallelized for scalability reasons.
(4) The data buffer can be in memory, disk file (batch files) or both. In periodically configured time frames/periods, data is distributed across the data warehouse nodes.
(5) The data switches are responsible for distributing (pop/extract) data from the ”Data Buffers” and set it for load into the data warehouse, which can be a single-node or a parallel data warehouse depending on configured parameters (e.g., load time, query performance).
(6) The data warehouse can be in a single node, or parallelized by many nodes. If parallelized, the ”Data Switch” nodes will manage data placement according to configurations (e.g., replication and distribution). Each node of the data warehouse loads the data independently from batch files.
(7) Queries (7) are rewritten and submitted to the data warehouse nodes for computation. The results are then merged, computed and returned.

The main concepts, we propose are the individual ETL+Q scalability mechanisms of each part of the ETL+Q pipeline. By offering the solution to scale each part independently, we provide a solution to obtain configurable performance. Then, in future work based on user configuration parameters, a framework using these components, scales automatically the ETL+Q when necessary.

IV. Scaling algorithms

In this section, we describe the algorithms which allow the framework to scale-in and scale-out each part of the ETL and Query process. For each part that we design for later, automatic scale in and out we explain the scaling algorithms.

A. Extraction & data distributors - Scale out

Depending on the number of existing sources and data generation rate and size, the nodes that process the extraction of the data from the sources might need to scale. The addition of more ”extraction & data distributors” (2) depends on if the current number of nodes is being able to extract and process the data with the correct period and inside the maximum extraction time (without delays). For instance, if

the extraction period is specified as every 5 minutes and the extraction duration is 10 seconds, every 5 minutes the "Extraction & Data distributor" nodes cannot spend more than 10 seconds extracting data, if so, a scale-out is needed. By scaling out the extraction, nodes will have fewer data to extract/process and more concurrent extraction, leading to a performance improvement. Listing 1 pseudo-code describes the algorithm used to scale, independently of the used extraction method.

Listing 1. Extraction scalability

```
startTime = getCurrentTime();
source = requestSourceToExtractData();
size = requestSizeToExtract();
data = requestSourceExtraction(source, size);
endExtractionTime = getCurrentTime();
extractionTime.add(endExtractionTime-startTime);
for (all in extractionTime()){
    if (extractionTime > getMaxExtractionTime()){
        processScaleOut();
        break;
    }
}
for (all in extractionTime()){
    if (extractionTime > getExtractionPeriod()){
        processScaleOut();
        break;
    }
}
```

B. Extraction & data distributors - Scale in

To save resources when possible, the nodes that perform the data extraction from the sources can be set in standby or removed. This decision is made based on the last execution times. If previous execution times of at least two or more nodes are less than half of the configured maximum, one of the nodes is set on standby or removed, and the other one takes over. Listing 2 pseudo code describes the used algorithm.

Listing 2. Extraction and data distribution

```
lowLoadNodes = 0;
nodesID = null;
for (all nodes){
    if (processingTime() < getExtractionFrequency() / 2){
        lowLoadNodes++;
        nodesID.add(nod eID);
    }
    if (extractionTime() < getMaxExtractionTime() / 2){
        lowLoadNodes++;
        nodesID.add(nodeID);
    }
}
if (lowLoadNodes >= 2){
    setNodeToSandBy(nodesID.getFirst());
}
```

C. Transform - Scale-out

The transformation process is critical. If the transformation is running slow, data extraction at the refereed rate may not be possible, and information will not be available for loading and querying when necessary. The transformation step has an important queue, used to determine when to scale the transformation phase. If this queue reaches a limit size (by default 50%) then it is necessary to scale, because the actual transformer is not being able to process all data that is arriving. Another mechanism used to scale the transformation process is the user-configured maximum transformation execution time. If this time is exceeded then, the transformation must be scaled-out. Listing 3 pseudo code describes the used algorithm.

Listing 3. Transformation scale-out

```
limitSize = getLimitSize(); //by default 50%;
for (all nodes){
    currentQueueSize = queue.getSize();
    if (currentQueueSize > limitSize){
        addTrasnformerNode();
        break;
    }
    if (transformationTime > getMaxTransformTime()){
        addTrasnformerNode();
        break;
    }
}
```

D. Transform - Scale in

The size of all queues is analyzed periodically. If this size at a specific moment is less than half of the limit size for at least two nodes and the average transformation time of at least two nodes is half of the specified then, one of those nodes is set on standby or removed, and another one of the low load nodes takes over. Listing 4 pseudo code describes the used algorithm.

