Scalable Cloud-based ETL for Self-serving Analytics

Conference Paper · July 2019

CITATION
1

READS
95

4 authors:

Eftim Zdravevski
Ss. Cyril and Methodius University in Skopje
157 PUBLICATIONS **1,489** CITATIONS
SEE PROFILE

Cas Apanowicz
CogniTrek Corp.
13 PUBLICATIONS **78** CITATIONS
SEE PROFILE

Krzysztof J. Stencel
University of Warsaw
101 PUBLICATIONS **530** CITATIONS
SEE PROFILE

Dominik Ślęzak
University of Warsaw
318 PUBLICATIONS **4,631** CITATIONS
SEE PROFILE

Scalable Cloud-based ETL for Self-serving Analytics

Eftim Zdravevski[1], Cas Apanowicz[2], Krzysztof Stencel[3], and Dominik Ślęzak[4]

[1] Faculty of Computer Science and Engineering
Ss. Cyril and Methodius University, Skopje, Macedonia
eftim.zdravevski@finki.ukim.mk
[2] CogniTrek, Toronto, Canada
cas.apanowicz@cognitrek.com
[3] Faculty of Mathematics, Informatics and Mechanics
University of Warsaw, Warsaw, Poland
stencel@mimuw.edu.pl
[4] QED Software, Warsaw, Poland
dominik.slezak@qed.pl

Abstract. Nowadays, companies must inevitably analyze the available data and extract meaningful knowledge. As an essential prerequisite, Extract-Transform-Load (ETL) requires significant effort, especially for Big Data. The existing solutions fail to formalize, integrate and evaluate the ETL process for Big Data in a scalable and cost-effective way. In this paper, we introduce a cloud-based architecture for data fusion and aggregation from a variety of sources. We identify three scenarios that generalize data aggregation during ETL. They are particularly valuable in the context of machine learning, as they facilitate feature engineering even in complex cases when the data from an extended time period has to be processed. In our experiments, we investigate user logs collected with Kinesis streams on Amazon AWS Hadoop clusters and demonstrate the scalability of our solution. The considered datasets range from 30 GB to 2.5 TB. The results were deployed in the domains, such as churn prediction, fraud detection, service outage prediction, and more generally – decision support and recommendation systems.

Keywords: Data warehouses, Data streams, ETL, Business analytics

1 Introduction

Ubiquitous smart devices, sensors and social media result in sheer data volumes, while consumers became accustomed to personalized services that are available instantaneously. Delivering targeted information shapes the success of many companies, health providers and governmental institutions. In the past, they could decide which data to store by making compromises between available resources and capabilities to manage the data. In the era of Big Data, companies experience growing pressure to store and analyze the whole data that is being collected just to stay competitive in the data-driven marketplace.

Several steps are needed to make the data available in a usable format: identification of all relationships and business context, data collection and ETL, which usually is time-consuming in terms of both development and execution. Once the data is processed and loaded into a data warehouse (DWH), it needs to be fully ready for reporting, visualization, analytics and decision support. Even though all building blocks for efficient ETL and Big Data analytics are present on the market, there is no comprehensive cloud-based architecture offering an integrated, scalable and cost-effective solution. Most approaches are either for specific purposes or only provide general definitions [1, 2].

In this article, we propose an architecture that first addresses the integration of high-velocity data by using scalable streaming technologies and Lambda functions. Then, it performs ETL using a combination of traditional tools for processing dimensional data, and Spark – for processing high-volume transactional data. We discuss detailed steps for performing three generic ETL scenarios covering a variety of real applications, ranging from traditional Business Intelligence (BI), to feature engineering in machine learning, such as churn prediction and fraud detection. In such applications, events like "the time that passed from the last occurrence of event X", "the time since the user's last login", "last use of a service", or "last bought product" could be valuable features.

Most importantly, the whole process is integrated from end-to-end and evaluated in a production environment on real high-velocity Big Data, something that lacks in most related approaches. The three scenarios were evaluated with different workloads ranging from 30 GB to 2.5 TB using the proposed architecture on Hadoop clusters deployed on Amazon AWS. The evaluation of each step of the three ETL scenarios showed that the cluster size could be optimized so it can process the required data volume within the expected time.

2 Related Work

Traditional BI relies on ETL tools for data import into DWH servers [3]. For reasonably sized data volumes there are ETL tools that have been successfully used in organizations throughout the years, such as Informatica, IBM Infosphere Datastage, Ab Intio, Microsoft SQL Server Integration Services (SSIS), Oracle Data Integrator, Talend, Pentaho Data Integration Platform (PDI), etc. Recently, ETL tools started to evolve into Enterprise Application Integration (EAI) systems that now perform much more functionalities than just ETL. Traditional ETL and ELT (Extract-Load-Transform) tools are reviewed in [4], with a focus on description of their terminology and capabilities, but without a discussion on how to tackle Big Data challenges. Scalable loading of data in NoSQL tables is one such challenge, which could be addressed by proper row key designs (i.e., their clustered index), as elaborated in [5].

The authors of [6] propose the BigDimETL approach, which aims to conserve the multidimensional DWH structure while integrating Big Data. However, the work is only theoretical, with no experimental evaluation.

