LOAD BALANCING IN DATA WAREHOUSE – EVOLUTION AND PERSPECTIVES

The problem of load balancing is one of the crucial features in distributed data warehouse systems. In this article original load balancing algorithms are presented. The Adaptive Load Balancing Algorithms for Queries (ALBQ) and the algorithms that use grammars and learning machines in managing the ETL process. These two algorithms base the load balancing on queries analysis, however the methods of query analysis are quite different. While ALBQ bases on calculation of computing power and available system assets, the gaSQL algorithm includes direct grammar analysis of the SQL query language and its classification using machine learning. The WINE-HYBRIS algorithm that uses the CUDA architecture and Cloud Computing will be presented as a platform for developing the gaSQL algorithm.

1. ADAPTIVE LOAD BALANCING ALGORITHM FOR QUERIES (ALBQ)

The ALBQ [15] was designed as a load balancing algorithm in IMR (Integrated Meter Reading)/DSTDW (Distributed Spatial Telemetric Data Warehouse) environment basing on research presented in [11], previous versions of load balancing algorithms used in these systems are presented in [1][2][3][4][17]. It adapts system state to each processed query. It is done through monitoring, and if necessary modifying system state, before query processing [5][6][7][9][12][13]. The ALBQ also allows performing the balancing action regardless of system changes. The main module in the ALBQ is...
the balancing process overseer module (denoted as a server), which oversees and manages nodes and all system processes. System nodes main function is still storing and processing data, however they also have possibility to communicate with other nodes, send data and transmit information about their load [8][10]. The node’s load includes its computing power as e.g. mean load of the CPU in a certain timeframe. Moreover, the algorithm expects that the node can assess (or appoint) the cost of query processing, before it decides whether to process it. To minimize the loss of data in case of a node failure, the ALBQ divides data space into regions, and assumes that the tuples existing in the same region cannot be stored only in one node. This solution is similar to using the partitioning attribute [4][10]. There are several parameters that influence and adjusts algorithms to users’ needs, the most important are **inertia parameter (IP)**, **overload threshold (OT)** and **computing power (P)**:

- The inertia parameter in the ALBQ algorithm determines modification method of load balancing results. The inertia parameter takes the value $0 < \epsilon < 1$.
- The overload threshold, determines the frequency of a balancing process activation – it is the load deviation from the reference value. The low value of this parameter implies that even the slightest load deviation will be considered as the imbalance state.

The **computing power $P$** can vary while system operates and it reflects changes in the node’s load. The actual value of the $P$ parameter does not have to be directly connected with any node’s feature, but its value has to adequately define node’s computing abilities.

### 1.2. QUERY PROCESSING PHASES

The query directed to the data warehouse is delivered to the server. Upon arrival the query is broadcasted to registered nodes, in which the query is analyzed in terms of required computing power. The computing power needed to process the query is directly proportional to the number of tuples needed to be processed. To simplify the computing power can be described as the estimated number of tuples being processed by the query. Although the estimated value along with the current load is sent back to the server, the server load is unchanged. After collecting responses from all nodes that processed the query, server normalizes the loads with regards to the estimate increase of their value and appoints the **imbalance factor (overload)** $b_i$ for each node. If the value
of \( b_i \) is greater than the default threshold, it is sent to the node, which appoints the number of tuples that should be moved to other nodes.

This value is set in a manner that allows the node load to be within boundaries of system’s mean nodes load.

\[
\Delta t = \frac{b_i \times P}{100} \tag{1}
\]

where \( \Delta t \) is a number of tuples to be transferred, \( b_i \) – imbalance factor (value of a load that needs to change in order to return to balances state), a \( P \) node’s computing power.

### 1.2. BALANCING THE SYSTEM

In a balancing process we can distinguish two main stages. This division is enforced by the need to assure the stability of nodes that do not need balancing, while performing balancing actions in imbalanced nodes. In the first stage of load balancing the server gathers data sent from nodes, and decides the actions to be taken in order to balance the system. This is implemented in the Balance method, which returns total values for given node. The obtained value actually is the node’s overload value that should be deducted from node and the decision how it should be realized is made by the node.

