Data warehouse ETL+Q auto-scale framework

Pedro Martins*, Maryam Abbasi and Pedro Furtado

Department of Informatics,
Faculty of Sciences and Technology,
University of Coimbra, Portugal
Email: pmom@dei.uc.pt
Email: maryam@dei.uc.pt
Email: pnf@dei.uc.pt
*Corresponding author

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 optimise the ETL process by speeding-up each part of the pipeline process as more performance is needed. We propose parallelisation solutions, called AScale, for each part of the ETL+Q, 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 scalablity to provide the desired processing speed.

Keywords: scalability; freshness; high-rate; performance; parallel processing; distributed systems; database; load-balance; extraction transformation load; ETL; algorithm.

Reference to this paper should be made as follows: Martins, P., Abbasi, M. and Furtado, P. (2016) 'Data warehouse ETL+Q auto-scale framework', *Int. J. Business Intelligence and Systems Engineering*, Vol. 1, No. 1, pp.49–76.

Biographical notes: Pedro Martins is a full time researcher at the University of Coimbra Portugal. His PhD degree main research topics are in the fields of distributed computation, big data warehouses, data freshness and continuous results processing.

Maryam Abbasi is a researcher at Evolutionary and Complex Systems group (ECOS) in CISUC, Coimbra, Portugal. She is currently completing her PhD studies in the Department of Informatics Engineering from University of Coimbra. Her research projects focus on utilising the multiobjective optimisation in bioinformatics problems such as sequence alignment.

Pedro Furtado is a Professor at the University of Coimbra UC, Portugal, where he teaches courses in both Computer and Biomedical Engineering. His main research interests are on performance and scalability qualities of systems. He applied these qualities in data warehousing, bigdata, analytics, data mining, cloud, and IoT.

1 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 organise 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 analysed. 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 times because the data volume is too large and one single, extraction, transform, loading or querying node are 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 (RR) 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 a load time control (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 to distribute, 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 data warehouses. We propose a set of mechanisms and algorithms, to parallelise 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-out when necessary.

2 Related work

Our framework optimises ETL by automatically scaling each part of the processing pipeline.

These are some previous works related to optimising and scaling ETL.

To increase the efficiency of the ETL process, Simitsis et al. (2005b, 2005a) propose searching methods based on heuristic algorithms that minimise the ETL execution cost, by modelling the problem as a space search graph to decide which execution is more efficient. Graphs are created by the decomposition of relational algebra operators. Heuristics are created based on the number of times each state is visited.

Albrecht and Naumann (2009) studies how to manage large ETL processes by implementing a set of basic management operators, such as 'MATCH', 'MERGE', 'INVERT', 'SEARCH', 'DEPLOY'. The framework is web-based. The user creates the ETL flow using drag-and-drop with the available filters, then the framework determines the best execution order for the ETL using a set of optimisation algorithms.

Karagiannis et al. (2013) discusses the problem of scheduling the execution of ETL activities (a.k.a. transformations, tasks, operations), with the goal of minimising ETL execution time and allocated memory. The paper investigates the effects of four scheduling policies (RR, Minimum Cost Prediction, Minimum Memory Prediction and Mixed Policies) on different flow structures and configurations. It shows that the use of different scheduling policies may improve ETL performance in terms of memory consumption and execution time.

Wang and Guo (2011) propose a distributed ETL engine architecture based on multi-agent systems (MAS) data partitioning technology. They also investigate methods of partitioning the massive data streams in both horizontal and vertical ways. The system partitions workflows into multiple sub-workflows for parallel execution in agents, also adding a splitter node to distribute work. Each sub-workflow is executed by an agent, so that multiple agents could work together to complete the collaborative work. At the end, another extra node, the merger, will merge all results.

Liu et al. (2012) describe an extract-transform-load programming framework using Map-Reduce to achieve scalability. Data sources and target dimensions need to be configured and deployed. The framework has built-in support for star schemas and snowflakes. Users have to implement the parallel ETL programs using the framework constructors. They use pygrametl (Thomsen and Bach Pedersen, 2009), a Python-based framework for easy ETL programming. The flow consists of two phases, dimension processing and fact processing. Data is read from sources (files) on a distributed file system (DFS), transformed and processed into dimension values and facts by the framework instances, which materialise the data into the DW. The framework requires users to declare (code) target tables and transformation functions. Then, it uses a master/worker architecture (one master, many workers), each worker running jobs in parallel. The master distributes data, schedules tasks, and monitors the workers.

Liu (2012) proposes a tool to build the ETL processes on top of Map-Reduce to parallelise the ETL operation on commodity computers. ETLMR contains a number of novel contributions. It supports high-level ETL-specific dimensional constructs for processing both star-schemas and snowflake-schemas, and data-intensive dimensions. Due to its use of Map-Reduce, it can automatically scale to more nodes (without modifications to the ETL flow) while at the same time also providing automatic data

synchronisation across nodes (even for complex dimension structures like snowflakes). Apart from scalability, Map-Reduce also give ETLMR a high degree of fault-tolerance. ETLMR does not have its own data storage (note that the offline dimension store is only for speedup purposes), 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.

Simitsis et al. (2010) 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 RR, hash (HS), range (RG), random, modulus, copy, and others (Vassiliadis and Simitsis, 2009).

Pentaho data integration (PDI) (Pentaho, 2014), is an ETL (graphical) design tool and provides Hadoop support. It consists of a data integration (ETL) engine, and GUI applications that allow the user to define data integration jobs and transformations. It supports deployment on single node computers as well as on a cloud, or cluster. Internally, it allows connection/integration with other systems using for instance sockets, web-services (e.g., SOAP, XML).

2.1 Analysis

The mentioned works focus optimisation of each individual ETL process by optimising data access, reordering operators execution and managing the available computational resources as well as possible. Some do not use parallelism, which limits capability to scale. Such approaches can easily be used together our proposed framework. To guarantee adequate ETL and query processing services in demanding environments, it is essential for the systems to scale automatically. A direct scalability approach would be to use Map-Reduce to implement the entire ETL process as same of the related works propose. However, the Map-Reduce model does not offer:

- automatic real-time performance monitoring and scaling mechanisms, new nodes must be added manually
- there is more network traffic in consequence of using a DFS and Map-Reduce paradigm
- the entire ETL process must be coded in Map-Reduce programming model, adding more complexity and potential performance limitations
- efficiency depends on implementation method of used operators, specially when data exchange is required.

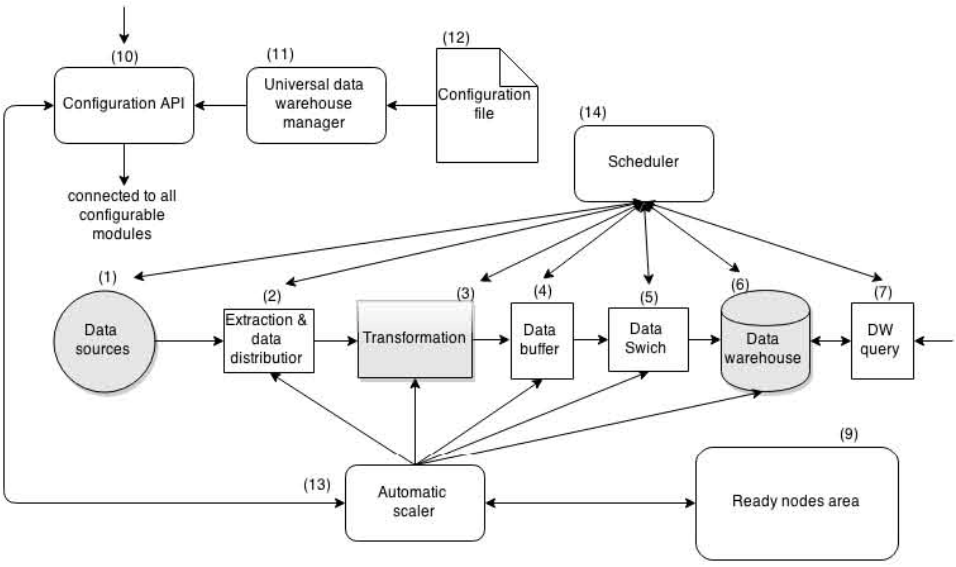
The proposed framework offers better usability and performance by:

- providing an automatic scaling mechanism based on monitoring
- allowing to scale each part of the ETL+Q process independently
- allowing to define the ETL using any programming language as long as a connector for the framework is provided
- data (dimension tables or all tables) can be replicated across the data warehouse nodes, avoiding high amounts of data exchanges over the network to merge results.

3 Architecture

In this section, we describe the main components of the propose architecture, AScale. Figure 1 shows its main components:

Figure 1 AScale pipeline architecture for automatic scalability



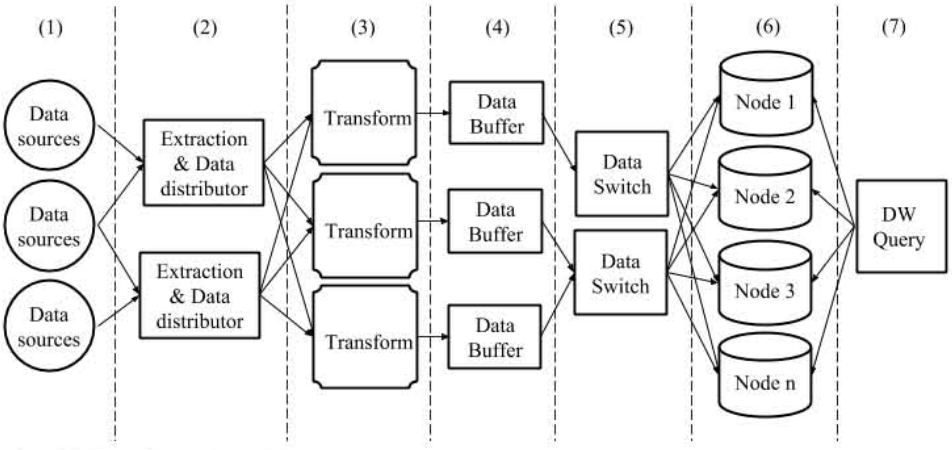
- Components (1) to (7), except (5) are the Extract, Transform, Load and Query (ETL+Q) pipeline.
- The 'Automatic Scaler' (13), is the component responsible for performance monitoring and scaling the system when necessary.
- The 'Configuration file' (12) represents the location where all user configurations are saved.
- The 'Universal Data Warehouse Manager' (11), uses the configurations provided by the user and the available 'Configurations API' (10) to set the system to perform according with the desired parameters and selected 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 'Configuration API' (10), is an access interface which allows to configure each part of the proposed Universal Data Warehouse architecture, automatically by (11) or manually by the user.
- The 'Scheduler' (14), is responsible for applying the data transfer policies between components.

