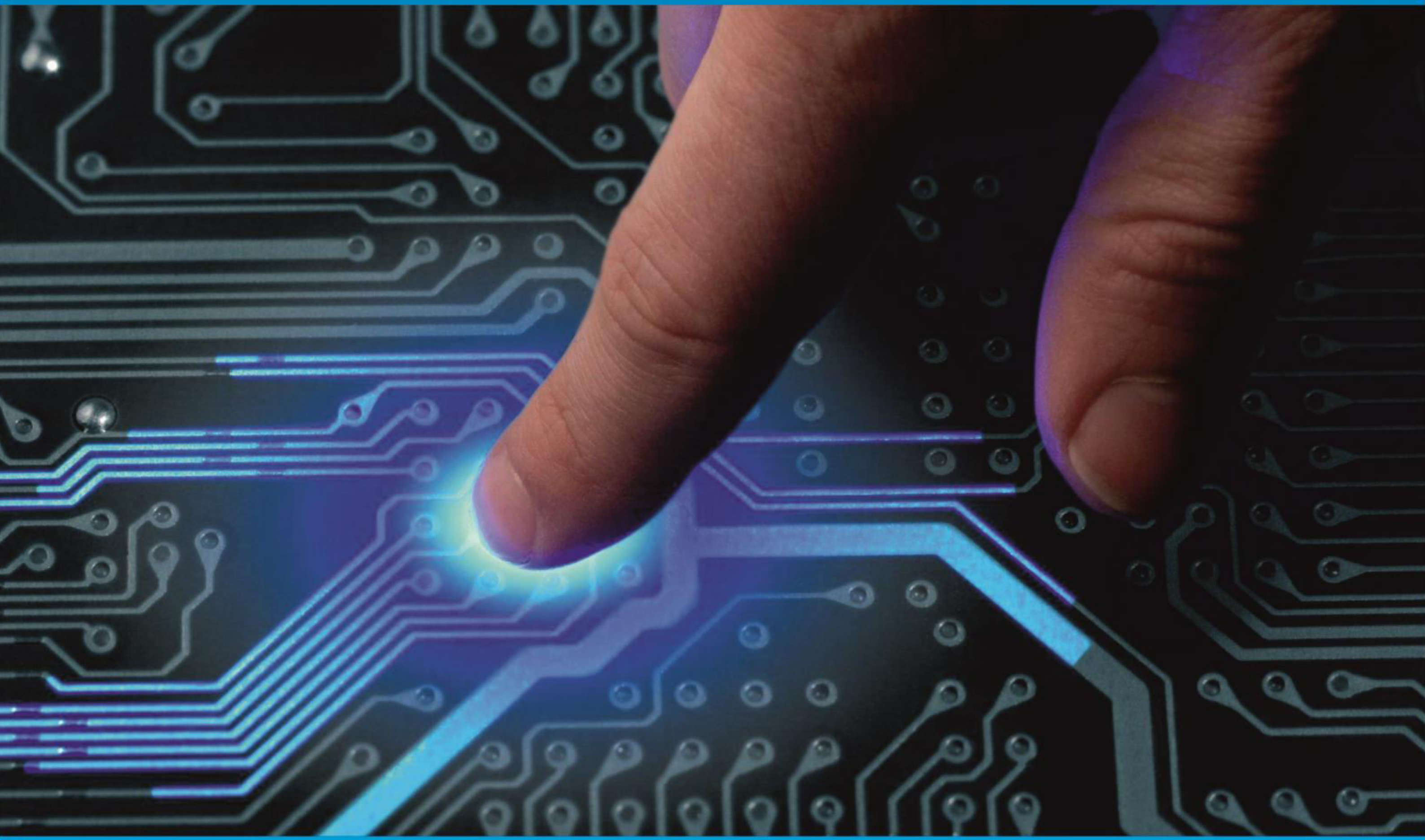
e-ISSN: 2320-9801 | p-ISSN: 2320-9798



INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN COMPUTER & COMMUNICATION ENGINEERING

Volume 9, Issue 3, March 2021

Impact Factor: 7.488

International Journal of Innovative Research in Computer and Communication Engineering

