Efficient ETL+Q for Automatic Scalability in Big or Small Data Scenarios

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 to data Extraction, Transformation, Load and Querying (ETL+Q) process of data warehouses. In general, data loading, transformation and integration are heavy tasks that are performed only periodically. 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 an approach to enable the automatic scalability and freshness of any data warehouse and ETL+Q process, suitable for smallData and bigData business. A general framework for testing and implementing the system was developed to provide solutions for each part of the ETL+Q automatic scalability. The results show that the proposed system is capable of handling scalability to provide the desired processing speed for both near-real-time results and offline ETL+Q processing.**

***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 (Extract) can be considered as a capture of data-flow from the sources with more than one high-rate throughput. T (Transform) represents transforming and cleansing data in order to be distributed and replicated across many processes and ultimately, L(Load) convey by loading the data into data warehouses to be stored and queried. For implementing these type of systems besides knowing all of these steps, the acknowledge of user regarding the scalability issues is essential, which the ETL+Q (queries) might be introduced.

When defining the ETL+Q the user must consider the existence of data sources, where and how the data is extracted to be transformed, loading into the data warehouse and finally the data warehouse schema; each of these steps requires different processing capacity, resources and data treatment. Moreover, the ETL is never so linear and it is more complex than it seems. Most often the data volume is too large and one single extraction node is not sufficient. Thus, more nodes must be added to extract the data and extraction policies from the sources such as round-robin or on-demand are necessary.

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 present to assure acceptable execution/transformation times. Consequently, once more new data distribution policies must be added. After the data transformed and ready to be load, the load period time and a load time control must be scheduled. Which means that the data have to be held between the transformation and loading process in some buffer. Eventually, regarding the data warehouse schema, 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 to distribute, replicate, and query the data within an acceptable time frame.

In this paper, we study how to provide parallel ETL+Q scalability with ingress high-data-rate in big data 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 later will be included in an auto-scale (in and out) ETL+Q framework. The presented results prove that the proposed mechanisms are able to scale when necessary.

Section II approaches the related work in the field. Section III describes the proposed architecture. Section IV describes the experimental setup and obtained results. Finally, Section V concludes the presented work and introduces some future research lines.

II. RELATED WORK

Works in the area of ETL scheduling includes efforts towards the optimization of the entire ETL workflows [6] and of individual operators in terms of algebraic optimization, e.g., joins or data sort operations. However, many works focus on complex optimization details that only apply to very specific cases. Munoz et al. [3] focus on finding approaches for the automatic code generation of ETL processes which is aligning the modeling of ETL processes in data warehouse with Model Driven Architecture (MDA) by formally defining a set of Query, View, Transformation (QVT) transformations. ETLMR [2] is an academic tool which builds the ETL processes on top of Map-Reduce to parallelize the ETL operation on commodity computers. ETLMR does not have its own data storage (note that the

 ISBN: 978-1-61208-438-1

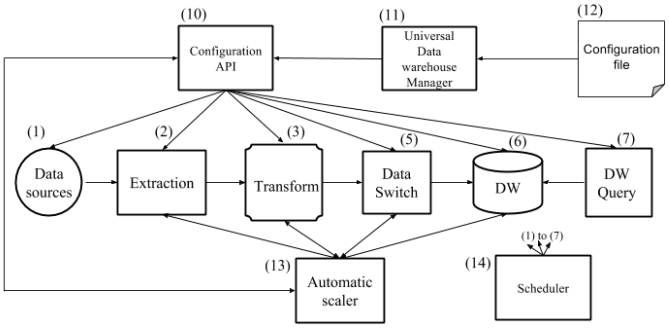

Figure 1. Automatic ETL+Q scalability

offline dimension store is only for speedup purpose), but is an ETL tool suitable for processing large scale data in parallel. ETLMR provides a configuration file to declare dimensions, facts, User Defined Functions (UDFs), and other run-time parameters. ETLMR toll has the same problem as the MapReduce architectures, too much hardware resources are required to guaranty basic performance.