Listing 4. Transformation scale-in

```
limitSize = getLimitSize() / 2; //by default 25%;
maxTransformTime = getMaxTransformTime();
count = 0;
for (all nodes){
    currentQueueSize = node.queue.getSize();
    currentTransformTime = node.getAvgTransformTime();
    if (currentQueueSize <= limitSize &&
      currentTransformTime <= maxTransformTime/2){
        count++;
        if (count >= 2){
            setNodeToSandBy(nodeID);
            break;
        }
    }
}
```

E. Data buffer - Scale

The data buffer nodes scale-out based on the incoming memory queue size and the storage space available to hold data. Low data warehouse load frequency will require data buffers with storage space to hold the data until the scheduled load time. Thus, the data buffers scale dynamically as more storage space is necessary. Another scale-out situation is when the available incoming memory queue becomes above a certain threshold (by default 50%). This means that the data ingress rate is higher than the data swap speed, thus, nodes must scale-out in order to not lose data. By user request, the data buffers can also scale-in. In this case, the system will allow it if the data from any data buffer can be fitted inside other data buffer.

 ISBN: 978-1-61208-457-2

F. Data switch - Scale

These nodes scale based on configured data rate limits. If after a data load process occurs the average limit extraction data rate is equal or above a certain limit, then these nodes are set to scale. The data switches can also scale-in. In this case, the system will allow it if the average data rate from the previews load period is less than the maximum supported by each data switch.

G. Data Warehouse - Scale

The data warehouse scalability is detected after each load process. The loading process might include among other operations: destroy indexes, load data, update materialized view, and rebuild indexes. The data warehouse load process has a limit time to be executed every time it starts. If that limit time is exceeded then, the data warehouse must scale. Listing 5 pseudo code describes the used to scale the data warehouse when the load process occurs.

Listing 5. Data warehouse scale

```
startTime = getCurrentTime();
data = getData(size);
preLoadTask();
process(data);
posLoadTask();
startLoad();
endTime = getCurrentTime();
if (endTime - startTime > getMaxLoadTime(){
    dataWarehouseScaleOut();
}
```

The data warehouse scalability is not only based on the load & integration speed requirements, but also on the queries desired maximum execution time. The faster queries need to execute, more nodes will be necessary. Listing 6 pseudo code describes the used algorithm:

Listing 6. Data warehouse scale, integration speed

```
execute(queries);
avgTime = getQueryAverageExecutionTime();
desiredExecutionTime = getDesiredExecutionTime();
if (avgTime < deisredExecutionTime){
    dataWarehouseScaleOut();
}
```

Because it is a computationally expensive operation, when an alarm is raised (the data warehouse needs to scale) the data warehouse nodes scale-in and scales-out, can only be triggered by user request and iff the average query execution time and the average load time respect the conditions 1 and 2 (where *n* represents the number of nodes):

$$\frac{(n-1) \times avgQueryTime}{n} \leq desiredQueryTime \quad (1)$$

and

$$\frac{(n-1) \times avgLoadTime}{n} \leq maxLoadTime \quad (2)$$

Every time the data warehouse scales-out or scales in the data inside the nodes needs to be re-balanced. The default re-balance process to scale-out is based on the phases:

- Replicate dimension tables;
- Extract information from nodes;
- Load the extracted information into the new nodes.

V. Experimental setup and results

In this section, we test the ability of the proposed auto-scale framework to automatic scale-out the ETL process when more performance is necessary to provide the desired results. For the purpose of these tests, we simulated the launch of an ETL system only concerning a single server machine. In this setup the considered ETL process consists on converting the TPC-H [1] benchmark data generator into the Star Schema Benchmark (SSB) [5], and execute the SSB queries. For all tests, we used equal nodes, with intel i5 3.00GHz, 16 GB of RAM and 1TB of disk. By applying the described algorithms, we observed how it scales to provide the desired performance. In the next sections, we demonstrate how each part of the ETL and Query execution scales-out.

A. Data extraction nodes scalability

Considering that, we have data sources and extraction nodes to extract data. When the data flow is too high a single data node can not handle all ingress data. In this section, we study how the extraction nodes scale to handle different data rates. The extraction process uses an on-demand approach to extract data, where an ”automatic scaler” process orders the nodes to extract data from sources. There is a configured maximum allowed extraction time and a extraction frequency, represented by the Equations 3 and 4. If any of them is not respected the system is set to scale-out.

$$\max_{extractionTime} < \max_{desiredExtractionTime} \quad (3)$$

$$\max_{extractionTime} < ExtractionFrequency \quad (4)$$

Figure 2 shows: In the left Y axis is the average extraction time in seconds; In the right Y axis is the number of nodes; The X axis is the data-rate; Black line represents the extraction time; Grey line represents the number of nodes; The maximum allowed extraction time was set to 1 second maximum extraction time, with periodic extraction of 5 seconds.

As we can conclude from Figure 2 experimental results, every time the maximum allowed extraction time has exceeded the system requested an additional extraction node to improve the extraction performance.