- The ‘Ready nodes area’ (9) represent nodes that are not being used. These nodes can be added to any part (2) to (6) of the system to scale-out, improving performance where needed, or removed to scale-in, saving resources that can be used in other places.

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

Instead of programming the entire ETL pipeline, the user can focus only in programming the transformations and data warehouse schema (Figure 1 highlighted in grey colour), leaving the other scalability details to be handled automatically by AScale. Additionally, he can choose any data warehouse engine to store data (e.g., Relational data warehouses, column oriented, no SQL, Map-Reduce architectures).

Figure 2 AScale, ETL+Q scalability



4 Scalability mechanisms

In this section, we introduce how each part of AScale (ETL+Q auto-scale framework) scales individually to obtain the necessary performance.

Figure 2 depicts each part of the ETL scaling including:

1 The increase of Data Sources 1, and data source rates, implies the increase of data, leading to the need to scale other parts of the proposed framework. Each data source 1 has an extraction frequency associated with it (e.g., every minute).

2 The ‘Extraction and Data Distributor’ nodes forward and/or replicate the extracted (raw data) into the transformer nodes. Scaling needs in 2 are detected by monitoring the extraction time. If the extraction time is larger than a maximum configured limit, or if data extraction is not complete until the next extraction instant (e.g., every minute), more data distributor nodes 2 are required.

3 Transformation nodes process the data transformations programmed by the user. These nodes include a buffer queue to monitor data ingress. If the queue increases its size above a certain limit, the transformation node is scaled by replicating the transformation code into another node.

4 The data buffer holds transformed data, it can be in-memory or/and disk. These nodes are scaled based on memory monitoring parameters. If at any time the memory use reaches the maximum configured usable memory, a new node must be added.

5 Data switches are responsible for data distribution (pop/extract) from the 'Data Buffers' and placing it in the correct nodes for loading into the data warehouse. Each data switch is configured to support a maximum data-rate (e.g., 100 MB/sec). If that limit is passed or reached during a defined time window, more data switch nodes are needed.

6 The data warehouse can be in a single node, or parallelised by many nodes. Scalability of the data warehouse is based in two parameters: the loading time and query response time. If the data warehouse nodes take more time to load data than the maximum configured time, more nodes are added and data is re-distributed. If queries average execution time is more than the desired response time, data warehouse nodes must also be added.

Finally, the last scalability mechanism introduced in AScale is the global desired ETL processing time. A global time for the entire ETL process can be defined. If that global time is exceeded, then the AScale pipeline component that is nearest to its scaling limit is scaled-out.

5 Configuration parameters

Based on configuration parameters provided by the user, the system scales automatically. All the components interact together for providing automatic scalability to ETL+Q when more efficiency is needed in each stage of the ETL+Q pipeline. Conversely, the system scales down in any stage when excess resources are not needed. The main configuration parameters, for each part represented in Figure 2 for automatic scalability are related with:

1 Configuration of sources location for data extraction; extraction frequency, maximum extraction duration.

2 Distribution algorithm, RR algorithm is used to distribute data by the transformation nodes.

3 Transformations to be applied. Users can program the transformations in ASclae framework, using (importing) a provided Java library, or program in any other language connecting to the provide API.

4 Data buffers size (memory and disk).

5 Maximum supported distribution rate.

6 Load frequency, maximum load duration, data warehouse schema (made by the user), definition of fact tables and dimension tables (i.e., replication parameters).

7 Maximum desired queries execution time parameters. If querying takes more time than the configured data warehouse nodes are set to scale.

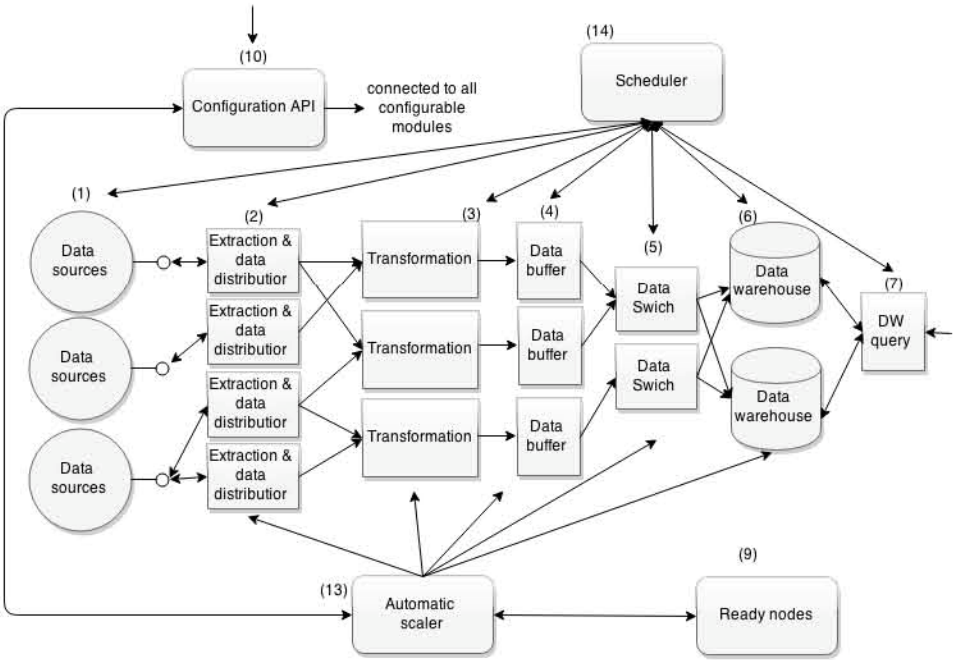
6 Scaling

In this section, we explain the mechanisms and configuration parameters that AScale uses to monitor, detect and scale each processing module. After explaining some of the most relevant configuration parameters, we detail how scaling decisions are made. In order to do that, we explain the implemented algorithms and illustrate how they work. Finally, we specify, for each part of AScale, the data distribution policies used to share and transfer data across different AScale modules.

6.1 Configuring AScale for automatic scalability

Each part of the ETL+Q process must scale in order to overcome performance limitations. We can have many Data Sources supplying data, and at each stage of the processing a single computing node may not be able to handle all data extraction, transformation or any other part of the AScale pipeline.

Figure 3 AScale framework for scalability



In this section, we describe the scalability configuration parameters used by AScale to scale each module, independently and as necessary.

Figure 3 shows each AScale module that may need to scale to offer desired performance.

Table 1 API configuration for data extraction

1	extractSetDataSourceLog(	
2	“source1”,	//source ID
3	{“logA”, “logB”},	//log ID
4	{“*/10 * * * * *”, “*/10 * * * * *”},	//extraction frequency
5	{“5s”, “5s”});	//maximum extraction time

Extraction nodes 2 are monitored to determine scaling needs based on extraction frequency, Table 1 line 4, and the maximum extraction time, Table 1 line 5.

If the maximum extraction time is exceeded, then more extraction nodes are added. If the maximum extraction time is not defined, then, if the extraction takes longer than the frequency cycle duration, more nodes 2 are added from the ready-nodes area 9.

Table 2 API configuration to configure the transformation maximum queue size

1	transformSetMaxSize(	
2	“16GB”,	//maximum queue size
3	“5GB”);	//maximum limit size for scaling detection

Transformation nodes 3 include a data queue with a maximum load size. Table 2 line 2 specifies the maximum queue size, and line 3 specifies the maximum limit size for scaling detection.

Ingress data goes inside the queues, then the transformation nodes, (with the transformation operations programmed by the data warehouse developer), extract and transform data. If at any point the queue starts filing up above a certain configured limit, it indicates that the ingress data-rate is more then the output transformation data-rate. Thus, more transformation nodes must be added.

When scaling-up, a new node is added and the entire transformation process, present in other nodes, is replicated to the new node.

Table 3 API configuration to specify the data buffer module storage size

1	dataBufferSetSize(	
2	“dataBuffer1”,	
3	“5GB”,	//maximum buffer memory size
4	“10GB”,	//limit buffer memory size
5	“500GB”,	//maximum buffer disk size
6	“250GB”,	//limit buffer disk size
7	“D:”);	//data buffer disk location

The *Data Buffer nodes 4* hold transformed data until the next data warehouse load instant.

Scaling decisions are made based on a number of parameters: maximum allowed in-memory buffer size; maximum allowed data write speed; and maximum allowed disk size. Table 3 illustrates these parameters.

If the memory usage reaches the maximum configured data buffer memory size, then data is swapped into disk. If even so the memory becomes completely full, reaching the maximum memory size, more data buffer nodes must be added. Also, if the disk space reaches a configured limit, more data buffer nodes must be added.