**Balance method**

Set \( L_p = \text{default load} \)

Set \( L_{\text{max}} = L(N_{\text{max}}) = \max(L(N_i)) = \text{max load through nodes} \)

If \( L_{\text{max}} \leq L_p + p \text{ terminate method} \)

Set \( L_{\text{min}} = L(N_{\text{min}}) = \min(L(N_i)) = \text{min load through nodes} \)

Transfer adequate load value \( l \) from \( N_{\text{max}} \) to \( N_{\text{min}} \)

Set \( l = \min(L_{\text{max}} - L_p, L_p - L_{\text{min}}) \)

Include the inertia parameter

Store the transfer in the transfer matrix \( m[N_{\text{max}}][N_{\text{min}}] += l \)

Go to begin of the method

Where: \( L_i \) i-th node load equal \( L(N_i) \); \( p \) – overload threshold; \( m \) – result matrix which is temporary division policy

While receiving imbalance factor \( b_i \) the node calculates the number of tuples to be transferred \( \Delta t \). Then the adequate number of tuples is appointed for transfer and their statistics are sent to the server. These statistics will be
used in a second stage of load balancing, when the server decides to which node the tuples should be transferred. This is implemented by the \textit{Dispatch} method, called by the server.

\textbf{The Dispatch method}

\begin{align*}
\text{Set } & k = \sum_{i=1}^{\infty} T(R_j) \\
\text{Set } & k_{\text{left}} \text{ as a number of tuples to transfer} = k \\
\text{Set } & K_i \text{ as a number of tuples that should be transferred to the } N_i \text{ node }
\end{align*}

\{\textit{CreateDistribution} method\}

\begin{align*}
\text{For each } R_j \text{ region} & \\
\text{For each destination } N_i & \\
\text{Set } & t = \frac{T(R_j) \times K_i}{k} \\
\text{Store the tuples division in a result matrix: } & S_{ij} = t \\
\text{Decrease the number of tuples for transfer: } & k_{\text{left}} = k_{\text{left}} - t \\
\text{If } & k_{\text{left}} = 0 \text{ terminate method} \\
\text{For each region } R_j & \\
\text{Set } & r_{\text{left}} = \text{number of not allocated nodes in region } R_j \\
\text{Set } & N_d = \text{the last destination node for tuples from } R_j \\
\text{For each node } & N_i \\
\text{Set } & N_d = \text{logically next node towards } N_d \\
\text{Set } & n_{\text{left}} = \text{number of nodes that can be sent to } N_d \\
\text{If the number of tuples in } N_d & \text{ is lower than } K_d \text{ then} \\
\text{Store the division in a } & S_{dj} = \min(n_{\text{left}}, r_{\text{left}}) \\
\text{If } r_{\text{left}} = 0 & \text{ then the last destination node for tuples from } R_j = N_d
\end{align*}

1.3. DIVISION POLICIES AND EXPERIMENTS

The key element in the ALBO is a module that stores and manages \textbf{division policies}. These policies are integral part of the algorithm and decide how the data (new or already stored) should be broadcasted through the whole system. There are two types of policies: \textbf{Main Division Policy (MDP)} and \textbf{Temporary Division Policy (TDP)}.

The \textit{MDP} decides how the tuples arriving to the system should be distributed, the total values of a \textit{MDP} for nodes should be constant during system activity. The \textit{TDP} is set during the Balance method and denotes how the tuples should be transferred (the \textit{Dispatch} method). The \textit{TDP} is created for a certain node and it presents the load that should be transferred to other nodes.
that exists in TDP. Because of the TDP characteristics the total values can vary. The experiments performed for 1.000.000 tuples, with 2% threshold and the inertia factor set to 0.5. Obtained results (Fig. 1, Test 3.9) are compared to the case of system without adaptive balancing (Fig.1, Test 3.8).

We can observe more dynamic increase of processing time in a system without adaptive balancing which is caused by a larger number of processed nodes. If the node’s computing power drops to 80% the adaptive balancing starts to be efficient. Moreover while computing power drops to 55% the ALBQ causes that the query processing is faster than in a system without load balancing.