Quite often, a user prefers a "quick and dirty" approximation over a correct answer that takes much longer to compute. Online aggregation in [7] was proposed to address this issue, as the batch-oriented nature of traditional MapReduce implementations makes these techniques hard to apply.

The idea of in-database analytics is pursued by the MADlib open source library [8]. It provides an evolving suite of SQL-based algorithms for machine learning, data mining and statistics that run at scale within a database engine, with no need for the data import/export to other tools.

GraphLab [9] expresses asynchronous, dynamic, graph-parallel computation while ensuring data consistency and achieving a high performance degree in the shared-memory setting, which is not originally supported by MapReduce and Spark. Our approach also recognizes that data consistency is essential, but achieves it differently, by relying on consistent dimensional tables. Consistent DWHs allow using data mining and machine learning libraries directly within the database system. Alternatively, consistent data in DWH could be used with more traditional visualization, reporting and BI services.

Another distributed parallel architecture for Big Data ETL is proposed in [10], but its limitation is that ETL should be completed before the data is aggregated. It is alleviated with our solution by performing aggregation during the ETL process. An approach that proposes a set of rules to map star schemas into NoSQL logical models with a pre-computed aggregate lattice is described in [11]. Similar to our case, the aggregate metrics need to be defined up front so that ETL can calculate them. The CloudETL system presented in [12] exploits MapReduce and Hive for distributed data processing, focusing on slowly changing dimensions. Our approach goes beyond it by using Spark for faster processing and Lambda functions for handling high-velocity data.

GENUS system [13] deals with data veracity by cleansing and tagging, similarly to our idea of standardization by templates. However, a drawback of this approach is poor evaluation, especially concerning high volume, versatility and velocity of the data. From the veracity perspective, the authors consider just one simple example. Moreover, the document store used is XML, without any scalability considerations. A real-time data ETL framework was presented in [14] to process historical/incoming data separately. Dynamic mirror replication technology was proposed to avoid the contention between OLAP queries and OLTP updates. A kind of drawback is that the evaluation of this methodology was conducted on a static dataset of only 16 GB.

Reference architecture for Big Data systems and classification implementation technologies and products/services, which is based on analysis of published implementation architectures of Big Data use cases, is provided in [15]. It aimed to facilitate architecture design and selection of technologies or commercial solutions when constructing Big Data systems. Their recommendations are considered in the design of the proposed system. From the perspective of the aforementioned areas of deployment of the proposed architecture, we also compared our work with other approaches referring to Big Data analysis in combination with data mining and machine learning, such as [16].

3 Architecture

The proposed system is shown in Figure 1. In organizations, commonly there are traditional data sources, such as relational database systems and structured/semi-structured data from internal or third-party data providers, that generate reasonably-sized data. This kind of data can be processed with traditional data integration tools. In our experiments, Pentaho Data Integration Platform (PDI) was utilized for such ETL tasks, which process the incoming low-volume data and store it in DWH (marked with light gray arrows in Figure 1).

We chose PDI because it enables users to ingest, blend, cleanse and prepare diverse data from any source. Its visual tools eliminate coding and complexity of creating data pipelines. It offers the data agnostic connectivity spanning from flat files to Hadoop, powerful orchestration and scheduling capabilities (including notifications and alerts), agile views for data modeling/visualization on the fly during the data preparation process, support for Hadoop distributions, Spark, NoSQL data stores and analytic databases, etc.

On the other hand, if there are data producers that generate Big Data with high volume, velocity or versatility, then the classical approach for ETL is not suitable. Big Data streams can be efficiently collected and processed by Distributed Streaming Platforms (DSP), which are scalable, replicated and fault-tolerant (e.g., Apache Kafka, Amazon Kinesis, etc.).

By defining a retention policy, DSPs can be configured to retain the data on the queue for a specific time after it was published, regardless if it was consumed or not. For example, for Amazon Kinesis the maximum data retention period is one week. DSPs allow the same data stream to be consumed by multiple consumers independently and simultaneously, each of them working at their own pace. Accessing the data on a DSP queue can be performed by either push or pull mechanisms. The pull mechanism is innate for Amazon Kinesis and Apache Kafka, so each consumer has and manages its read pointer.

Our solution allows consumption of DSP queues by the three most common types of consumers: Push Lambda functions (stream-based model), as well as Storage and Analytics Stream Pullers. The first two types are redundant alternatives for permanent raw data storage on different Object Storage containers, such as Amazon S3 or Windows Azure Blob Storage (WABS). Each of them is reliable with guaranteed Service Level Agreement (SLA). Using both of them can simplify deployment procedures and further improve the system's reliability. If the data format changes drastically or sources vary, consumers can be updated without any downtime or risk of data loss. Having both alternatives also provides integration convenience with the existing infrastructure.

Once the data is permanently stored on S3 or WABS in a raw format, we employ another Lambda function, which triggers after new files are deposited in a particular location. This function can cleanse the data (e.g., extract plain text from HTML files) and ingest it to Full-text search indexing services, such as Elasticsearch or Solr, which would subsequently provide free-text search functionality [17]. To some extent, this results with robustness to data veracity and complements the analytical capabilities of DWHs.