Table 4 API configuration for the data switch supported data rate

1	dataSwitchSetDataRate(	
2	“dataSwitch1”,	//data switch ID name
3	“80000l/ s”,	//maximum supported data-rate
4	“2m”);	//maximum time delay to trigger scale-out

Data switch nodes 5 distribute and replicate data across the data warehouse nodes. These nodes extract data from the data buffers 4 using a scheduler-based extraction policy and load it into the data warehouse nodes 6. However, there are limitations regarding the amount of data each data switch node can handle. The command line in Table 4 line 3, is used to specify the maximum supported data-rate in lines per second. Line 4 represents the maximum time delay before trigger scale-out mechanisms. If, for the configured time duration, the data switch is always working at the maximum configured data-rate that means that these nodes are working at their maximum capacity (according to configuration) and must be scaled.

Table 5 API configuration for the data warehouse extraction frequency

1	dataWarehouseSetLoad(	
2	“* 30 1 * * *”,	//load frequency
3	“5h”,	//maximum load time
4	“100MB”);	//maximum batch file

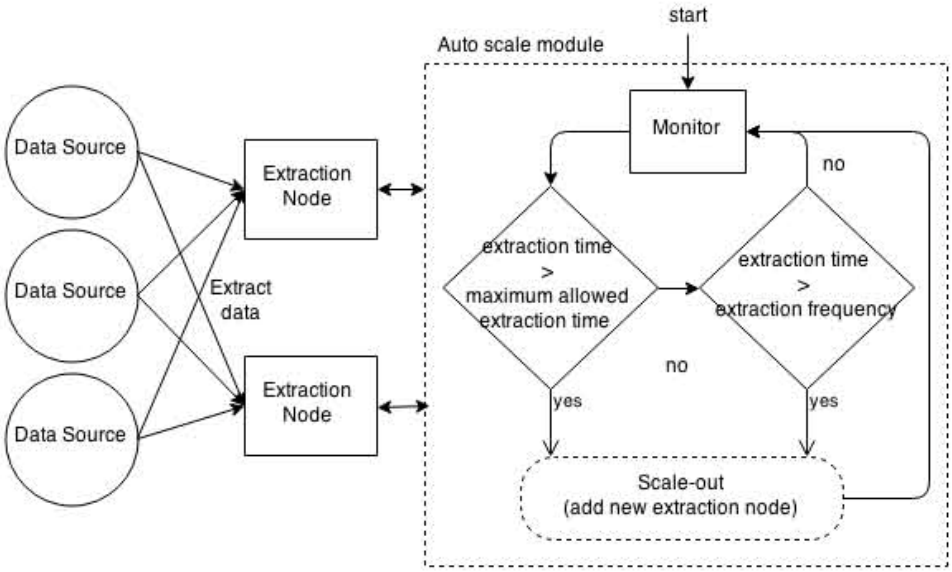
The *Data Warehouse nodes 6* load data during fixed instants for a certain time window. Table 5 line 2 represents the load frequency using the Unix cronjob time format, and line 3 represents the maximum allowed load time. If the maximum allowed load time is exceeded, then more data warehouse nodes need to be added. Another data warehouse scale scenario regards queries execution time. If queries take more time than a maximum configured limit to output the results, data warehouse nodes 6 must scale to offer more performance.

Table 6 Maximum query execution time API configuration

1	querySetMaxDWQueryExecutionTime(
2	value); //max execution time for DW queries
3	
4	querySetMaxD-DWQueryExecutionTime(
5	value); //max execution time for D-DW queries

Table 6 shows the API to configure the maximum query execution time. The input parameters include the maximum desired execution time in seconds (s) or minutes (m).

Figure 4 Extraction algorithm – scale-out



7 Decision algorithms for scalability

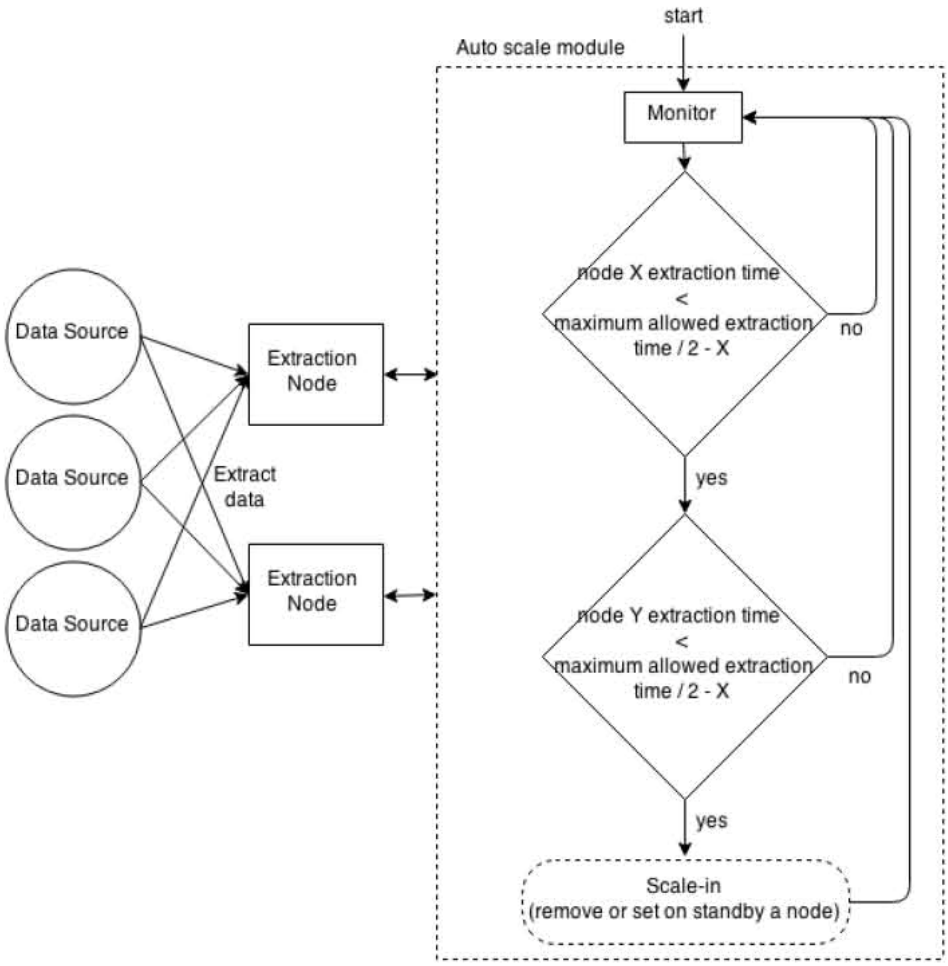
This section defines scalability decision methods as well as algorithms which allow AScale to automatically scale-out and scale-in each part of the proposed pipeline.

7.1 Extraction and data distributors

Figure 4 describes the algorithm used to scale-out. Depending on the number of existing sources and increasing data generation rate, eventually extraction nodes have to scale. The addition of more 'extraction and data distributors' nodes 2 depends on whether the current number of nodes is able to extract and process data with the correct frequency, within the configured maximum extraction time bound. For instance, if the extraction frequency is specified as every 5 minutes and extraction duration 10 seconds, then every 5 minutes, the 'Extraction and Data Distributor' nodes cannot spend more than 10 seconds extracting all data. Otherwise a scale-out is needed. If the maximum extraction duration is not configured, then the extraction process must finish before the next extraction instant, as specified by the extraction frequency parameter.

Figure 5 describes the algorithm used to scale-in. To save resources and reuse them, data extraction nodes can scale-in. The decision is made based on last execution times. If previous execution times of at least two or more nodes are less than half of the maximum extraction time, minus a configured variation parameter (X), one of the nodes is set on standby (as ready-node) or removed and the other ones takes over.

Figure 5 Extraction algorithm – scale-in



7.2 *Transform*

The transformation process is critical. If the transformation is running slow, data extraction at the current data-rate may not be possible, therefore information will not be available for loading and querying when necessary.

Transformation nodes have an input queue as shown in Figure 6. In Figure 6, we show the transform queue, used to determine when to scale the transformation phase. If this queue reaches a limit size (configured by the developer) because the actual transformer node(s) is not being able to process all data that is arriving (i.e., current ingress data-rate is larger than transformation output data-rate), then it is necessary to scale-out. Figure 6 describes the algorithm used to scale-out.