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 7.488 || A Monthly Peer Reviewed & Referred Journal |

|| Volume 9, Issue 3, March 2021 ||

| DOI: 10.15680/IJIRCCE.2021.0903222 |

Designing a Scalable Cloud-based ETL Framework for Big Data Aggregation and Feature Engineering

Archana Bhat[1], Laxmi M C[2], Sriraksha S[3], Sreevidya G[4], Harish R[5]

Assistant Professors, Department of Computer Science and Engineering, City Engineering College, Bengaluru, Karnataka, India[1,2,3,4,5]

ABSTRACT: The success of businesses largely hinges on their ability to effectively analyze data and derive actionable insights. The Extract-Transform-Load (ETL) process plays a crucial role in achieving these objectives, but it demands considerable effort, particularly when dealing with Big Data. Previous research has struggled to formalize, integrate, and assess the ETL process for Big Data challenges in a scalable and cost-efficient manner. In this paper, we introduce a cloud-based ETL framework designed for the fusion and aggregation of data from various sources. We then outline three scenarios for data aggregation during the ETL process: (i) ETL without aggregation; (ii) aggregation based on specified columns or time intervals; and (iii) aggregation within individual user sessions over varying time frames. The third scenario is particularly beneficial for feature engineering, enabling the definition of features such as "the time since the last occurrence of event X." We assessed the scalability of our framework using Amazon AWS Hadoop clusters.

KEYWORDS: ETL (Extract-Transform-Load), Big Data, Cloud Computing, Data Aggregation, Feature Engineering

I. INTRODUCTION

The prevalence of smart devices, sensors, and social media has resulted in massive amounts of data being generated. Concurrently, consumers have become accustomed to receiving personalized services instantly. Numerous companies, healthcare providers, and institutions have dedicated substantial resources to meet this demand. In the era defined by Big Data, organizations are increasingly under pressure to store and analyze all collected data to maintain a competitive edge in the data-driven market. The intrinsic characteristics of Big Data include volume, velocity, and variety. Recently, other aspects such as variability, veracity, visualization, and value have been recognized as equally important. Together, these elements represent the 7 Vs of Big Data, highlighting the significant complexity faced by those who seek to process, analyze, and leverage this information. Additionally, companies encounter challenges with data analysis outcomes such as joins, transformations, and aggregations—as well as the integration of data with other system components.

To present the data in a practical format, several essential steps must be undertaken [5, 6]: conducting analyses and modeling to uncover all relationships and the relevant business context; gathering data; and performing Extract-Transform-Load (ETL) processes, which typically involve lengthy development and execution phases. After the data is processed and integrated into a data warehouse, it should be accessible for reporting, visualization, analytics, and decision support purposes [1]. According to [7], data warehousing (DW) and business intelligence (BI) face numerous challenges regarding Big Data, including issues related to size, complexity, design and data modeling, computing methodologies, query languages, usability, end-user performance, data consistency and lineage, as well as adapting traditional tools for data exploration, visualization, and analytics and their integration into existing DW/BI solutions and platforms. As proposed in [7], future research directions concerning DW and OLAP in the context of Big Data should strive to provide se...

The usability of data throughout various processing stages is closely linked to its consistency, which presents substantial challenges within Big Data systems. Adopting strong consistency models can severely hinder the scalability and performance of these systems. Conversely, weak and eventual consistency models allow for greater availability and reduced latencies, but they may notably detract from the quality of the information obtained and diminish its usability. In the context of cloud computing, which has become a key paradigm offering a range of cost-effective hardware and software solutions well-suited for deploying Big Data systems, new challenges also arise. These include architectural

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 7.488 || A Monthly Peer Reviewed & Referred Journal |

|| Volume 9, Issue 3, March 2021 ||

| DOI: 10.15680/IJIRCCE.2021.0903222 |

design, cost and performance optimization, assurance of reliability, security, as well as maintaining privacy and data consistency. The distributed nature of Big Data intensifies these issues, necessitating more sophisticated synchronization, replication, and scheduling mechanisms.