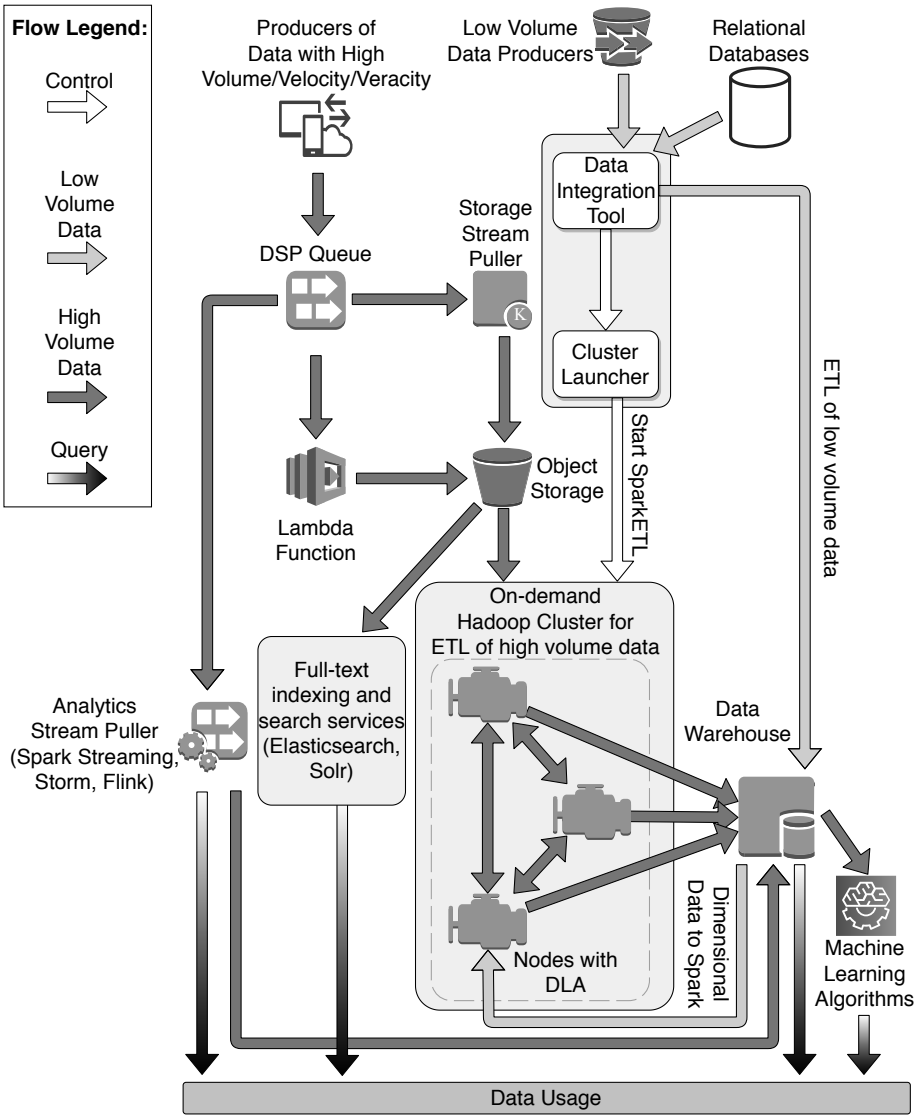


Fig. 1: Architecture of scalable cloud-based Big Data Warehouse.

Analytics Stream Pullers (e.g., Apache Spark Streaming, Apache Storm or Apache Flink) are a different kind of consumers that process data streams to provide near real-time insights and analytics. Our architecture complements this by employing on-demand Spark clusters for implementing more sophisticated algorithms for ETL and feature engineering. They can analyze changing trends over extended time periods (e.g., week-by-week or month-by-month comparisons of various metrics) or find the time since some particular event happened. Such metrics are not computable with Analytics Stream Pullers.

To facilitate on-demand starting of Spark clusters, on the machine that hosts Data Integration Tool (DIT) there is a Cluster Launcher module. It can be invoked manually or based on a predefined schedule by DIT. Cluster Launcher can start an Amazon EMR or Azure HDInsight cluster with configurable size and can run a particular Spark job. After the Spark cluster is started, it downloads the source code from a release branch of a code repository and automatically starts it. Code development and management adhere to the adopted organization's strategy (e.g., GitFlow), which defines rules and best practices for conflict resolution, peer-review, merging to staging and production branches, etc. Each Spark cluster during its lifetime executes only a specific ETL job. If the organization requires multiple ETL processes of unrelated data, then multiple Spark jobs can be defined and for each of them a separate workflow is managed (i.e., separate code repositories, execution schedules, target DWHs).

Next, the so-called Distributed Load Agent (DLA) is executed on all cluster nodes to process distinct portions of HDFS data generated by Spark. After DLA work is complete, the data is available in DWH for various BI tools and data mining or machine learning methods. Traditionally, data ingestion is a massive burden on database servers and often is a bottleneck. After Extract-Transform steps are completed and the primary/foreign keys are set, the load needs to be performed. The idea of DLA comes from the principles of edge computing, and it is partly inspired by the technology described in [18]. The goal is to offload most of the work to remote machines away from DWH. These edge nodes would compress the data and prepare an output, which could be simply copied to the database end. Thus, the overall impact on the database server would be minimal. In the proposed architecture, the whole on-demand cluster is considered as being on the edge from the DWH perspective (see Figure 1).