Figure 6 Transformation – scale-out

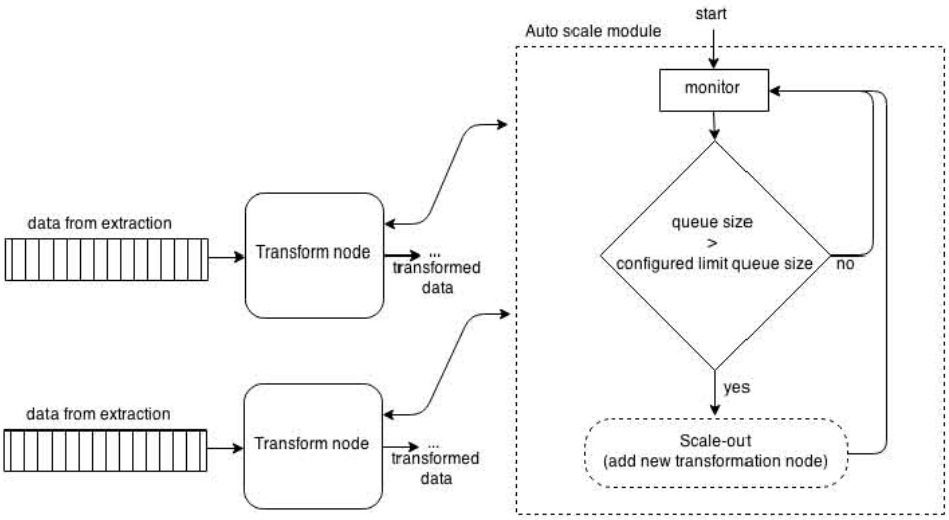


Figure 7 Transformation – scale-in

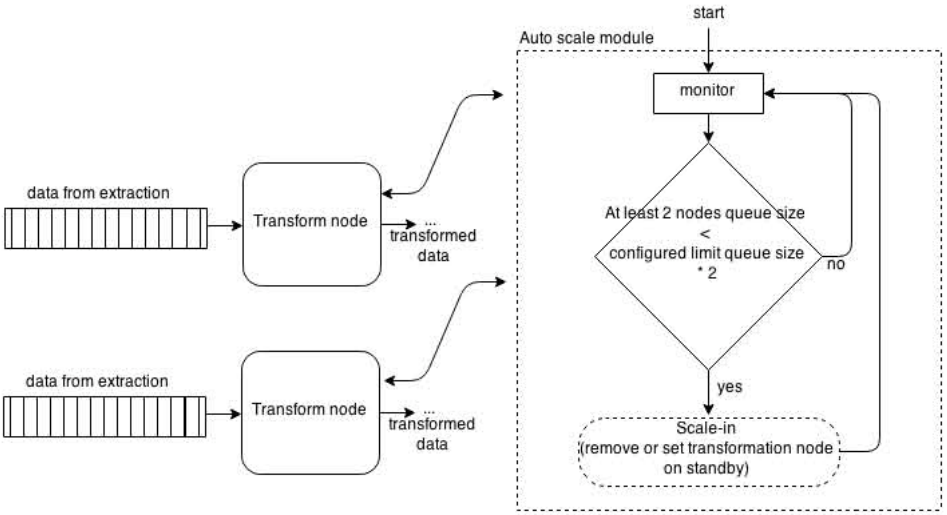


Figure 7 describes the algorithm used to scale-in. If queues size at a specific moment is less than half of the limit size for at least two nodes, then one of those nodes is set on standby (as ready node) or removed.

7.3 Data buffer

The data buffer nodes scale-out based on the memory size, the data swap speed from memory into disk and the available storage space to hold data. When the available memory queue size becomes above a certain limit, data starts being swapped into disk to reduce memory use under the limit size. If even so the data buffer memory reaches the

maximum memory limit size, then the data buffer scales-out. This means that the incoming data-rate (going into memory storage) is not being swapped to the disk storage fast enough, therefore more nodes are necessary.

If the disk space becomes full above a certain configured size, the data buffers are also set to scale-out.

Figure 8 describes the algorithm used to scale-out the data buffer nodes. Data buffers can also scale-in. In this case, the system will do so if the data from any data buffer can fit inside the data buffer of any other node.

Figure 8 Data buffers – scale-out

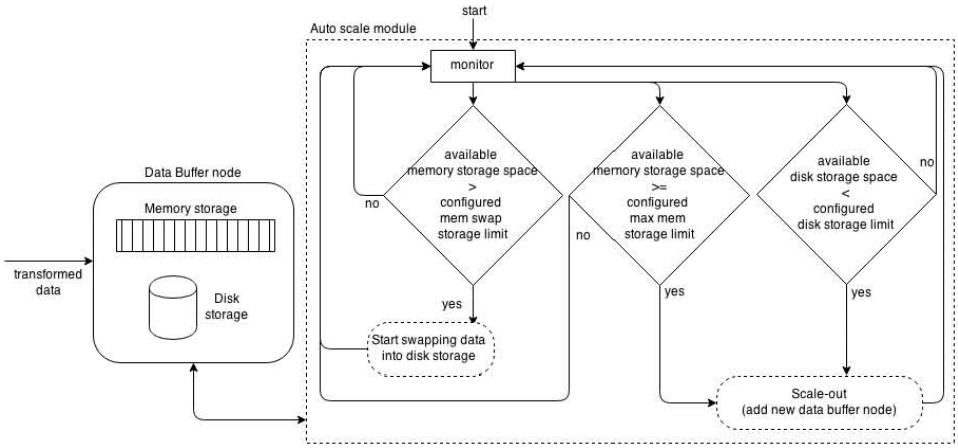
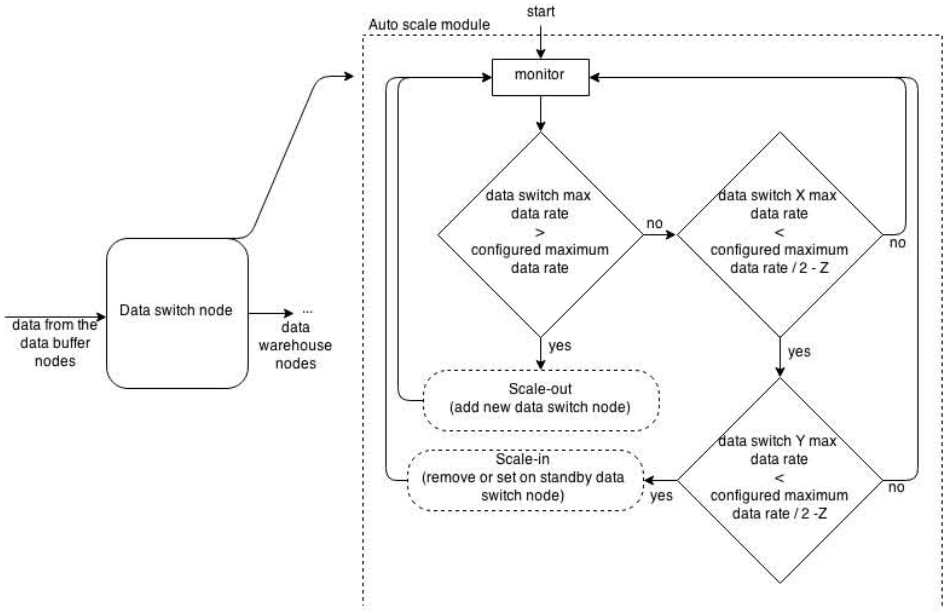


Figure 9 Data switch – scale



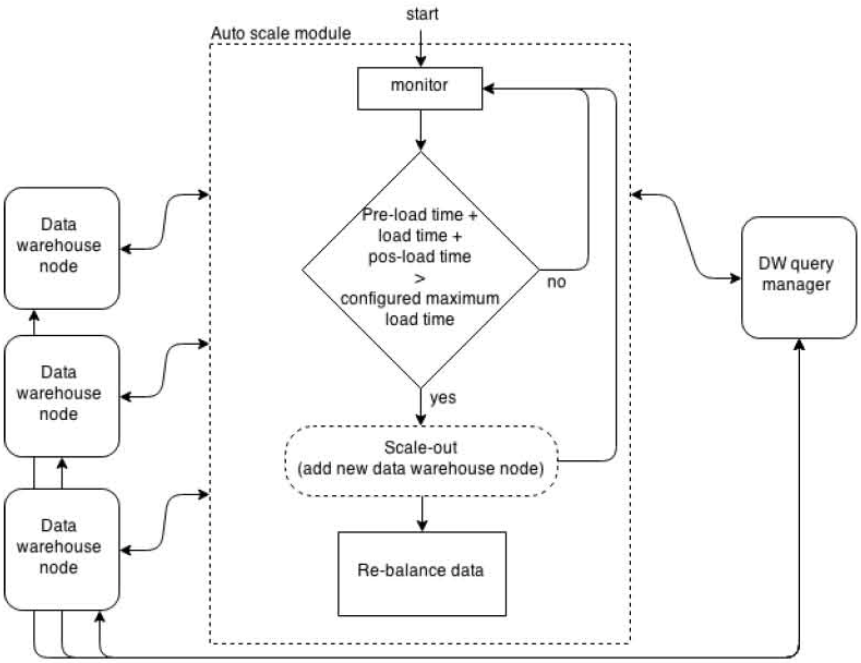
7.4 Data switch

The data switch nodes scale based on a configured maximum supported data-rate. That data-rate cannot be reached or passed for more then a configured time window. If the datarate rises above the configured limit for a certain time window, data switch nodes are set to scale-out. Figure 9 describes the algorithm used to scale the data switch nodes. The data switches can also scale-in. In this case, the system will allow it if the data-rate for at least two nodes is half of the configured maximum, minus a (Z) configured variation parameter.

7.5 Data warehouse

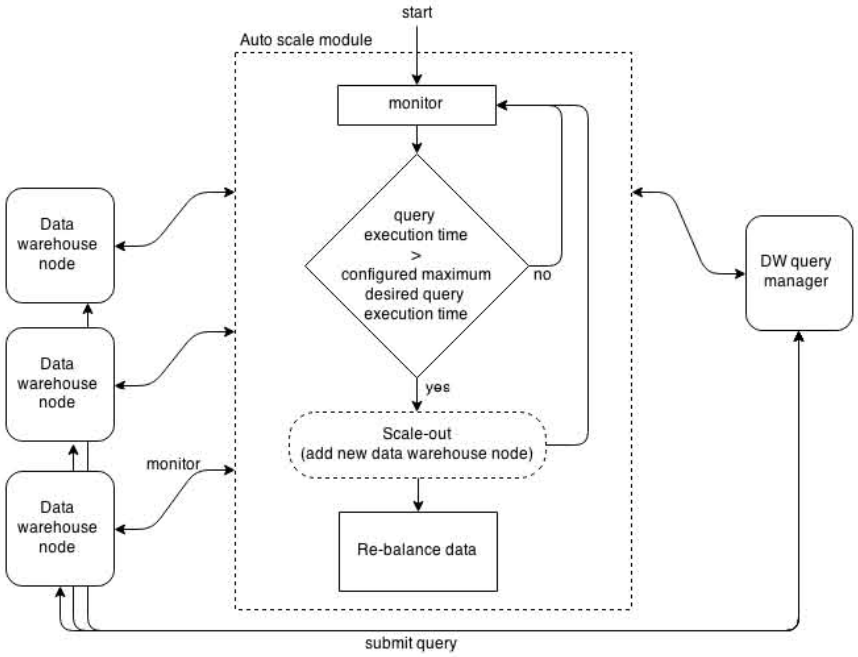
Data warehouse scalability needs are detected after each load process or after any query execution. The data warehouse load process has a configured limit time duration to be executed every time it starts. If that time is exceeded, then the data warehouse must scale-out. Likewise, queries also have a maximum execution time. If query execution time is exceeded, the data warehouse must scale-out. The number of nodes to scale-out is determined assuming linear scalabilty based on previous number of nodes and execution time $\left(\frac{loadTime}{targetTime} \times n\right)$. In this equation, the '*loadTime*' represents the last recorded load time, '*targetTime*' represents the desired target load time and '*n*' represents the current number of nodes. Figure 10 describes the algorithm used to scale the data warehouse when the maximum load time is exceeded.