II. RELATED WORK

Traditional Business Intelligence (BI) relies on ETL (Extract, Transform, Load) tools such as Informatica, IBM Infosphere DataStage, Ab Initio, Microsoft SQL Server Integration Services (SSIS), Oracle Data Integrator, Talend, and Pentaho Data Integration Platform (PDI) to load data into warehouse servers. Recently, Enterprise Application Integration (EAI) systems have started to replace ETL tools, offering a broader range of functionalities. Another category of tools is "Extract-Load-Transform" (ELT), which are typically promoted by database vendors. ELT tools defer transformations until later, leveraging the power of database engines for this purpose. ELT is particularly useful when business requirements change frequently, as it allows data to be loaded before transformations are applied. A review in [11] examines traditional ETL and ELT tools, focusing on their terminology and capabilities, but it does not significantly address the challenges associated with Big Data.

Apache Spark builds on concepts from the MapReduce paradigm, focusing on applications that reuse a working set of data across multiple parallel operations. The work discussed in [12] highlights the similarities between ETL and MapReduce processes, noting that traditional ETL deployments often lack data partitioning. In their approach, data is partitioned during the Map step and aggregated in the Reduce step. This method demonstrated scalability with static datasets up to 300 GB but does not address dimension matching from fact tables. Our approach not only tackles this issue but also supports high-velocity data processing in production environments.

Consistent data warehouses enable the use of traditional visualization, reporting, and business intelligence services, as well as simplify data mining and machine learning tasks. The BigDimETL approach [13] seeks to maintain the multidimensional structure of data warehouses while integrating Big Data. However, this work is theoretical and lacks experimental evaluation. The distributed parallel architecture for ETL of Big Data proposed in [14] requires ETL completion before data aggregation, whereas our method performs aggregation during the ETL process.

An approach in [15] describes rules for translating conceptual designs and mapping star schemas to NoSQL logical models with precomputed aggregate lattices. Like our approach, this method requires predefined aggregate metrics for ETL computation. However, while this paper focuses on model generation rules, we concentrate on systematizing ETL scenarios and developing distributed load agents to evenly distribute the load on demand clusters before reaching the data warehouse.

Customer churn is a prevalent issue for many businesses, and while advanced techniques have been applied to churn prediction for years, achieving significant breakthroughs in prediction accuracy is becoming increasingly challenging. Research in [21] demonstrates that utilizing high-volume training data, which includes diverse features from business and operations support systems, alongside high-velocity new data, can markedly enhance prediction performance. This improved prediction capability can be leveraged to design effective retention campaigns targeting potential churners.
Inspired by this work in the telecommunications industry, we applied similar techniques to the video streaming sector and reached comparable conclusions. Additionally, we used Local Interpretable Model-agnostic Explanations (LIME) [22] to explain predictions, aiding customer sales teams in their retention efforts. A notable distinction in our approach is the automation of feature generation: rather than manually creating potentially valuable features, we utilized the ETL capabilities of our architecture to automatically produce a range of features through various aggregations.

Table 1 highlights the differences between related works that propose various approaches to ETL for Big Data. Our approach, marked in the table, demonstrates significant improvements in both capabilities and experimental evaluation compared to other methods. The last two rows provide references for applying Spark Streaming, Spark, Pig, or MapReduce directly, without the proposed framework.

III. METHODS

Organizations typically handle traditional data sources like relational database management systems (RDBMS) and structured or semi-structured data from internal or third-party providers, which generate reasonably-sized datasets. To

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 7.488 || A Monthly Peer Reviewed & Referred Journal |

|| Volume 9, Issue 3, March 2021 ||

| DOI: 10.15680/IJIRCCE.2021.0903222 |

process this type of data, companies often use traditional Data Integration Tools. In our experiments, we utilized the Pentaho Data Integration Platform (PDI) for ETL tasks, managing low-volume incoming data and storing it in the Data Warehouse (indicated by light gray arrows in Figure 1). We selected PDI due to its ability to ingest, blend, cleanse, and prepare diverse data from various sources without requiring extensive coding. Its visual tools simplify the creation of data pipelines, and it offers broad connectivity options, including flat files, RDBMS, Hadoop, Spark, NoSQL data stores, and analytic databases, along with robust orchestration, scheduling, and agile data modeling capabilities.