In [5], the authors consider the problem of data flow partitioning for achieving real-time ETL. The approach makes choices based on a variety of trade-offs, such as freshness, recoverability and fault-tolerance, by considering various techniques. In this approach, partitioning can be based on round-robin (RR), hash (HS), range (RG), random, modulus, copy, and others [7].

In [1] the authors describe Liquid, a data integration stack that provides low latency data access to support near real-time in addition to batch applications.

There is a vast related work in ETL field. Although main 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. We focus on offering scalability for each part of the ETL pipeline process, without the nightmare of operators relocation and complex execution plans. Our main focus is automatic scalability 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 shows the main components to achieve automatic scalability.

- All components from (1) to (7) are part of the Extract, Transform, Load and query (ETL+Q) process. All can auto scale automatically when more performance is necessary.
- The ”Automatic Scaler” (13), is the node responsible for performance monitoring and scaling the system when is necessary.
- The ”Configuration file” (12) represents the location where all user configurations are defined by the user.
- The ”Universal Data Warehouse Manager” (11), based on the configurations provided by the user and using the available ”Configurations API” (10), sets the system to perform according with the desired parameters and algorithms. The ”Universal Data Warehouse Manager” (11), also sets the configuration parameters for automatic scalability at (13) and the policies to be applied by the ”Scheduler” (14).
- The automatic scaler module (13), based on time bounds configurations and limit amounts of resources to use (mainly memory) scales the ETL pipeline modules.
- The ”Configuration API” (10), is an access interface which allows to configure each part of the proposed Universal Data Warehouse architecture, automatically or manually by the user.
- Finally, the ”Scheduler” (14), is responsible for applying the data transfer policies between components (e.g., control the on-demand data transfers).

All these components when set to interact together are able to provide automatic scalability to the ETL and to the data warehouses processes without the need for the user to concern about its scalability or management.

IV. Experimental setup and results

In this section, we describe the experimental setup, and experimental results to show that the proposed system, AScale, is able to scale and load balance data in small and big data scenarios for near real-time and offline ETL+Q.

The experimental tests were performed using 30 computers, denominated as nodes, with the following characteristics: Processor Intel Core i5-5300U Processor (3M Cache, up to 3.40 GHz); Memory 16GB DDR3; Disk: western digital 1TB 7500rpm; Ethernet connection 1Gbit/sec; Connection switch: SMC SMCOST16, 16 Ethernet ports, 1Gbit/sec; Windows 7 enterprise edition 64 bits; Java JDK 8; Netbeans 8.0.2; Oracle Database 11g Release 1 for Microsoft Windows (X64) - used in each data warehouse nodes; PostgreSQL 9.4 - used for look ups during the transformation process; TPC-H benchmark - representing the operational log data used at the extraction nodes. This is possible since TPC-H data is still normalized; SSB (star schema benchmark) benchmark - representing the data warehouse. The SSB is the star-schema representation of TPC-H data. Data transformations consist loading from the data sources the ”lineitem” and ”order” TPC-H data logs and besides the transformation applied to achieve the SSB benchmark