4 ETL Data-flow Scenarios

Let us describe three ETL data-flow scenarios commonly needed in organizations. These scenarios relate to the significant data portion to be stored in DWH, i.e., fact tables. The volume of dimensional data is considerably smaller. Thus, it usually does not require processing based on Big Data technologies; rather traditional ETL tools are sufficient. The proposed architecture assumes that traditional ETL tools already process dimensional data and that one only needs to handle the data to be stored in fact tables. The steps for implementing the ETL process of the three scenarios are shown in Figure 2.

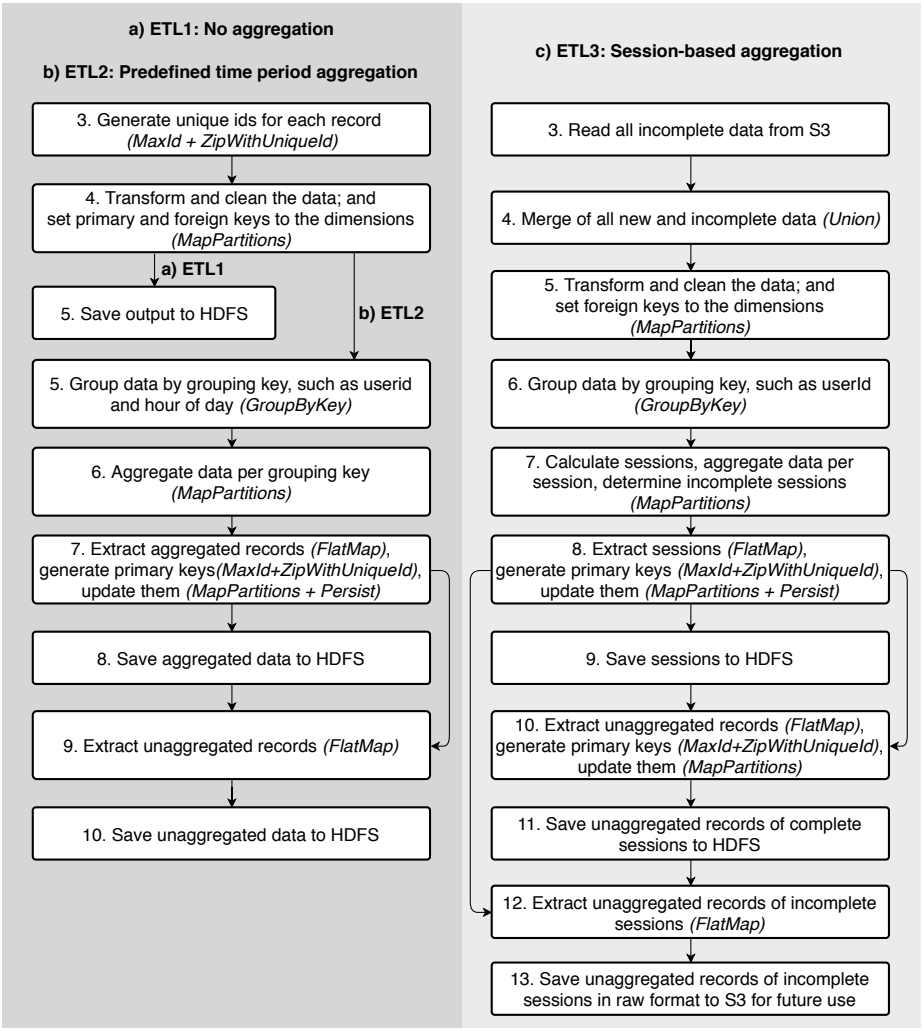


Fig. 2: Extract/Transform steps in Spark for completing the three ETL scenarios

Business (i.e., natural) keys denote unique record identifiers that could have some business meaning, but most importantly, they are managed by operational data stores (ODS). In DWH, a surrogate key is a necessary generalization of the ODS business key and is one of the essential elements of DWH design. Every join between dimension tables and fact tables in a DWH environment is based on surrogate keys, not business keys. It is up to the data extraction logic to systematically look up and replace every incoming business key with a DWH surrogate key each time either a dimension record or a fact record is brought into DWH. Surrogate keys provide independence from ODS business keys, which could be subject to deletion, updating or recycling.

Populating foreign keys in fact tables in a traditional way with joins between the fact and dimension tables would be inefficient for large datasets because it requires shuffling and redistributing the data across the cluster nodes. On the other hand, if the dimensions' business and surrogate keys are distributed across nodes in advance, populating foreign surrogate keys is a simple dictionary lookup operation based on business keys with $O(1)$ complexity. This method does not require reshuffling and adheres to the data locality principle.

Figure 2 shows the data flow for all proposed scenarios. The first two steps are common, so they are omitted. Each scenario is described in details in the following subsections. After the cluster is started per a defined schedule, Spark first loads business and surrogate keys of all processed dimensions. Also, the maximum values of surrogate keys for each table are calculated in step 1, because they define the starting values of new surrogate keys to be generated during a subsequent run of the ETL process. Then, Spark distributes them with the Broadcast operation to each node. For non-existing business keys, new surrogate keys are generated as a sequence of increasing integers, starting from the current maximum key for the table. Gaps in the generated sequence of numbers are allowed by design (for computational efficiency). During the surrogate key generation, their density is calculated (defined as the ratio of the total number of surrogate keys and the maximum), which shows how efficiently they are used. If the density is low and the maximum value of surrogate keys increases rapidly, this may be used to recommend a redesign. Step 2 reads all new data from S3 (using Spark operations TextFiles or WholeTextFiles).