For data producers generating Big Data with high volume, velocity, or versatility, traditional ETL approaches are inadequate. Instead, Distributed Streaming Platforms (DSPs), such as Apache Kafka and Amazon Kinesis, are more suitable for efficiently collecting and processing these data streams. DSPs are scalable, replicated, and fault-tolerant. They allow data retention on the queue for a defined period after publication, regardless of consumption status; for instance, Amazon Kinesis currently allows a maximum retention period of one week. DSPs enable multiple consumers to access the same data stream independently and simultaneously, each at its own pace. Data access on a DSP queue can be managed through either push or pull mechanisms, with the pull mechanism being inherent to systems like Amazon Kinesis and Apache Kafka, where each consumer manages its own read pointer.

3.1 Push Lambda Functions (Stream-based model)

In this setup, event sources publish events to the Distributed Streaming Platform (DSP), which triggers lambda functions multiple times per second as data arrives on the queue. The lambda functions are configured to automatically read batches of records from the DSP queue and process them as they appear in the stream. They poll the queue periodically, up to a few times per second, to check for new records. Typically, lambda functions require infrequent updates since they store unprocessed raw data in its original format on Object Storage Services (OSS) such as Amazon S3 or Windows Azure Blob Storage. However, when updates are necessary, they can cause minimal downtime. Updates are most commonly made to enhance error handling capabilities in response to unparsable or illegal input from event sources.

3.2 Storage Stream Pullers

These consumers have greater control over fetching records from DSPs because they independently manage their read pointers, allowing them to reprocess events if necessary (e.g., for recovery after failures). However, this management is more complex as the uptime of the machines running the storage stream pullers must be ensured through manual mechanisms, since Amazon AWS does not provide a guarantee. This approach offers an alternative for durable and persistent storage of raw data, with flexibility in the frequency of data persistence.

3.3 Analytics Stream Pullers

The first two types of consumers provide redundant yet reliable alternatives for storing incoming data in its raw format across different S3 buckets. Each alternative offers a high level of reliability with guaranteed Service Level Agreements (SLAs), and either one alone is sufficient for the proposed architecture. Using both alternatives can simplify deployment procedures and enhance system reliability. Specifically, if there are significant changes to the data format or sources, the consumers can be updated without any downtime or risk of data loss. Additionally, having both alternatives provides integration benefits with existing organizational infrastructure.

To enhance data veracity and complement data warehouse analytics, full-text search capabilities can be beneficial. Services like Elasticsearch or Solr [27] can be used to cleanse raw data stored in Object Storage Services (OSS) by extracting plain text from HTML files and indexing it for full-text search functionalities, as illustrated in Figure 1. On the other hand, Analytics Stream Pullers, such as Apache Spark Streaming and Apache Storm, process streaming data to offer near real-time insights and analytics. These technologies allow for the implementation of complex algorithms to derive valuable business insights and can also store data in the warehouse. However, due to data retention policies, Analytics Stream Pullers are limited to analyzing only recent data.

To address this limitation, the proposed architecture incorporates on-demand Spark clusters for executing more complex ETL and feature engineering algorithms. These clusters can analyze dynamic trends over extended periods (e.g., weekly or monthly comparisons of various metrics) or compute metrics related to the time since specific events occurred, which is not feasible with Analytics Stream Pullers. The Cluster Launcher module, which is located on the same instance as the Data Integration Tool (DIT), manages the initiation of on-demand Spark clusters. It can be triggered manually or according to a schedule set by DIT. The Cluster Launcher starts a configurable Hadoop cluster, runs a specific Spark job, and downloads the necessary source code from a release branch of a code repository (e.g., git, mercurial, subversion, FTP server, S3, or Azure Blob Storage). Code development and management follow the

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 7.488 || A Monthly Peer Reviewed & Referred Journal |

|| Volume 9, Issue 3, March 2021 ||

| DOI: 10.15680/IJIRCCE.2021.0903222 |

organization's adopted strategy, such as GitFlow [28], which outlines rules and best practices for conflict resolution, peer reviews, and merging to staging and production branches. Each Spark cluster is dedicated to executing a specific ETL job during its operation.