 ISBN: 978-1-61208-438-1

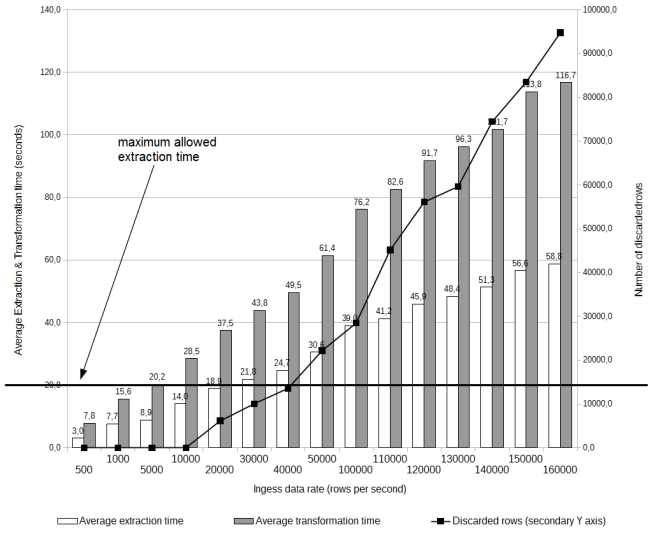

Figure 2. Extract and transform without automatic scalability

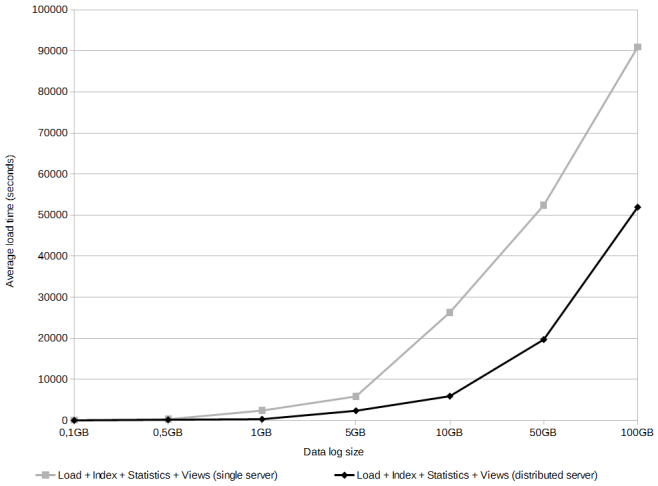

Figure 3. Loading data, one server vs two servers

star schema [4] we added some data quality verification and cleansing.

A. *Performance limitations without automatic scalability*

In this Section, we test both ETL and data warehouse scalability needs when the entire ETL process is deployed without automatic scalability options. The system is stressed with increasing data-rates until it is unable to handle the ETL and query processing in reasonable time. Automatic scalability which we evaluate in following sections, is designed to handle this problem.

The following deployment is considered: One machine to extract, transform data and store the data warehouse; extraction frequency is set to perform every 30 seconds; desired maximum allowed extraction time, 20 seconds; data load is performed in offline periods.

Based on this scenario, we show the limit situation in which performance degrades significantly, justifying the need to scale the ETL (i.e., parts of it) or data warehouse.

Extraction & transformation: Considering only extraction and transformation, using a single node, Figure 2 shows: in the left Y axis is represented the average extraction and transformation time in seconds; in the right Y axis is represented the number of discarded rows (data that was not extracted and not transformed); in the X axis we show the data-rate in rows per-second; white bars represent extraction time; gray bars represent the transformation time; lines represent the average number of discarded rows (corresponding values in the right axes). For this experiment, we generated log data (data to be extracted) at a rate λ per second. Increasing values of λ were tested and the results are shown in Figure 2.

Extraction is performed every 30 seconds. This means that in 30 seconds there is 30x more data to extract. Extraction must be done in 20 seconds maximum. As the data-rate increases, a single node is unable to handle so much data. At a data-rate of 20.000 rows per second, buffer queues become full and data starts being discarded at sources because the extraction time is too slow. The transformation process is slower than transformation, requiring more resources to perform at the same speed as the extraction.

Loading the data warehouse: Figure 3 shows the load time as the size of the logs is increased. It also compares the time taken with single single node versus two nodes. All times were obtained with the following load method: destroy all indexes and views, load data, create indexes, update statistics and update views; data was distributed by replicating and partition the tables. Differences are noticeable when loading more than 10GB. When adding two data warehouse nodes, performance improves and the load time becomes almost less than half.

- The Y axis represents the average load time in seconds and the X axis represents the loaded data size in GB.
- The black line represents two servers and the grey line represents one server.