4.1 ETL Scenario 1: No Aggregation

The first scenario requires parsing, data type conversion, setting foreign keys (Extract-Transform steps) and loading only into the fact tables of DWH. This scenario is the simplest of the three and does not need any aggregations in the fact tables. It is required for the whole generated data be available at the lowest level of granularity (after performing proper cleansing), including associations with other entities in the system. Some typical use-cases of this workflow refer to the log analysis in resource management, application troubleshooting, marketing insights, regulatory compliance, security, etc. What is common about these use-cases is that the original data needs to be preserved entirely without any level of aggregation so that particular events can be pinpointed. Hence, it is also worth

mentioning one more application aspect – regulatory compliance and security. Indeed, maintaining compliance with industry regulations often requires the data to be preserved in a source format to tag certain events.

Step 3 generates unique numeric identifiers for each record with the ZipWithUniqueId transformation. Even though there can be gaps in the generated numbers, it does not require data shuffling, making it very efficient. When the maximum surrogate key value (MaxId) is added to the generated number, a unique surrogate key of each record is obtained. In step 4 the transform phase of ETL is performed, consisting of data transformations, data cleaning, type casting, setting primary surrogate keys (using unique IDs generated in the previous step) and setting foreign keys to the dimensions (by performing lookups in the dictionary already distributed to each node in step 1). This step uses the MapPartitions Spark operation, which guarantees that the transformations will not cause shuffling, thus adhering to the data locality principle. Step 5 stores the output of the transformations to HDFS in text format.

4.2 ETL Scenario 2: Predefined Time Period Aggregation

Scenario 2 refers to a predefined time period aggregation. It is present through aggregating the data for nominal or dynamically quantized column domain (e.g., user, campaign, asset) in conjunction with some predefined time period. Associating the aggregated records with the actual records that comprise them, allows drilling down. For example, if suddenly a spike in the number of daily signups happens, the change can be quickly validated by checking logs to see who signed up and when. Such functionality is not always possible with traditional dashboards, as they do not maintain the data source that is used to calculate the metrics. Another use of aggregated data is for concept drift detection, trend analysis over extended periods, or feature engineering [19].

The corresponding steps are shown in Figure 2, flow b. Steps 1 to 4 are like in scenario 1. The GroupByKey operation handles records with the same grouping key in step 5. Step 6 aggregates records within the same group, thus producing a new record with one or more aggregate values (e.g., count, sum). For each such new record, all records that comprise it are also preserved. Step 7 extracts the aggregated records (with the Map or FlatMap operations), generates primary keys for them with the same method as applied in step 3 and updates the aggregated records to reflect primary keys. It also sets the foreign key to the new record in all records that comprise it. Step 8 stores new records (without the comprising records) on HDFS. Step 9 extracts the comprising records of each new record with the FlatMap operation in one set of the unaggregated records. Then these records are stored to HDFS in step 10.

4.3 ETL Scenario 3: Session-based Aggregation

Aggregation on predefined time periods still does not cover all use-cases. For instance, consider the task of feature engineering. In many applications, e.g., churn prediction and fraud detection, meaningful features may be defined as: "the

time that passed from the last occurrence of event X", "the time since the user's last login", "last use of a service", or "last bought product". Similar attributes could be utilized in other data mining applications, such as identifying reasons for service outages or even predicting them. Even though such features are easy to understand, their calculation requires to look in a variable, practically unlimited data periods [20]. On the other hand, in typical streaming scenarios, only the very recent data portions are accessible.

Scenario 3 calculates user sessions, performs aggregation on session level and loads the data in both aggregated and unaggregated formats. We show ETL for this scenario in Figure 2, flow c. Incomplete sessions are defined as those that were still active at the end of the period that is being processed. Records of incomplete sessions are then stored separately so that they could be taken into account in the next run. Step 3 reads the data corresponding to incomplete sessions. Step 4 merges two datasets – new and incomplete data. In step 5, the data is cleansed and transformed, and foreign keys to dimensions are set. In step 6, the records are grouped by a more coarse grouping key, such as the user id. This enables implementation of complex business rules in step 7 for determining user sessions because all recent user records are available sequentially in one logical and physical location. During step 7, full sessions are determined, along with the records that comprise them and some aggregations are performed per session. Step 8 extracts full sessions with the FlatMap operator, generates primary keys for them and updates the records to reflect the generated keys: aggregated to have proper primary keys and comprising records to have proper foreign keys to the corresponding aggregate (session) records. The result of this step is preserved in memory as it will be needed three times in the following steps. Step 9 stores full sessions to HDFS. Using the result from step 8, Step 10 combines the unaggregated records that comprised completed sessions, and generates and sets their primary keys. In step 11, these records are stored to HDFS. Step 12 extracts the unaggregated records of incomplete sessions into one set and then step 13 stores them in the original format (without any data cleansing and transformations) in S3 so they can be used in the next run in step 3.