Figure 10 Data warehouse – scale



Data warehouse scalability is not only based on the load and integration speed requirements, but also on the maximum execution queries time. After a query is executed, if the query time is more than the configured maximum, then the data warehouse is set to scale-out. Figure 11 describes the algorithm used to scale-out the data warehouse based on the query execution time.

Figure 11 Data warehouse – scale based on query time



Data warehouse nodes scale-in is performed iff the average query execution time and the average load time respects the conditions (1) and (2) (where n represents the number of nodes):

$$\frac{(n-1)\times avgQueryTime}{n} \leq desireQueryTime \tag{1}$$

and

$$\frac{(n-1)\times avgLoadTime}{n} \leq \max LoadTime \tag{2}$$

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: extract and replicate data from data warehouse nodes; load the extracted information into the new nodes (data is extracted and loaded across the available nodes as if it is new data).

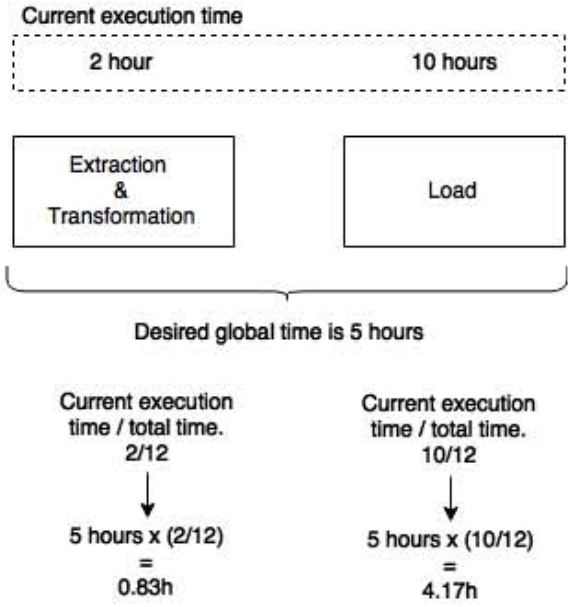
Both data warehouse scale-out and scale-in require the administrator approval, since this process can lead to extended offline periods.

7.6 Global ETL scalability

Besides defining partial limits for each part of the ETL+Q pipeline, it is also useful to configure only a desired global ETL processing time. In this case, AScale will choose to scale-out the part of the pipeline that is performing slower.

Figure 12 explains based on an example the scaling of the ET and L. Assume that the current execution time of ET and L are respectively 2 and 10 hours, and the desired execution time is 5 hours. Based on these times, the target time for the ET is 0.83 hours and for the L is 4,17 hours. Then by applying following formula the necessary number of nodes is estimated linearly, $\frac{currentTime}{targetTime} \times n$, where '*currentTime*' is the current execution time, '*targetTime*' is the desired execution time and '*n*' represent the current number of nodes. For the given example this results in: for the ET we would need $\frac{2}{0.83} \times 1 = 2,\ 4\ nodes$ which correspond to 3 nodes, and for the L, $\frac{10}{4.17} \times 1 = 2,\ 39\ nodes$, which corresponds to 3 nodes. Note that, in the ET the extra nodes are added to the E process only, then the T process scales based on ingress data queues monitoring to keep up with the extraction rate.

Figure 12 Example, estimating the scaling proportion of the ET and L



Another option is to configure both types of time bound limits: a global time bound for the entire ETL process and at the same time local bounds for the parts. In this case, AScale can use the local bounds to decide where to scale.

8 Experimental setup

In this section, we describe the testbed, hardware, software and ETL operations used in the setup. An experimental setup was built to simulate not only the data warehouse, but also, all ETL processes. The decision of using TPC-H data as source data logs, and SSB as the data warehouse schema and queries, was taken to reuse related work from the research group, for instance, Ferreira and Furtado (2013) and Martins et al. (2011), which had already some parts of the framework pipeline developed. This option also allowed us to better control data transformations and corresponding staging area volume and data synchronisation, allowing us to build it with less complexity, thus easier to test the proposed AScale concepts.

Data sources logs for extraction: the structure of the simulated logs is the same as the TPC-H generated data logs structure, consisting of logs representing each of the tables: part, supplier, nation, region, partsupp. Regarding the tables 'lineitem' and 'orders', they were merged into a single log with the following structure: the log is a set of 'order' rows and for each order the log contains the respective related lineitem rows as subsequent rows of the order.

Data extraction is made considering the start and end of each order (including the respective items), in order to keep data together and consistent.

Data transformation: after data is extracted from TPC-H log files, it is set to be transformed. The TPC-H tables, part, supplier, nation, region, partsupp are also keep in the transformation nodes (staging area) using a Postgre SQL database. The stored data is transformed in order to recreate the SSB schema.

Additional transformations were applied: names were split and concatenated into last name and given names. The first letter of each name was set to upper-case and remaining letters were set to lower case; Addresses were cleaned and transformed by converting keywords (e.g., street) into abbreviated words, using a translation lookup table in memory. Moreover, the first letter of each name was set to upper-case and remaining letters to lower case. The postal code was added (concatenated), using a translation table in memory to correlate each city with a postal code; The phone numbers were converted into groups of three numbers, adding the country and city code, depending on the postal address code; Categories were converted/written into full text (no abbreviations); Dollar coin numbers were converted into Euros; sizes and weights were converted into the normalised international system. The data output from these transformation operations is then stored and ready to load.

Data warehouse: the data warehouse has the same base structure as the SSB benchmark.

8.1 Hardware and software

The experimental tests were performed using 12 computers, denoted as nodes, with the following characteristics: Intel Core i5-5300U Processor (3M Cache, up to 3.40 GHz); Memory 16 GB DDR3; Disk western digital 1TB 7,500 rpm; Ethernet connection 1 Gbit/sec; Connection switch SMC SMCOST16, 48 Ethernet ports, 1 Gbit/sec.

Software installed/used. The 12 nodes were formatted before the experimental evaluation and installed with: Windows 7 enterprise edition 64 bits; Java JDK 8; Netbeans 8.0.2 Oracle Database 11g Release 1 for Microsoft Windows (X64); MySQL 5.6.23 used in the dynamic data warehouses; PostgreSQL 9.4 used for lookups during the

transformation process; Esper 5.1.0 for Java as CEP process engine; TPC-H benchmark dataset; SSB benchmark, representing the data warehouse schema and queries.

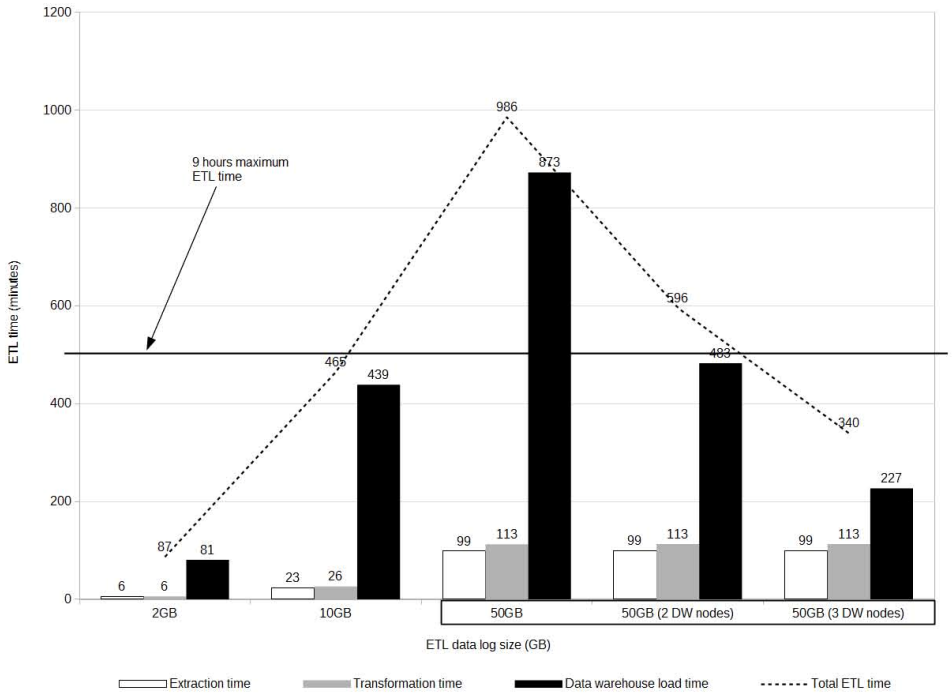
For this experimental evaluation, we assume that a node corresponds to a physical machine. However, due to limited available resources, some virtual nodes were created, and other nodes resources redirected and reused, in such way that AScale pipeline processing was not affected.

9 Typical data warehouse scenario

In this section, we evaluate AScale in a scenario where, due to log sizes and limited resources, data load takes too long to perform if scalability is not applied. We start with only two nodes (two physical machines), one for handling extraction and transformation, the other to hold the data warehouse. 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 configured with an extraction frequency of every 24 hours and a maximum extraction duration of 4 hours, a transformation queue with a limit size of 10 GB and data warehouse loads were configured for every 24 hours, with a maximum duration of 9 hours.