2. THE WINE-HYBRIS AND THE GASQL ALGORITHM

Modern systems tend to implement the zero-latency (real-time) or near-zero-latency ETL processes that updates data without interrupting the data warehouse activities [20]. The data that users obtain are most up-to-date. While designing the zero-latency ETL process there are some problems to be faced [21]. In certain solutions it is hard to assign the processing priority to users queries, and the updates generated by system [2]. The other complicated task is foreseeing the query processing time. The problems mentioned above should be taken into consideration while designing the zero-latency ETL process. To successfully cope with the real-time requirements the ETL process have to solve the problem of assigning priorities to processed queries and upcoming updates. One of the simplest method is the use of FIFO algorithm, however this leaves the system highly ineffective and does not adjusts itself to the data warehouse working environment. Other solution that can oversee the ETL process is the LEMAT system [16] based on the WINE algorithm [18].
Moreover, to meet the goals of adjusting to current system state, the LEMAT system uses the SVN learning machine. The next solution implements the load balancing problem is the original solution - the WINE-HYBRIS algorithm that uses CUDA technology [19] [27] and Cloud Computing to speed-up query responses [14].

![Fig. 2. The main processing scheme of the WINE-HYBRIS](image)

The use of Workload Balancing Unit (WBU), controls all operations related to the processing and analysis of both updates and queries in the data warehouse (DW). The base that was used to create the WINE-HYBRIS algorithm is a two-level scheduling WINE algorithm (Fig.2). Using the particular characteristics of the WINE algorithm, it was possible to incorporate user preferences, balancing and prioritization of queries and updates. Such approach enables the balancing to: a) maintain the freshness of data at the appropriate level, and b) respond to users’ queries. The algorithm also uses the term partition that can be presented as a subset of the data warehouse’s data table. The tests performed in the WINE-HYBRIS environment based on two architectures 1) CUDA and CPU (Fig.3) and 2) Cloud Computing.
All experiments were performed on two different machines, in order to verify the behavior of data extraction in different hardware environments.

Tests proven that the use of both the CUDA architecture as well as the Windows Azure platform (Cloud Computing) increases performance, which translates into more efficient data processing. An additional advantage of Cloud Computing, is the costs optimization, - you only pay for the allocated capacity in a given period of time. This also relieves allocated resources while not processing any queries and updates in the data warehouse. While creating the WINE-HYBRIS algorithm, method, which is responsible for packet prioritization queries and updates, was enhanced with the schema of two algorithms with FIFO Group. It is based on the simplest solution, resulting in minimizing the number of operations on queues during processing queries and updates.

2.1 QUERY ANALYSIS IN GASQL

During implementation the usage of a Ridor classifier was proposed for machine learning. The general schema of above mentioned solutions can be described as follows: first the query content is analyzed, and then processed with machine learning, an adequate speed class is assigned to it. In other research the use of OneR learning machine was proposed along with usage of a larger number of classifiers. The system created for the ETL management bases on the researched LEMAT system. It also includes query analysis and classification and the load balancing method. The main part is terms of query processing is the module of queries analysis and classification. If properly configured it can minimize the analysis time and increase the number of properly classified queries. The query is processed in a following manner: the update or
query arriving to the system activates the algorithm. If the incoming event is a query it is analyzed and classified as slow, medium or fast (in terms of processing time). Information about classification result along with certain analysis data is rerouted to the queuing module. Next actions are taken in the load balancing module. If the query is allowed to be executed, it is processed and the information about the real processing time is sent to the classification module. There the information is stored and includes in the actions of a learning machine. During the balancing process, when the update request appears it is analyzed, and the chosen information is forwarded to the load balancing module. In the LB module update is queued and executed in an appropriate moment. Updates are queued using FIFO. The LEMAT system used for query analysis bases on regular expressions. To make this mechanism more efficient algorithms basing on direct grammar analysis connected with the SQL grammar syntax was proposed. There are basic issues connected with grammar analysis that should be presented: (1) Lexical symbol, denoted as sequence of signs matched to certain pattern (2) Non-terminal symbol, denoted as the representation of a lexical symbols sequence (3) Production, denoted as a manner in which lexical symbols and non-terminal symbols connects to create signs (4) Introduction of symbols sequences (non-terminal and terminal) A and B that are a sequences of used productions that leads from A to B.