Query execution: Figure 4 shows the average query execution time for a set of tested workload (using the SSB benchmark queries): workload 1, 10 sessions, 5 Queries (Q1.1, Q1.2, Q2.1, Q3.1, Q4.1); workload 2, 50 sessions, 5 Queries (Q1.1, Q1.2, Q2.1, Q3.1, Q4.1); workload 3, 10 sessions, 13 Queries (All); workload 4, 50 sessions, 13 Queries (All); for all workloads, queries were executed in a random order; the desired maximum query execution time was set to 60 seconds.

The Y axis shows the average execution time in seconds. The X axis shows the data size in GB. Each bar represents the average execution time per query fro each workload. Note that, Y axis scale is logarithmic for better results representation.

 ISBN: 978-1-61208-438-1

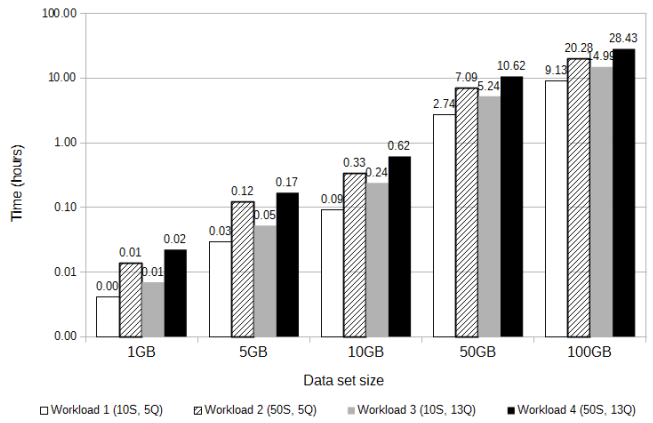

Figure 4. Average query time for different data sizes and number of sessions

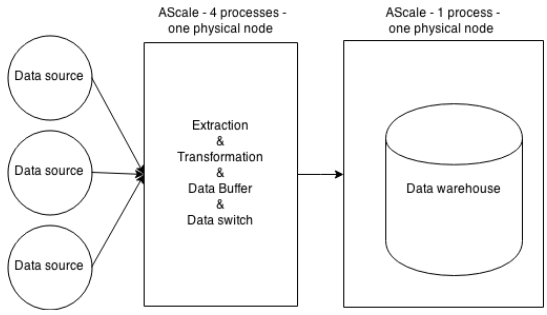

Figure 5. AScale for simple scenarios

Depending on the data size, number of queries and number of simultaneous sessions (e.g., number of simultaneous users), execution time can vary from a few seconds to a very significant number of hours or days, especially when considering large data sizes and simultaneous sessions or both. In these results, and referring to 10GB and 50GB, we see that an increase of 5x of the data size resulted in an increase of approximately 20x in response time. An increase in the number of 5x resulted in an increase of approximately 2x in query response time.

B. Typical data warehouse scenarios

In this section we evaluate AScale in a scenario where because of log sizes and limited resources, data load takes too long to perform without scaling.

We start with only two nodes (two physical machines), one for handling extraction and transformation, the other to hold the data warehouse as shown in Figure 5. AScale is setup to monitor the system and scale when needed.

Data is extracted from sources, transformed and loaded only during a predefined period (e.g., night), to be available for analysis the next day. The maximum extraction, transformation and load time, all together cannot take longer than 9 hours (e.g., from 0am until 9am). AScale was

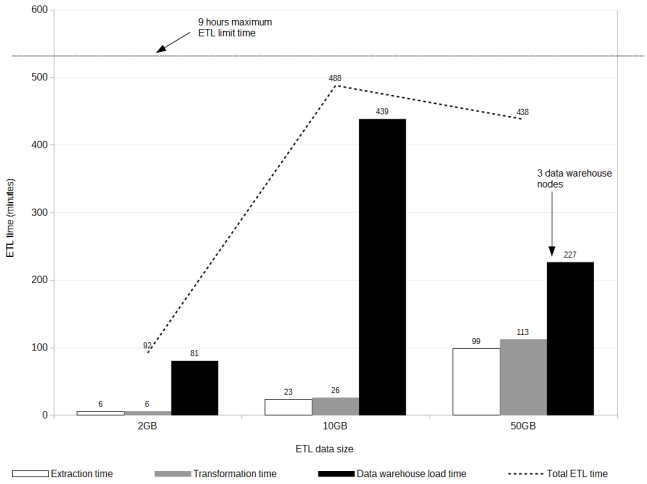

Figure 6. AScale, 9 hours limit ETL time

configured with an extraction frequency of every 24 hours and a maximum duration of 4 hours, a transformation queue with a limit size of 10GB and data warehouse loads were configured for every 24 hours, with a maximum duration of 9 hours. Note that, the entire ETL process was set for a maximum duration of 9 hours.