4.4 Data Load Steps for All Scenarios

After the last step described in each scenario, Data Load steps are executed. First, the data is loaded to DWH, using the proposed distributed data load algorithm that processes one table at a time in a parallel way. Finally, all metadata that was collected during the cluster lifetime (i.e., various metrics such as duration of each step, the number of processed records per table, etc.) is loaded into DWH and then the cluster self-terminates.

5 Experimental Results

Let us present the results of evaluation of three ETL scenarios with the proposed architecture. Table 1 shows information about the considered datasets.

Table 1: Statistics on datasets and generated records in each ETL scenario

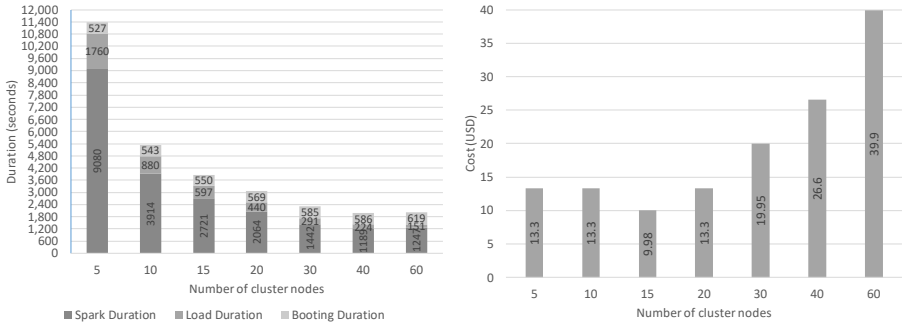
	ETL scenario			
	ETL1	ETL2	ETL3	ETL3 (100 days)
Source type	CSV	JSON	JSON	JSON
Source columns	31	17	17	17
Destination aggregated columns	-	86	86	86
Destination unaggregated columns	86	26	26	26
Source S3 objects	550	410K	410K	36M
Source size (GB)	53	30	30	2603
Source records	137M	44M	46M	3987M
Destination unaggregated records	137M	44M	44M	3985M
Destination unaggregated size (GB)	94	28	28	2427
Destination aggregated records	-	2M	1M	108M
Destination aggregated size (GB)	-	2	1	70

The data was provided from a service that collects user logs very frequently, and the processing result was timely and actionable information. All three scenarios were used to populate data marts that we designed for a subscription video-on-demand company that was competing with Netflix in their local market. First, decision support systems leveraged the aggregated data for evaluating investment opportunities and tracking historical performance. ETL scenarios 2/3 were applied for feature engineering to build machine learning systems for: churn prediction, fraud detection (i.e., account sharing against the terms of use), and predicting service outages. Finally, we preprocessed the log data to infer implicit user feedback, in order to build a recommendation system.

The experiments with each of the scenarios were repeated ten times and all presented times represent the average of the repetitions. All scenarios were evaluated on clusters with 5, 10, 15, 20, 30, 40 and 60 nodes so that we could investigate the impact of cluster size on the speedup.

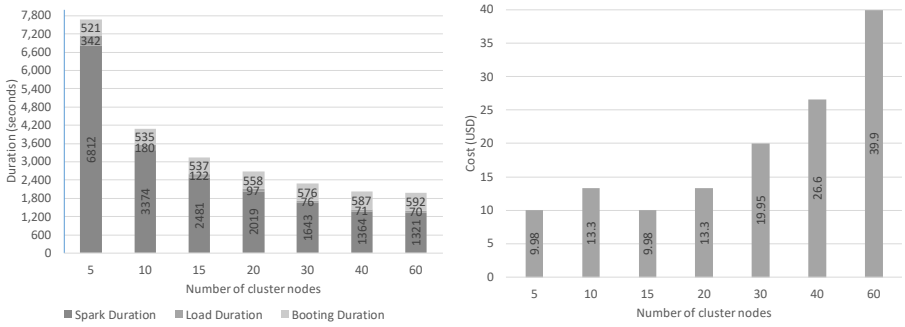
ETL scenario 1 We experimented with the whole data stored in one large text file, as well as 550 smaller text files (see Table 1, column ETL1). The size of the source files did not influence the performance of the system, which is understandable, considering that S3 is a distributed storage system. The performance of each step (Spark duration and DLA duration) depending on cluster size is shown in Figure 3a. Note that Amazon does not bill the booting duration. Similarly, the cost depending on cluster size is shown in Figure 3b. It is evident that the 15-node cluster was the most cost-effective because its chargeable duration is just under one hour. It is also notable that when we used more than 30 nodes, the overall duration did not improve significantly.

ETL scenario 2 Duration of each step and the cost for this case study (Table 1, column ETL2) depending on cluster size is shown in Figures 4a and 4b, respectively. Obviously, the 5-node and 15-node clusters are the cheapest. However, the latter completes the job faster for the same cost.