3.4 ETL Data-flow Scenarios

In this section, we describe three common ETL data-flow scenarios needed by organizations, focusing on data stored in fact tables, which constitutes the major portion of the data warehouse. Dimensional data, being much smaller in volume, typically does not require Big Data technologies for processing; traditional ETL tools are usually sufficient for this type of data. The proposed architecture assumes that traditional ETL tools handle dimensional data and that the primary focus is on processing data for fact tables. The ETL process for the three scenarios is outlined in Figure 3, which details the implementation steps.

Business keys are unique identifiers with business significance managed by operational data stores (ODS), but in a data warehouse, surrogate keys are used as generalizations of these business keys. Surrogate keys are essential in data warehouse design and facilitate joins between dimension and fact tables. The data extraction logic must systematically replace incoming business keys with surrogate keys to maintain independence from ODS business keys, which may be subject to changes. For large datasets, populating foreign keys in fact tables traditionally through joins is inefficient due to the need for data shuffling across cluster nodes. Instead, distributing dimension business and surrogate keys across nodes beforehand allows for efficient dictionary lookups with O(1) complexity, adhering to the data locality principle. Figure 3 illustrates the data flow for the three scenarios, with Spark loading and distributing business and surrogate keys to each node in the first step, a process common to all scenarios.

3.5 ETL scenario 1: No aggregation

The first scenario of ETL requires parsing, data type conversion, setting foreign keys (Extract and Transform steps) and only loading into the fact tables of a data warehouse. This scenario is the simplest of the three and does not require any aggregations in the fact tables. Such use-cases require that all generated data be available at the lowest level of granularity (after performing proper cleansing), as well as their associations with other entities in the system. Some typical use-cases of this work-flow refer to the log analysis in the following application fields:

3.6 Resource Management

Monitoring across systems to detect particular log events and patterns in the data; Identifying performance or configuration issues; Meeting operational objectives and SLAs; etc.

3.7 Application Troubleshooting

Diagnosing and identifying root causes of application installation and run-time errors; Pinpointing areas of poor performance; Assessing application health and troubleshooting.

3.8 Marketing Insights

Gathering insights and analyzing their impact on visibility, traffic, conversions, and sales; Revealing potential improvements of Search Engine Optimization (SEO); Understanding which pages are useful and useless; etc.

3.9 Regulatory Compliance and Security

Maintaining compliance with industry regulations often requires all data to be preserved in source format, while tagging and identifying certain events in data.

3.10 ETL scenario 2: Predefined time period aggregation

The second ETL scenario involves aggregation over a predefined time period. This scenario is commonly used for aggregating data related to nominal or dynamically quantized column domains (e.g., user, campaign, asset) over a specified time frame. It extends the first scenario by analyzing granular data, extracting key pieces of information (such as response times, customer plan values, or performance metrics), and then rolling these into aggregated records for display on metrics dashboards. This aggregation allows for detailed analysis, such as drilling down into spikes in daily signups by examining log records to identify who signed up and when. Unlike traditional metrics dashboards, which often lack the source data used for metrics calculations, this approach maintains a connection to the underlying records. Additionally, aggregated data can be used for detecting concept drift, conducting trend analysis over extended periods, or engineering features for machine learning algorithms [6, 19].

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 7.488 || A Monthly Peer Reviewed & Referred Journal |

|| Volume 9, Issue 3, March 2021 ||

| DOI: 10.15680/IJIRCCE.2021.0903222 |

Determining the necessary aggregate metrics requires significant manual effort from data warehouse architects [5, 6]. To address this, a conventional approach is to generate a broad range of potentially useful metrics and later use statistical and feature selection methods to identify the most relevant ones. Since the importance of metrics can change over time, generating a diverse set of metrics upfront, as suggested in [15], is advantageous. In Figure 3b, the steps for the second ETL scenario are illustrated. Steps 1 to 4 mirror those of the previous scenario, involving data transformations, cleaning, and setting primary and foreign keys. In step 5, records are grouped using the GroupByKey operation based on the grouping key, ideally a time dimension column (e.g., hourly or daily aggregations). This recommendation is due to the use of the Infobright data warehouse in the proposed architecture, which supports efficient ad-hoc queries and aggregations due to its knowledge grids. Step 6 involves aggregating the records within each group to produce aggregate records with one or more values (e.g., count, sum, min, max, or custom aggregate functions). Importantly, each aggregate record retains the original records that comprise it, ensuring no data is lost during the aggregation process.