The experimental results form Figure 6 show the total AScale ETL time using two nodes (two physical machines), one for extract, transform, data buffer and data switch, other for the data warehouse. Up to 10GB the ETL process can be handled within the desired time windows. However, when increasing to 50GB, 9 hours are no longer enough to perform the full ETL process. In this situation, the data warehouse load process (load, update indexes, update views) using one node (average load time 873 minutes) and two nodes (average load time 483 minutes) exceeds the desired time window. When scaled to 3 nodes, by adding one data warehouse node, the ETL process returns to the desired time bound.

The extraction and transformation process were never scaled, since they were able to perform within the desired time, the same for the data buffer and data switch that were able to handle all data within defined bounds.

C. ETL offline scalability with huge data sizes

In this section we create an experimental setup to stress AScale under extreme data rate conditions. The objective is to test scaling each part of the pipeline.

For this experiment we did the following configured: E (extraction) was set to perform every 1 hour with 30 minutes maximum extraction time, T (transformation) queue maximum size was configured to 500MB, and L (load) frequency to every 24 hours with a maximum duration of 5 hours.

 ISBN: 978-1-61208-438-1

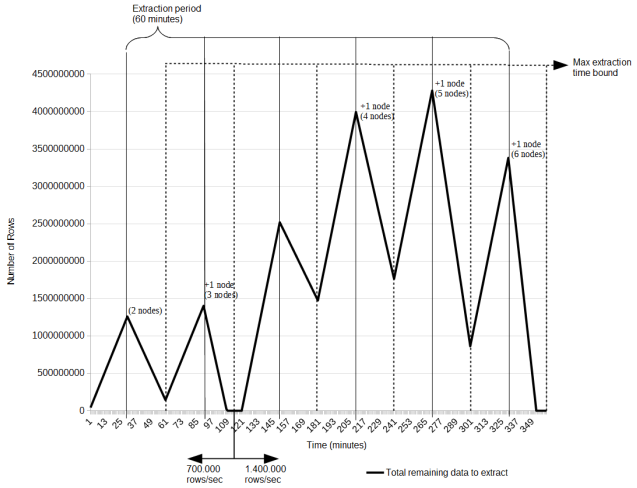

Figure 7. Extraction (60 minutes frequency and 30 minutes maximum extraction time)

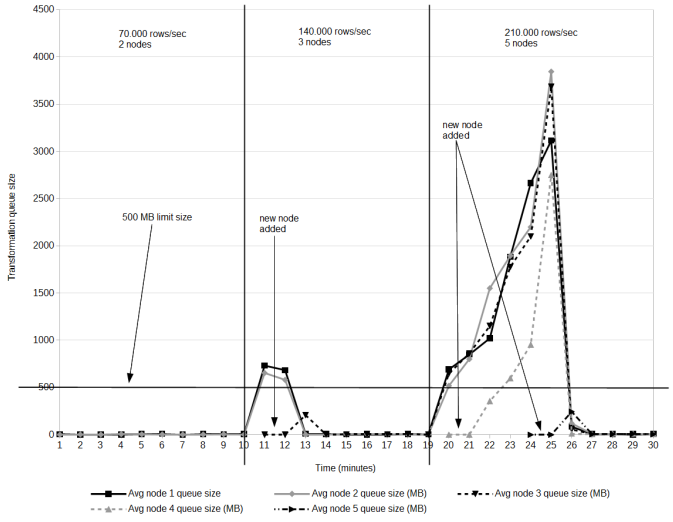

Figure 8. Offline, transformation scale-out

Extraction: Figure 7 shows the AScale extraction process when using an extraction frequency of 60 seconds and 30 seconds for the maximum extraction time.