Figure 13 AScale, 9 hours limit for ETL



The experimental results in Figure 13 show the total AScale ETL time using two nodes (two physical machines), one for extraction, transformation, data buffer and data switch [Figure 13(a)], the other for the data warehouse [Figure 13(b)]. Up to 10 GB of log size, the ETL process can be handled within the desired time windows. However, when increasing to 50 GB, 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) exceed the desired time window. When scaled to 3 nodes, by adding another data warehouse node [Figure 13(b)], the ETL process returns to the desired time bound.

10 Near-real-time and minimal downtime scenarios

In this section, we assess the scale-out and scale-in abilities of the proposed framework in near-real-time scenarios, as well as scenarios where downtime should be minimised. Near real-time scenarios require data to be always up to date and available to be queried (i.e., data freshness). In order to guarantee this, it is necessary to integrate new data in a predefined and very small time window.

The scenario was set-up as follows: E (extraction) and L (load) were set to perform every 2 seconds; T (transformation) was configured with a maximum queue size of 500 MB; the load process was made in batches of 100 MB maximum size. The ETL process is allowed to take 3 seconds at most.

Figure 14 Near-real-time, full ETL system scale-out

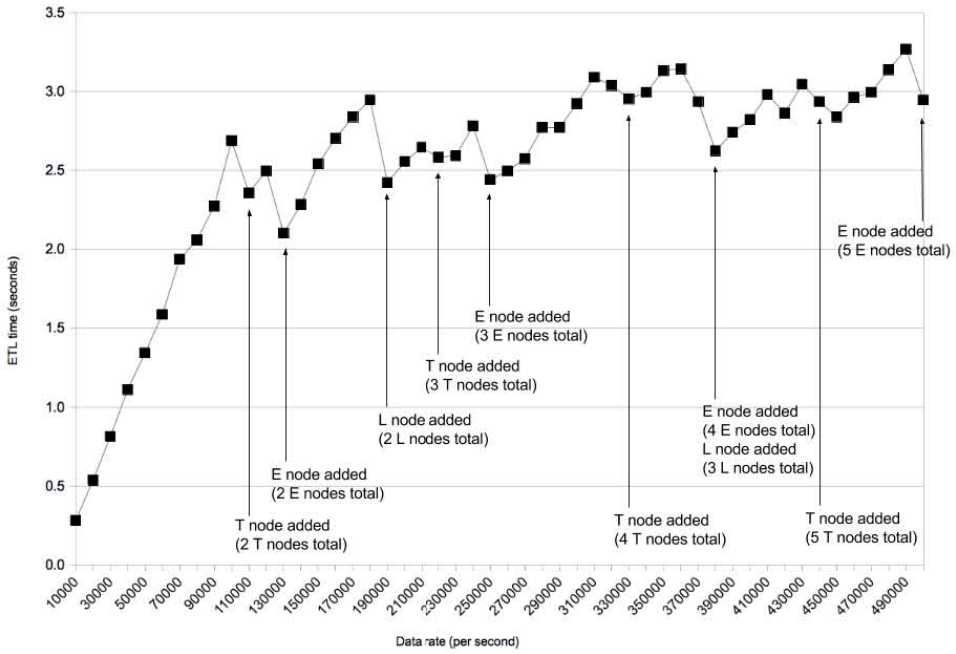
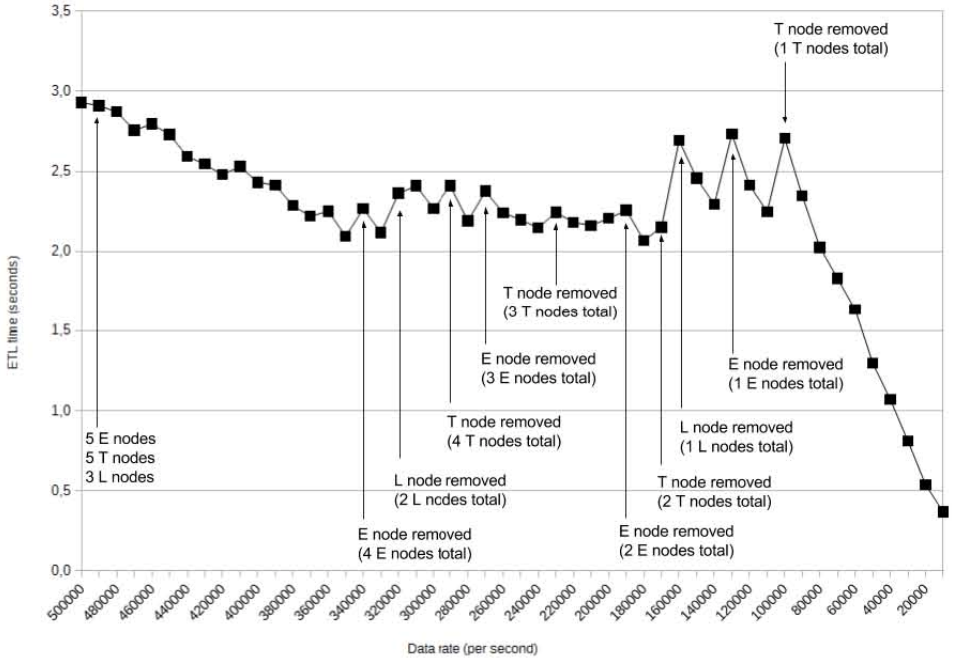


Figure 15 Near-real-time, full ETL system scale-in



Figures 14 and 15 show AScale scaling-out and scaling-in automatically to deliver the configured near-real-time ETL time bounds while the data rate increases or decreases respectively. The X axis represents the data-rate, from 10,000 to 500,000 rows per second; the Y axis is the ETL time expressed in seconds; 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 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.

Scale-out results in Figure 14 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.

Scale-in results in Figure 15 show the instants when the current number of nodes is no longer necessary to ensure the desired performance, leading to removal of some nodes (i.e., set as ready nodes in stand-by, to be used in other parts).

The next (sub)sections detail how each part of the ETL and queries scale-out in the near-real-time scenario.

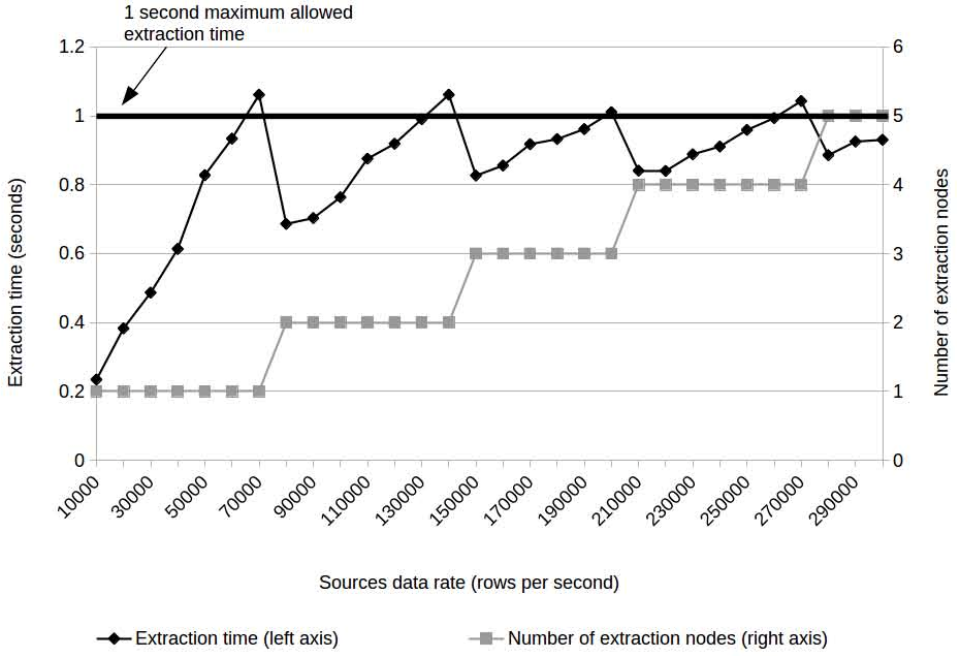
10.1 Scalability of data extraction

Considering data-sources generating high-rate data and extraction-nodes to extract the generated data, when the data flow is too high, a single data node cannot handle all ingress data. In this section, we study how the extraction nodes scale to handle different data-rates, using the data setup similar to the one used in

the previous section. The maximum allowed extraction time was set to 1 second ($max_{extractionTime} < max_{desiredExtractionTime}$); and extraction frequency was set to every 3 seconds ($max_{extractionTime} < ExtractionFrequency$).

Figure 16 shows the extraction nodes automatic sacalbility in near-real-time. The left Y axis represents average extraction time in seconds, the right Y axis represents the number of nodes used; the X axis represents data-rate; the black line represents extraction time; the grey line represents the number of nodes as they scale-out. Every time the extraction time is above the configured threshold limit of 1 second, a new node is automatically added (from the pool of ready-nodes). After the new node is added, more nodes are being used to extract data from the same number of sources, improving the extraction performance.

Figure 16 Extraction scalability



10.2 Transformation scalability

During the ETL process, after data is extracted, it is set for transformation. Because this process can be computationally heavy, it is necessary to scale the transformation nodes to ensure that all data is processed without delays. Data ingress transformation queues are monitored. Once it is detected that a queue is full above a certain configured threshold (i.e., $Rate_{extract} \geq Rate_{transform}$), AScale scales the transformation process.

The transformation nodes scale-out mechanisms were set to limit the queue memory size to a maximum of 5 GB before swapping data into disk, and 500 MB as the limit to trigger the scaling mechanism (corresponding to 100,000 rows). Figure 17 shows the addition of new transformation nodes, as queue sizes increase above the configured limit. The Y axis represents average queue size in number of rows, the X axis shows the data

rate expressed in rows per second. Each plotted bar represents the average transformation queue size (up to 4 nodes). As can be seen in Figure 17, as soon as the data rate reaches 80,000 rows/sec, AScale detects overload in the queue (queue size above 100,000 rows) and triggers addition of a new node. The new node can be seen in the 120,000 rows/sec bar, where two queues allow the system to handle the data rate satisfactorily. After that, as the data rate reaches 200,000 rows/sec the maximum queue size is reached again triggering addition of one more node. The scale-out can be seen in the 240,000 rows/sec bar, where three queues are again able to handle the increased load. A fourth and fifth node are added again to scale at 320,000 rows/sec and 440,000 rows/sec respectively, as can be seen in Figure 17.