In the presented SQL query grammar analysis we need lexical analysis and syntax analysis. The lexical analysis is responsible for downloading and converting source data into lexical symbols stream that can be used in the syntax analysis. During this stage source signs are read, grouped in lexical symbols and forwarded for further analysis, moreover there can be an initial analysis of input signs. In the second stage of the analysis, the syntax analysis, the lexical symbols sequences, generated by the lexical analyzer are checked if they can be generated using grammar. The syntax analysis relies on grouping the source program lexical symbols into grammar expressions, which can be used by the compiler to synthesize the result code.

2.2 MACHINE LEARNING AND CLASSIFIERS

In a data warehouse load balancing module, queries are queued according to the WINE and the LEMAT. One of main algorithm parameters is the expected time of query processing in a data warehouse. Queries classification uses learning machines, and appropriate classification strongly influences the efficiency of a data warehouse load balancing module. To evaluate learning machines we use the efficiency parameter \( S \), which denotes the quotient of a
adequately classified instances of analyzed objects \( (n_p) \), to the number of all analyzed objects \( (n_c) \):

\[
S = \frac{n_p}{n_c}
\]  

(1)

In case of a presented system, the instance is the classified query. There are also defined following terms:

- All-out Efficiency (AE) – the efficiency value for all queries appearing in the system.
- Temporary Efficiency (TE) – is a temporal evaluation of classifiers efficiency. It is a value of efficiency for all queries appearing in the system from the last learning process.

To increase the effectiveness of query classifiers, we implemented classifiers from the Lemat system e.g. SVM, OneR and Ridor. The Ridor classifier was also used in a WINE_HYBRIS. The OneR learning machine proven to be the most efficient during classifiers tests, provided by the WEKA library while working in the Lemat system. The idea of using multiple classifiers was bases on mechanisms used in implementing single learning machine, with an additional transitional module between classifiers and the load balancing module.

3. SUMMARY

The presented ALBQ algorithm bases mainly on the analysis presented in [10]. The algorithm balances the nodes load before processing the actual query, which minimizes the time needed to obtain query response. The test shows quite useful features of the ALBQ when used in a system with nodes incapable of constant utilization of their full computing power. The next ALBQ’s useful feature is ability to divide the data in a ETL update process basing on statistical data collected during system activity. The WINE-HYBRIS along with modified query classification gaSQL proves to be quite effective when using the CUDA technology and cloud environment. The method of a grammar analysis used in gaSQL is faster and more efficient then analysis using regular expressions in the LEMAT system. Moreover, the idea of using multiple classifiers enables system to adapt to incoming queries or improving the query instances optimization.
Currently, author works on a load balancing problem and mobility models in a cellular networks environment, as well as problems connected with data stream processing paradigm and user identity unification. Along with an introduction of the Long Term Evolution (LTE) network technology, new challenges of solving the load balancing problem emerged.

- In Distributed Data Warehouses (DDW) main goal of load balancing is minimizing the Minimal Response Time (MRT) which is a mean time interval between query arrival time and its response time.

- In Cellular Networks (CN) main goal of load balancing is to distribute the traffic from highly loaded cells to other parts of the system thus optimizing the availability of radio resources through the whole network.

Although these two environments and load balancing goals are virtually quite different, however the above-presented algorithms could be partly adapted from the data warehouse environment to fit other distributed systems e.g. the cellular network system. The load balancing is still an unresolved issue in LTE networks, where only basic methods are implemented as a standard and each vendor freely introduces its own solutions to the system. There are several methods of dealing with overloaded cells in LTE, e.g. adjusting the pilot power. Although manually controlled larger pilot power gives more range and larger coverage area, it can lead to creating coverage holes. The other method of load balancing, more similar to load balancing in data warehouse is to modify handover region between neighboring cells so users can migrate from heavy loaded cells to less loaded ones. Such approach is called mobility load balancing (MLB) and is a part of Self-Organizing Network (SON). The load in case of data warehouse are data which have to be loaded to the DDW, and in case of CN are the mobile devices connecting to base stations. Above mentioned author load balancing algorithms could be partially adapted as a MLB in LTE networks.

The other important research perspectives of load balancing in DW, are solutions to data privacy problem and data recovery.

REFERENCES