B. Transformation scalability

During the ETL process after data extraction, it is set for the transformation. In our tests the transformation consists

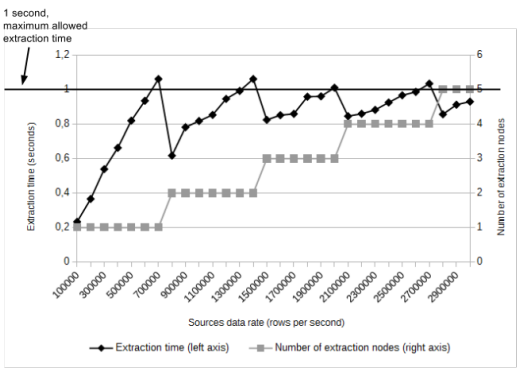

Figure 2. Extraction scalability

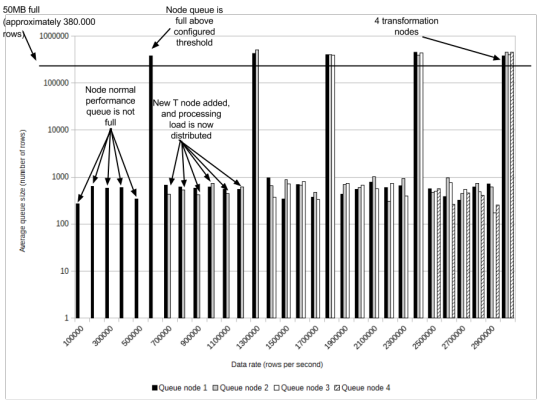

Figure 3. Automatic transformation scalability. For different data rates, for 60 minutes processing per data rate. Transformation threshold set to 50MB, approximately 380.000 rows

on converting the TPC-H dataset into SSB format. Because this process is computationally heavy it is often required to scale the transformation nodes. Each transformation node has an entrance queue for ingress data and an automatic scale monitors all queues. Once it detects that a queue is full above a certain (configured) threshold it starts the scaling process, this means that $Rate_{extract} \geq Rate_{transform}$.

Figure 3 shows: Y axis, average queue size in number of rows; X axis, the data rate in rows per second; Each plotted bar represents a node queue size, up to 4 nodes; The limit queue size to trigger the scale mechanisms was set at 50MB, approximately 380.000 rows; The maximum transformation time for each row was set to 1 second. Each measure is the average queue size of 60 seconds run.

During the experimental tests, as depicted in Figure 3, the maximum allowed transformation time was never exceeded. However, the queues size increased while increasing the data rate, allowing to show that the proposed approach is efficient to scale the transformation nodes.

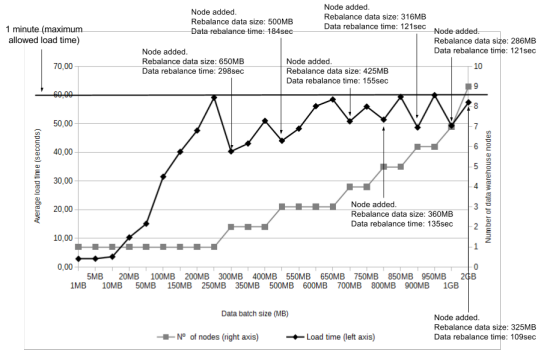

Figure 4. Data warehouse load scalability

C. Data Buffer nodes

These nodes hold the transformed data until it is loaded into the data warehouse. During all our tests, we used a single machine with 16GB memory and 1TB disk, all available to be used. If the available storage space becomes full then, the automatic scale sets the system to scale the Data Buffer node (add one more node). However, during our tests, we never had the necessity to do so since all transformed data could fit into memory until the next load (into the data warehouse) period.

D. Data warehouse scalability

In this section, we test the data warehouse scalability, which can be triggered or by the load process (because it is taking too long), or because of the queries time (they are taking more time than the desired execution time). If the maximum configured load time is exceeded, the data warehouse is set to scale.

Experimental results from Figure 4 show: Left Y axis, average load time in seconds; Right Y-axis, number of data warehouse nodes; X axis, data batch size in MB; The maximum allowed load time, set to 60 seconds; Each time a data warehouse (scales) node is added, we show the data size that was moved into the new node and the required time in seconds (re-balance time).

Based on our experimental results, we conclude that the proposed method to scale the data warehouse when the bottleneck is related to the load time is efficient, improving the overall load performance. Note that, every time a new node was added the data warehouse required to be re-balanced (data distributed by the nodes evenly). This process requires 3 steps, first extract (in parallel) the data from the existent nodes, second load the data into the new node, third load the new data (distributed and parallel) in batch and check if the load time is lower than the maximum allowed load time.