Figure 17 Automatic transformation scalability

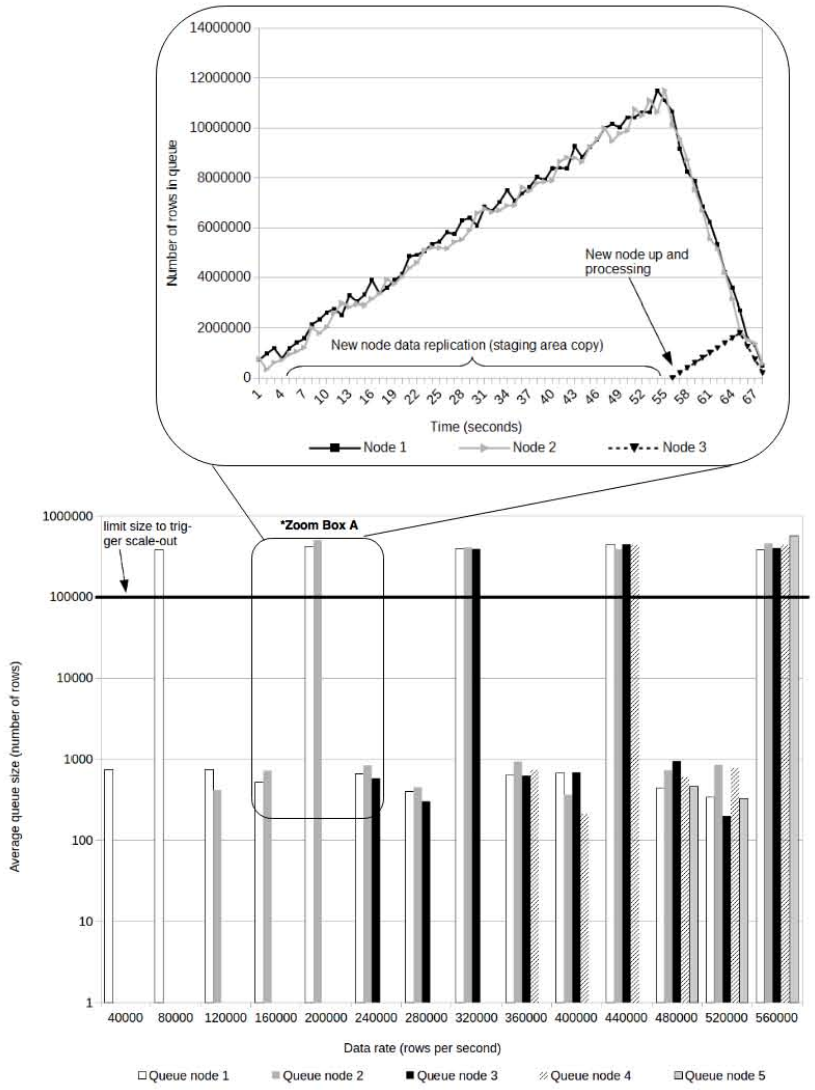


Figure 17 ‘Zoom Box A’ shows the detail of what happens when the queue sizes of transformation nodes increase above the configured limit. As overload is detected a new transformation node is added. The new node takes approximately 1 minute to be ready (see zoom box A), the delay corresponds to the necessary time to replicate/synchronise the node and respective staging area. After the node is fully working, data is distributed using a least renaming work (LWR) algorithm, which makes the queue of the new node balance the sizes of all nodes.

10.3 Data buffer nodes

Data buffer nodes hold the transformed data until it is loaded into the data warehouse. For this experiment the data buffers were configured as follows: we consider only the data generation/producer, there is no data ‘consumer’, so the buffer must hold all ingress data; the generation data-rate speed was set to 1,500,000 rows per second (i.e., transformation output data rate) in such way that the disk speed cannot swap all data fast enough, leading the memory to increase until its maximum; available memory storage was set to 10 GB; memory storage limit before data swap into disk was set to 5 GB; available disk storage was 1 TB.

When the limit memory size is reached (5 GB), data starts being swapped into disk. However, because the disk speed cannot handle all ingress data-rate, the memory reaches the maximum limit size (10 GB). At that moment a new data buffer node is added. After the new node is added, data is distributed, using LWR, making the queue of the newly added node increase faster. After a while, the data volume in each data buffer returns to normal.

Figure 18 Data buffer swap into disk and scaling

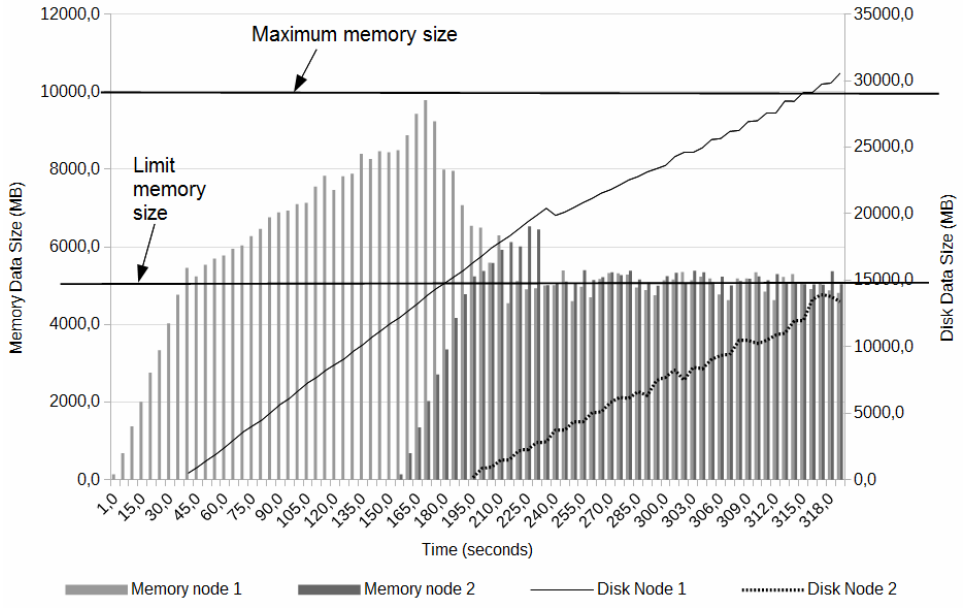
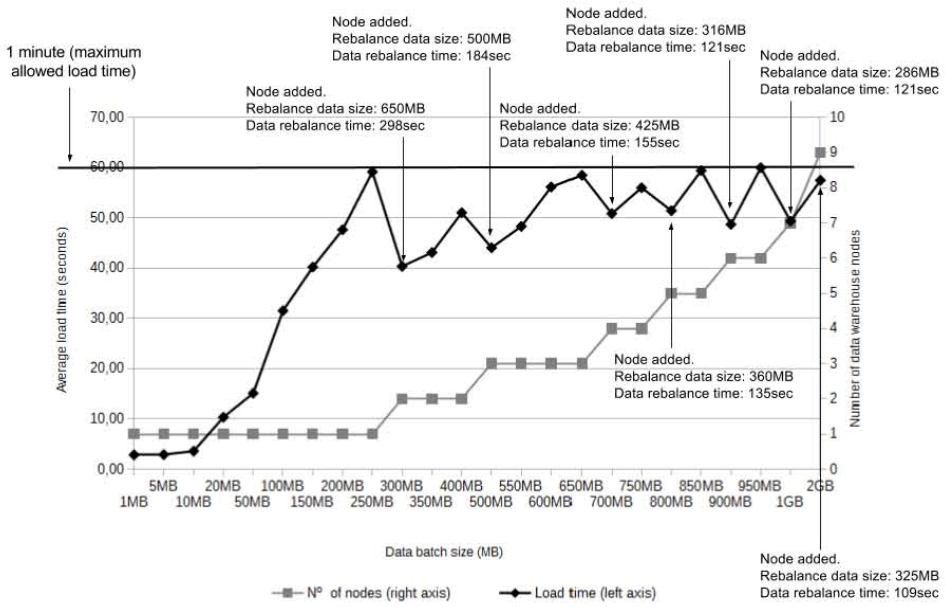


Figure 18 shows an extreme scenario where the data buffer write speed from memory into disk (swap) is not fast enough. This scenario leads to a scale-out of the data buffer node. Figure 18 shows the data queue size increasing until the 'limit memory size' (5 GB), at that moment data starts being swap into disk in an attempt to release memory space. However, because the data rate is too high the memory continues increasing until the 'maximum memory size'. Once the memory is at the 'maximum memory size' another node is added and data is distributed by both nodes. In Figure 18, we also show the data memory queue of the second node increasing and the swap process occurring again. Although because now there are two nodes handling the ingress data rate, the data swap speed can free the memory.

Figure 19 Data warehouse load scalability



10.4 Data warehouse load and query scalability

In this section, we test the data warehouse scalability, which can be triggered either by the load process (because it is taking too long), or because query execution is taking more time than the configured response time bounds.

To test the load scalability we create the setup: loads are from batch files, each approximately size 100 MB each; the maximum allowed load time is set to 60 seconds; each time a data warehouse node is added we show the data size that was moved into the new node and the required time in seconds to re-balance data; all load and re-balance times include the execution of pre-load tasks (i.e., drop all indexes and views) and pos-load tasks (i.e., build all indexes and views).