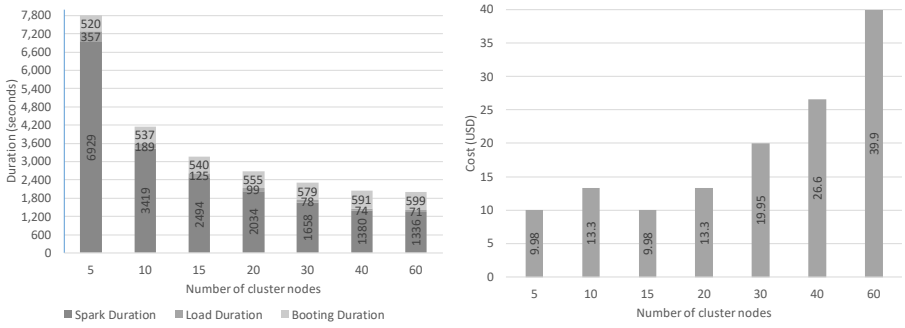
(a) Duration (b) Cost

Fig. 3: Duration of each step (a) / cost (b) for ETL scenario 1 per cluster size



(a) Duration (b) Cost

Fig. 4: Duration of each step (a) / cost (b) for ETL scenario 2 per cluster size



(a) Duration (b) Cost

Fig. 5: Duration of each step (a) / cost (b) for ETL scenario 3 per cluster size

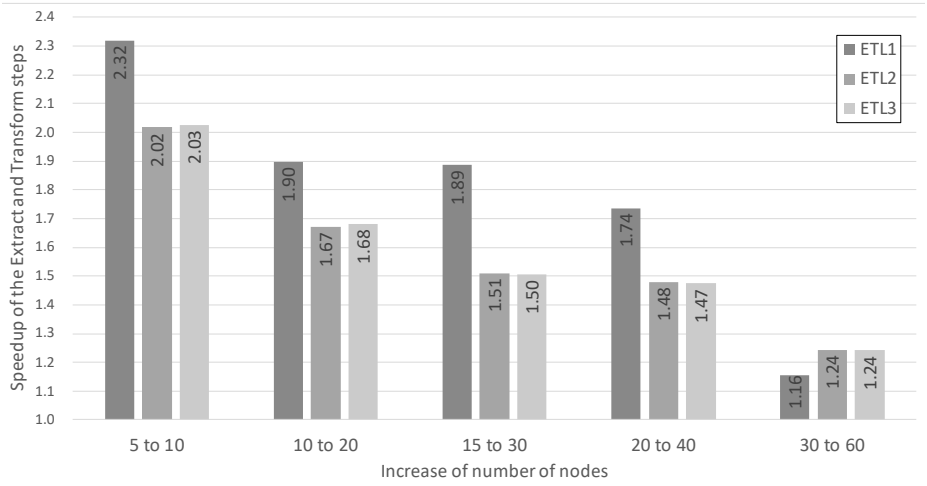


Fig. 6: Speedup of Extract-Transform steps (Spark) of the three ETL scenarios.

ETL scenario 3 The results of the experiments with the third scenario, which performs session-based aggregation, are shown in Figures 5a and 5b. As before, the most cost-effective is the 15-node cluster.

To verify that our architecture is reliable and sustainable, we executed scenario 3 (the most complex one) on a considerably increased workload using the data collected during 100 days (Table 1, column ETL3 (100 days)). We used a 20-node cluster with "r3.2xlarge" instances. Considering that the volume of source data (2.6 TB) exceeds the cluster's storage capacity ($20 \times 160 = 3.2$ TB total hard drive space, of which less than 1 TB is available for HDFS), the Spark and DLA jobs were executed interchangeably one day at a time (i.e., flow c shown in Figure 2 was executed 100 times on the same cluster). Execution in a one-day-at-a-time fashion also enabled the results of the ETL to be available even though the whole process is still in progress. The Spark jobs completed in 174,481 seconds in total, or on average about 1,745 seconds per daily data volume. This is considerably less than when a cluster of same size processes daily data (2,034 seconds, see Figure 5a). We attribute these savings to the overhead of starting a Spark job on a new cluster and to the variance in daily data volumes. DLA completed in 8,514 seconds, an increase which is linearly proportional to the processed data volume. Figure 6 shows the obtained speedup of Extract-Transform steps in Spark when comparing different cluster sizes for the three ETL scenarios, which is based on the results reported in Figures 3a-5a. Obviously, as the number of nodes increases, the speedup decreases.

6 Conclusions

We proposed a cloud-based architecture for efficient ETL of Big Data. Spark performs Extract-Transform phases. Then the results are loaded into a data warehouse using distributed load agents that utilize the processing resources of the cluster slaves (edge nodes), instead of the database server. To that end, ETL employs on-demand Hadoop clusters with a variable size that run for a limited duration on Amazon AWS. By defining and evaluating three ETL scenarios that cover a variety of use cases, we demonstrated the scalability of our solution. Most notable was the non-trivial usage of the proposed scenarios for feature engineering in the considered data mining applications.

Having such run-time facilities, one can think about automatizing ETL's design too. Elemental data analysis (file formats, data types, measure units), data model recovery, dimensional model identification, are activities that at least to some extent can be performed by a computer program. Our initial experiments are promising. We believe that the full ETL effort from the design to a running data warehouse can be limited to days instead of months.