IV. RESULTS

To evaluate the proposed architecture, Amazon Web Services (AWS) was chosen due to its widespread popularity in both industry and research communities, as well as the hardware heterogeneity offered by its various instance types. Although AWS was used for the experiments, the architecture could be easily replicated on other cloud providers like Windows Azure or Google Cloud, as they offer similar services. Additionally, on-premises infrastructures could also serve as suitable alternatives to the AWS components utilized. Among the available instance families on AWS, the “r3.2xlarge” instance type was selected for our experiments due to its high network performance and finer control over cluster resources compared to larger instances**.**

For the data warehouse, Infobright DB was chosen because of its fast ad-hoc queries, lack of need for physical database design (indexing, partitioning, tablespaces design, etc.), and extremely fast load process that can scale to terabytes per hour. Additionally, it supports multiple parallel loads to a single table and has a very small database footprint. The data warehouse was installed on an “r3.8xlarge” instance, which is equipped with 244 GB of RAM, 32 CPUs, a 640 GB HDD, and a 10 Gigabit network. The data used to evaluate the framework was collected from various devices, such as smartphones, websites, and desktop applications, which published events to an Amazon Kinesis stream with five shards in the “us-east-1” region. Events on the stream triggered a Python Lambda function up to five times per second per shard, storing raw JSON records in plain text files on S3. The producers generated up to 1,000 records per second per Kinesis shard, allowing a maximum of 5,000 events per second (432 million events per day) to be processed and stored in up to 2.16 million S3 objects daily.

4.1 Evaluation of the ETL scenarios

All three proposed scenarios were used to populate several data marts for different purposes, most notably a churn prediction system, a recommendation system and a frauddetection system. The first ETL scenario is evaluated with structured data that only requires cleansing, transformation, surrogate keys generation, and setting up foreign keys to dimensions. The second and third ETL scenarios additionally required aggregations, and they were evaluated on user logs data collected during a typical workday. For the ETL scenario 1, which does not require aggregation, we experimented when all data was stored in one large text file, as well as in 550 smaller text files (see Table 2, 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 DDL duration) depending on cluster size is shown in Figure 5a. Note that Amazon does not bill the booting duration. Similarly, the cost depending on cluster size is shown in Figure 5b. From these two figures, 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 more than 30 nodes were used, the overall duration did not improve significantly

To verify that the proposed architecture is reliable and sustainable, we have executed the third scenario (as it is the most complex of the three) on a considerably increased workload using data collected during 100 days (see Table 2, column ETL3 (100 days)). We used a 20-node cluster with “r3.2xlarge” instances. Considering that the volume of the source data (2.6 TB) exceeds the storage capacity of the cluster ($20 \times 160 = 3.2$ TB) total hard drive space, of which less than 1 TB is available for HDFS), the Spark and DDL jobs were executed interchangeably one day at a time (i.e., the flow shown in Figure 3c was executed 100 times on the same cluster). This also enabled the results of the ETL to be available even though the whole process is not fully completed. The Spark jobs completed in 174,481 seconds in total, or on average, about 1,745 seconds per day’s data. This is considerably less than when a cluster of the same size processed the daily data (2,034 seconds, see Figure 7a). We attribute these savings to the overhead of starting a Spark

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 7.488 || A Monthly Peer Reviewed & Referred Journal |

|| Volume 9, Issue 3, March 2021 ||

| DOI: 10.15680/IJIRCCE.2021.0903222 |

job on a new cluster and initialization of dimensional data key lookup dictionaries, and to the variance in daily data volumes. DDL completed in 8,514 seconds, an increase which is linearly proportional to the processed data volume. Figure 8 shows the obtained speedup of the Extract and Transform steps in Spark when comparing different cluster sizes for the three ETL scenarios, which is based on the results provided in Figures 5a, 6a and 7a. Obviously, as the number of nodes increases, the speedup decreases.