In Figure 7, we show the followings: the left Y axis the number of rows, the X axis is represented the time in minutes and the black line represents the total number or rows left to be extracted at each extraction period. By analyzing the results from Figure 7 we conclude that the extraction process is able to scale efficiently when more computational power is necessary. However, if the data rate increases very fast in a small time window AScale requires additional extraction cycles to restore the normal extraction frequency.

Transformation: In Figure 8 are shown the transformation scale-out tests based on nodes ingress data queue size.

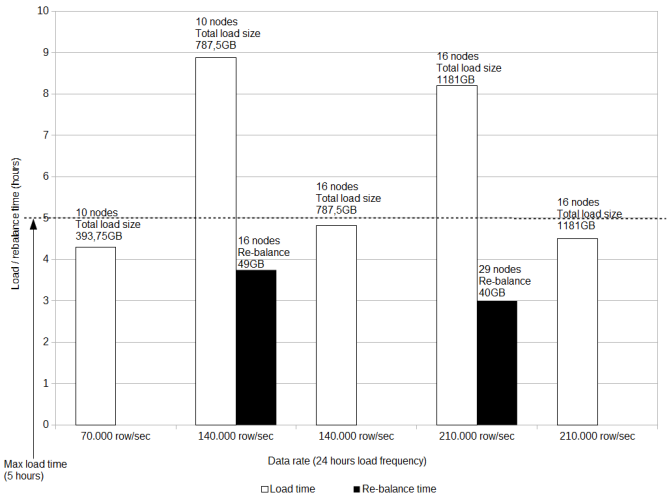

Figure 9. Offline, load scale-out

Every time a queue fills-up until the maximum configured size AScale automatically scale-out. This monitoring process allows to scale-out very fast, even if the data rate increases suddenly. Each scale out took only an average of 2 minutes, referring to the copy and replication of the staging area. Experimental results show that AScale transformation can be scaled-out more than one node in a very short time frame.

Load: AScale load process is done at the end of each load cycle that did not respected the maximum load time. The number of nodes to add is calculated linearly based on previews load time. For instance, if load time using 10 nodes was 9 hours. To load in 5 hours we need x nodes, estimated in equation 1.

$$\frac{loadTime}{targetTime} \times n \quad (1)$$

Where "loadTime" represents the last load time, "targetTime", represents the desired load time and "n" represents the current number of nodes.

Figure 9 shows the data warehouse nodes scalability time and data (re)balance time. We conclude that the data warehouse nodes can be scaled efficiently in a relatively short period of time given the large amounts of data being considered.

D. Near-Real-time DW scalability and Freshness

In this section we assess the scale-out and scale-in abilities of the proposed framework in near-real-time scenarios requiring data to be always updated and available to be queried (i.e., data freshness).

The near-real-time scenario was set-up with: E (extraction) and L (load) were set to perform every 2 seconds; T (transformation) was configured with a maximum queue size of 500MB; the load process was made in batches of