References

1. Apanowicz, C.: Data Warehouse Discovery Framework: The Foundation. In: Proceedings of International Conferences on Database Theory and Application (DTA 2010) and Bio-Science and Bio-Technology (BSBT 2010), Held as Part of Future Generation Information Technology Conference (FGIT 2010). Volume 118 of Communications in Computer and Information Science., Springer (2010) 142–154
2. Apanowicz, C.: Data Warehouse Discovery Framework: The Case Study. In: Proceedings of International Conferences on Database Theory and Application (DTA 2010) and Bio-Science and Bio-Technology (BSBT 2010), Held as Part of Future Generation Information Technology Conference (FGIT 2010). Volume 118 of Communications in Computer and Information Science., Springer (2010) 155–166
3. Chaudhuri, S., Dayal, U., Narasayya, V.: An Overview of Business Intelligence Technology. Communications of the ACM **54**(8) (2011) 88–98
4. Mukherjee, R., Kar, P.: A Comparative Review of Data Warehousing ETL Tools with New Trends and Industry Insight. In: Proceedings of IEEE 7th International Advance Computing Conference (IACC 2017). (2017) 943–948
5. Zdravevski, E., Lameski, P., Kulakov, A.: Row Key Designs of NoSQL Database Tables and Their Impact on Write Performance. In: Proceedings of 24th Euromicro International Conference on Parallel, Distributed, and Network-based Processing (PDP 2016). (2016) 10–17
6. Mallek, H., Ghozzi, F., Teste, O., Gargouri, F.: BigDimETL: ETL for Multidimensional Big Data. In: Proceedings of 16th International Conference on Intelligent Systems Design and Applications (ISDA 2016). Volume 557 of Advances in Intelligent Systems and Computing., Springer (2017) 935–944
7. Condie, T., Conway, N., Alvaro, P., Hellerstein, J.M., Elmeleegy, K., Sears, R.: MapReduce Online. In: Proceedings of 7th USENIX Conference on Networked Systems Design and Implementation (NSDI 2010), USENIX Association (2010) 21–21

8. Hellerstein, J.M., Ré, C., Schoppmann, F., Wang, D.Z., Fratkin, E., Gorajek, A., Ng, K.S., Welton, C., Feng, X., Li, K., Kumar, A.: The MADlib Analytics Library or MAD Skills, the SQL. Proceedings of the VLDB Endowment **5**(12) (2012) 1700–1711
9. Low, Y., Bickson, D., Gonzalez, J., Guestrin, C., Kyrola, A., Hellerstein, J.M.: Distributed GraphLab: A Framework for Machine Learning and Data Mining in the Cloud. Proceedings of the VLDB Endowment **5**(8) (2012) 716–727
10. Boja, C., Pocovnicu, A., Batagan, L.: Distributed Parallel Architecture for "Big Data". Informatica Economica **16**(2) (2012) 116
11. Chevalier, M., El Malki, M., Kopliku, A., Teste, O., Tournier, R.: How Can We Implement a Multidimensional Data Warehouse Using NoSQL? In: Proceedings of 17th International Conference on Enterprise Information Systems (ICEIS 2015). Volume 241 of Lecture Notes in Business Information Processing., Springer (2015) 108–130
12. Liu, X., Thomsen, C., Pedersen, T.B.: CloudETL: Scalable Dimensional ETL for Hive. In: Proceedings of 18th International Database Engineering Applications Symposium (IDEAS 2014), ACM (2014) 195–206
13. Souissi, S., BenAyed, M.: GENUS: An ETL tool treating the Big Data Variety. In: Proceedings of IEEE/ACS 13th International Conference on Computer Systems and Applications (AICCSA 2016). (2016) 1–8
14. Li, X., Mao, Y.: Real-time Data ETL Framework for Big Real-time Data Analysis (ICIA 2015). In: Proceedings of IEEE International Conference on Information and Automation. (2015) 1289–1294
15. Pääkkönen, P., Pakkala, D.: Reference Architecture and Classification of Technologies, Products and Services for Big Data Systems. Big Data Research **2**(4) (2015) 166–186
16. Singh, K., Guntuku, S.C., Thakur, A., Hota, C.: Big Data Analytics Framework for Peer-to-peer Botnet Detection Using Random Forests. Information Sciences **278** (2014) 488–497
17. Bai, J.: Feasibility Analysis of Big Log Data Real Time Search based on Hbase and ElasticSearch. In: Proceedings of 9th International Conference on Natural Computation (ICNC 2013), IEEE (2013) 1166–1170
18. Ślęzak, D., Synak, P., Wojna, A., Wróblewski, J.: Two Database Related Interpretations of Rough Approximations: Data Organization and Query Execution. Fundamenta Informaticae **127**(1-4) (2013) 445–459
19. Zdravevski, E., Lameski, P., Kulakov, A., Gjorgjevikj, D.: Feature Selection and Allocation to Diverse Subsets for Multi-label Learning Problems with Large Datasets. In: Proceedings of Federated Conference on Computer Science and Information Systems (FedCSIS 2014), IEEE (2014) 387–394
20. Ślęzak, D., Grzegorowski, M., Janusz, A., Kozielski, M., Nguyen, S.H., Sikora, M., Stawicki, S., Wróbel, Ł.: A Framework for Learning and Embedding Multi-Sensor Forecasting Models into a Decision Support System: A Case Study of Methane Concentration in Coal Mines. Information Sciences **451-452** (2018) 112–133