E. Query scalability

When running queries, if the maximum desired query execution time (i.e. configured parameter) is exceeded then,

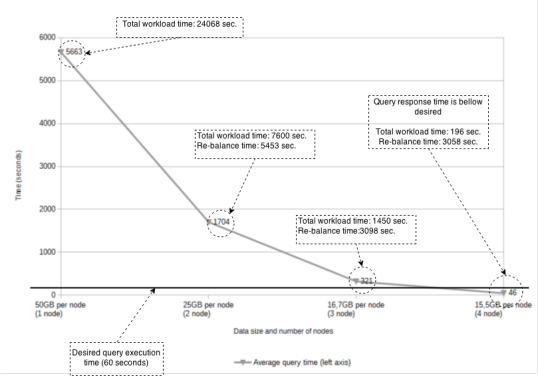

Figure 5. Data warehouse scalability, workload 1

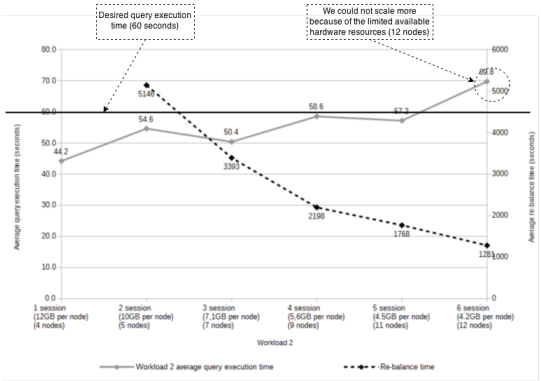

Figure 6. Data warehouse scalability, workload 2

the data warehouse is set to scale in order to offer more query execution performance. The following workloads were considered to test the proposed system:

- Workload 1;
  - 50GB total size;
  - Execute Q1.1, Q2.1, Q3.1, Q4.1 randomly chosen;
  - Desired execution time per query: 5 minutes (300 seconds).
- Workload 2 (as workload 1 but with more sessions);
  - **1 to 8 sessions;**

Workload 1 studies how the proposed mechanisms scale the data warehouse when running queries. Workload 2 studies the scalability of the system when running queries and the number of simultaneous sessions (e.g., the number of simultaneous users) increases. Both workloads with the objective to deliver the configured execution time per query (300 seconds).

F. Query scalability - Workload 1

Figure 5 shows: The experimental results for workload 1; Y axis, average execution time in seconds using a logarithmic scale; X axis the data size per node and the current number of nodes; The horizontal line over 300 seconds represents the desired query execution time;

The results from Figure 5, show that the proposed system can detect and scale the data warehouse nodes until the average query execution time is the desired.

G. Query scalability - Workload 2

Figure 6 shows: The experimental results for workload 2; Y axis, average execution time in seconds using a logarithmic scale; X axis the number of sessions, the data size per node and the number of nodes; The horizontal line over 300 seconds represents the desired query execution time; The last result does not respect the desired execution time because of the limited resources for our tests, 12 nodes.

In Figure 6, we show that while the number of simultaneous sessions increases the system scales the number of nodes in order to provide more performance, thus, the query average execution time follows the configured parameters. As our experimental results show the proposed system scales efficiently to provide the desired performance.

VI. Conclusions & Future work

In this work, we propose mechanisms and algorithms to achieve automatic scalability for complex ETL+Q, offering the possibility to the users to think solely in the conceptual ETL+Q models and implementations for a single server. The tests demonstrate that the proposed techniques are able to scale-out. Future work will investigate an auto-scale framework for scale-out and scale in any ETL+Q and, at the same time, providing data freshness and support for near-real-time data stream processing.

References

[1] T. P. P. Council. Tpc-h benchmark specification. *Published at http://www. tcp. org/hspec. html*, 2008.

[2] R. Halasipuram, P. M. Deshpande, and S. Padmanabhan. Determining essential statistics for cost based optimization of an etl workflow. In *EDBT*, pages 307–318, 2014.

[3] A. Karagiannis, P. Vassiliadis, and A. Simitsis. Scheduling strategies for efficient etl execution. *Information Systems*, 38(6):927–945, 2013.

[4] L. Muñoz, J.-N. Mazón, and J. Trujillo. Automatic generation of etl processes from conceptual models. In *Proceedings of the ACM twelfth international workshop on Data warehousing and OLAP*, pages 33–40. ACM, 2009.

[5] P. O'Neil, E. O'Neil, X. Chen, and S. Revilak. The star schema benchmark and augmented fact table indexing. In *Performance Evaluation and Benchmarking*, pages 237–252. Springer, 2009.

[6] A. Simitsis, K. Wilkinson, U. Dayal, and M. Castellanos. Optimizing etl workflows for fault-tolerance. In *Data Engineering (ICDE), 2010 IEEE 26th International Conference on*, pages 385–396. IEEE, 2010.

 ISBN: 978-1-61208-457-2