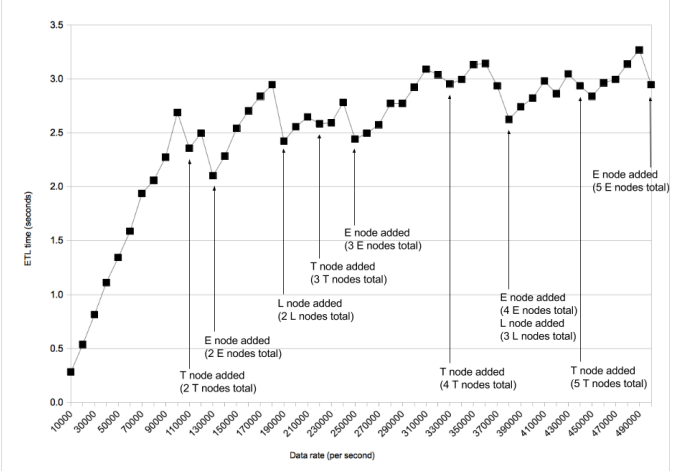

Figure 10. Near-real-time, full ETL system scale-out

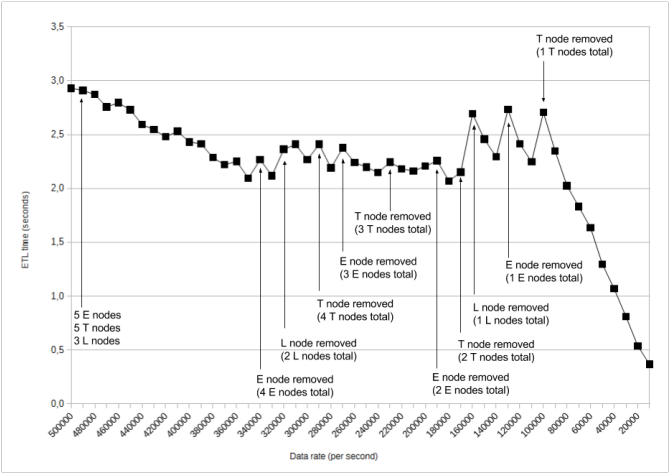

Figure 11. Near-real-time, full ETL system scale-in

100MB maximum size. The ETL process is allowed to take 3 seconds.

Figures 10 and 11 show AScale, scaling-out and scaling-in automatically, respectively, to deliver the configured near-real-time ETL time bounds, while the data rate increases/decreases. The system objective was set to deliver the ETL process in 3 seconds. The charts show the scale-out and scale-in of each part of the AScale, obtained by adding and removing nodes when necessary. A total of 7 data sources were used/removed gradually, each one delivering a maximum average of 70.000 rows/sec. AScale used a total of 12 nodes to deliver the configured time bounds.

Near-real-time scale-out results in Figure 10 show that, as the data-rate increases and parts of the ETL pipeline become overloaded, by using all proposed monitoring mechanisms in each part of the AScale framework, each individual module scales to offer more performance where and when necessary.

Near-real-time scale-in results in Figure 11 show the instants when the current number of nodes is no longer necessary to ensure the desired performance, leading to some nodes removal (i.e., set as ready nodes in stand-by, to be used in other parts).

V. Conclusions & Future work

In this work we proposed mechanisms 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.

Tests demonstrate that the proposed techniques are able to scale-out and scale-in when necessary to assure the necessary efficiency. Future work includes real-time event processing integration oriented to alarm and fraud detection. Other future work included making an visual drag and drop interface, improve monitoring and scale decision algorithms, and finally provide usability comparisons with other academic tools. A beta version of the framework is being prepared for public release.

References

[1] N. Ferreira, P. Martins, and P. Furtado. Near real-time with traditional data warehouse architectures: factors and how-to. In *Proceedings of the 17th International Database Engineering & Applications Symposium*, pages 68–75. ACM, 2013.

[2] X. Liu. *Data warehousing technologies for large-scale and right-time data*. PhD thesis, dissertation, Faculty of Engineering and Science at Aalborg University, Denmark, 2012.

[3] 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.

[4] P. E. O'Neil, E. J. O'Neil, and X. Chen. The star schema benchmark (ssb). *Pat*, 2007.

[5] A. Simitsis, C. Gupta, S. Wang, and U. Dayal. Partitioning real-time etl workflows, 2010.

[6] A. Simitsis, P. Vassiliadis, and T. Sellis. Optimizing etl processes in data warehouses. In *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on*, pages 564–575. IEEE, 2005.

[7] P. Vassiliadis and A. Simitsis. Near real time etl. In *New Trends in Data Warehousing and Data Analysis*, pages 1–31. Springer, 2009.

 ISBN: 978-1-61208-438-1