Load scalability: the experimental results in Figure 19 show the data warehouse scaling when the data size to be loaded increases and as consequence the load time also increases above the predefined bound: the left Y axis represents average load time in

seconds; the right Y axis shows the number of data warehouse nodes; the X axis represents the data batch size in MB; the horizontal bar at Y = 60 seconds represents the maximum configured load time; at each scale-out moment there are notes specifying the data re-balanced size and time to perform it; the black plotted line represents average load time; the grey plotted line represents number of data warehouse nodes.

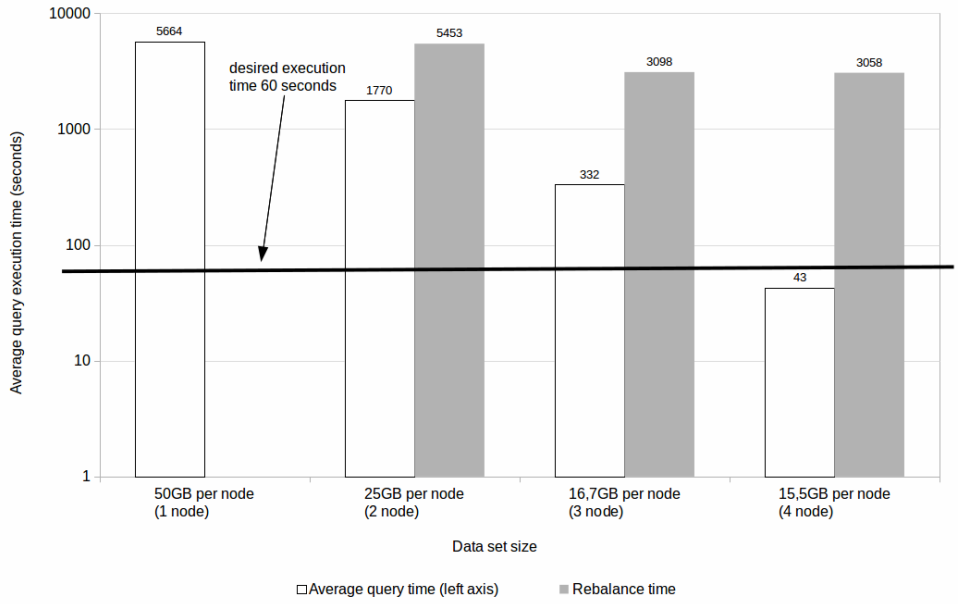
The results in Figure 19 show how load performance degrades as the data size increases and how it improves when a new node is added. After a new node is added, performance improves to meet the maximum configured load limit.

Query scalability: when running queries, if the maximum desired query execution time is exceeded, the data warehouse is set to scale-out in order to offer better query execution performance. The following workloads were considered to test AScale query scalability:

- Workload 1 (WL 1):
  - a 50 GB total size
  - b execute Q1.1, Q2.1, Q3.1, Q4.1 randomly chosen
  - c desired execution time per query: 1 minute (60 seconds).
- Workload 2 (WL 2) – as workload 1 but, 1 to 8 simultaneous sessions used.

Workload 1 studies how the proposed mechanisms scales-out the data warehouse when running many queries. Workload 2 studies AScale scalability running simultaneous sessions (e.g., number of simultaneous users). Both workloads were set with the objective of guaranteeing the maximum execution time per query of 60 seconds.

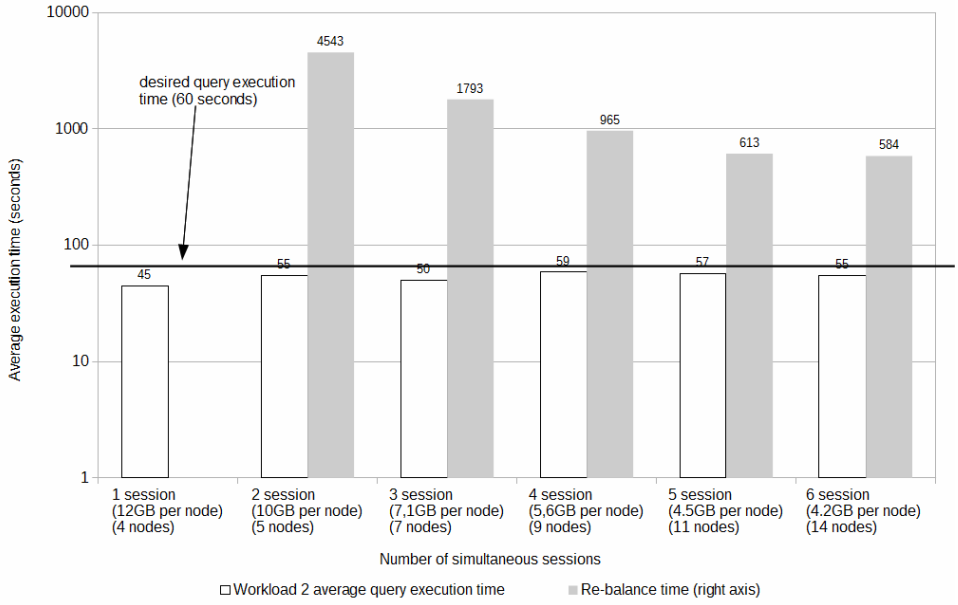
Figure 20 Data warehouse scalability, workload 1, 50 GB dataset



Query-based scalability, WL1: Figure 20 shows the experimental results for workload 1, where: the Y axis represents the average execution time in seconds; X axis represents the data size per node and the current number of nodes; the horizontal line over 60 seconds represents the desired query execution time; white bars identify the total workload time and grey bars the re-balance time (i.e., extract data, load into nodes, rebuild indexes and views). The results show that every time a query is executed and the average query time is not inferior to the maximum configured query execution time, one extra node is added (scale-out). In each scale-out the re-balance time represents the necessary time to extract data from existing nodes, re-distribute it and rebuild indexes and views. Once the average query time becomes lower than the configured desired execution time, the framework stops scaling the data warehouse nodes.

Simultaneous session query scalability, WL 1: Figure 21 shows how the data warehouse scales when simultaneous sessions are executing. Figure 21 shows the left Y axis represents average query execution time in seconds; X axis shows the number of sessions, the data size per node and the number of nodes; grey bars represent the data re-balance average time in seconds (i.e., extract from nodes, load into new node, rebuild indexes and views); white bars show average query execution time. The results in Figure 21 show that, the number of simultaneous session's increases, the system scales the number of nodes in order to provide more performance. Thus, average query execution time follows the configured parameters.

Figure 21 Data warehouse scalability, workload 2, 50 GB dataset



Note that since both loads and query execution are performed against the data warehouse and the data warehouse is scaled, AScale query execution performance improves at the same time the data warehouse load performance improves and vice versa.

11 Conclusions and future work

We proposes an approach and a framework, named AScale, that automatically scale the ETL+Q process, allowing the developer to focus only in the conceptual ETL+Q model. We highlight the contributions: an approach to automatically parallelise ETL and Query execution (ETL+Q), able to modify individual components when they need to scale out or in; dynamic-data-warehouse (D-DW). We propose an in-memory dynamic store and processing approach which, when added to the system, provides total freshness and real-time; experimental evaluation of the proposals, showing that AScale is able to scale-out when performance bottlenecks are detected, and that it is also able to scale-in when resources are not needed.

There are a number of interesting directions for future work, for instance, the implementation of proactive and predictive scalability mechanisms, visual tools (drag-and-drop) to build schemas and to configure ETL processes, exploration and applicability to the cloud for elastic scalability.

References

Albrecht, A. and Naumann, F. (2009) 'METL: managing and integrating ETL processes', *VLDB PhD Workshop*.

Ferreira, N. and Furtado, P. (2013) 'Real-time data warehouse: a solution and evaluation', *International Journal of Business Intelligence and Data Mining*, Vol. 8, No. 3, pp.244–263.

Karagiannis, A., Vassiliadis, P. and Simitsis, A. (2013) 'Scheduling strategies for efficient ETL execution', *Information Systems*, Vol. 38, No. 6, pp.927–945.

Liu, X. (2012) *Data Warehousing Technologies for Large-Scale and Right-Time Data*, PhD thesis, dissertation, Faculty of Engineering and Science at Aalborg University, Denmark.

Liu, X., Thomsen, C. and Pedersen, T.B. (2012) 'Map Reduce-based dimensional ETL made easy', *Proceedings of the VLDB Endowment*, Vol. 5, No. 12, pp.1882–1885.

Martins, P., Costa, J., Cecílio, J. and Furtado, P. (2011) 'VarDB: high-performance warehouse processing with massive ordering and binary search', *Data Warehousing and Knowledge Discovery*, pp.184–195, Springer.

Pentaho (2014) Pentaho, 2014-10-07.

Simitsis, A., Gupta, C., Wang, S. and Dayal, U. (2010) 'Partitioning real-time ETL workflows', in *Data Engineering Workshops (ICDEW), 2010 IEEE 26th International Conference on*, pp.159–162, IEEE.

Simitsis, A., Vassiliadis, P. and Sellis, T. (2005a) 'Optimizing ETL processes in data warehouses', *Data Engineering, 2005: ICDE 2005: Proceedings 21st International Conference on*, pp.564–575, IEEE.

Simitsis, A., Vassiliadis, P. and Sellis, T. (2005b) 'State-space optimization of ETL workflows', *Knowledge and Data Engineering, IEEE Transactions on*, Vol. 17, No. 10, pp.1404–1419.

Thomsen, C. and Bach Pedersen, T. (2009) 'pygrametl: a powerful programming framework for extract-transform-load programmers', *Proceedings of the ACM Twelfth International Workshop on Data Warehousing and OLAP*, pp.49–56, ACM.

Vassiliadis, P. and Simitsis, A. (2009) 'Near real time ETL', *New Trends in Data Warehousing and Data Analysis*, pp.1–31, Springer.

Wang, G. and Guo, C. (2011) 'Research of distributed ETL engine based on MAS and data partition', *Computer Supported Cooperative Work in Design (CSCWD), 2011 15th International Conference on*, pp.342–347, IEEE.