4.2 Business Analytics application: Customer churn prediction

Using the approach outlined in subsection 3.3.1, the training, validation, and test datasets were regenerated every two weeks, focusing only on active users within the relevant time period. During the retrospective period evaluated in this study, each dataset—training, validation, and test—contained approximately 200,000 subscribers, with around 10% of them having churned. The automated feature extraction process during the ETL phase generated about 2,000 features per subscriber, which were then reduced to around 350 during the validation phase. The test results, as shown in Table 3, highlight the top five algorithms that performed best using these 350 features. Notably, Support Vector Machines (SVMs) were excluded from the test experiments due to the significant time required for hyper-parameter tuning, which did not lead to improved scores.

It presents the information gain of the top 10 most informative features. This analysis revealed that user activity in the two weeks leading up to the prediction day was the most significant predictor. It was observed that users who did not engage with the service in the last two weeks were more likely to churn compared to those who watched one or more videos. Additionally, users who had subscribed for over two months and were still active showed a lower likelihood of cancelling their subscription. Even though the black-box ensemble models had superior predictive performance, the more-easily interpretable Decision Trees model provided easier interpretation. However, it was still over-whelming for production use, hence the LIME approximation was used. Figure 9 shows an exemplary LIME explanation of a “churns” and “remains” predictions made by XGBoost. In this example, only 5 features are shown.

V. DISCUSSION

This study proposed an architecture for efficient and scalable ETL of Big Data, utilizing distributed processing for the extraction and transformation steps with Apache Spark and a distributed load into a data warehouse. The architecture was evaluated across three common ETL scenarios: no aggregation, predefined time period aggregation, and session-based aggregation. While Amazon does not charge for the booting time required to provision the cluster, this time does impact the overall duration of the ETL process and should be taken into account. The booting time was found to be similar across both smaller and larger clusters, as illustrated in Figures 5a, 6a, and 7a. Repeated experiments for each scenario, conducted at least ten times, revealed instances where the booting time for 5-node clusters was even longer than that for 60-node clusters, likely due to varying resource availability in Amazon AWS. However, in hundreds of experiments involving clusters of up to 60 nodes using “r3.2xlarge” instances, this variance was minimal, with all clusters booting in under 15 minutes.

For clusters with 100 nodes or those requiring more powerful instance types than “r3.2xlarge,” the booting time was more variable, which is an important consideration when using on-demand clusters. In all three ETL scenarios, the distributed load into the data warehouse demonstrated near-linear scalability, as shown in Figures 5a, 6a, and 7a. However, in rare instances where the smallest cluster (5 nodes) was used, one of the nodes would occasionally fail. Fortunately, this did not result in data loss, thanks to HDFS replication. While this could have posed an issue for Distributed Load Applications (DLAs), the algorithm proposed ensured that the load of any particular data chunk, which is an atomic operation from the warehouse's perspective, was repeated by the first available node. Node failures were infrequent during Spark's execution, and no manual intervention was needed for recovery, as Spark automatically redistributed the failed tasks to other nodes using Hadoop and YARN facilities.

Despite the logical differences between the predefined time period aggregation ETL scenario and the session-based aggregation scenario, they are quite similar from Spark's data movement perspective. The session-based scenario includes only two additional steps (steps 12 and 13 in Figure 3c), which store the unaggregated data of incomplete sessions to S3. In the experimental dataset, about 5% of the sessions were incomplete, resulting in approximately 1.5 GB of data being stored in S3 for processing in the next run. Depending on the cluster size, it took Spark between 50 and 70 seconds to complete this S3 storage step, which was negligible compared to the overall ETL process duration. Similarly, the number of rows loaded into the warehouse in these two scenarios was almost identical (see Table 2), with only about one million aggregated rows' difference, which was insignificant (see Figures 6a and 7a).

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 7.488 || A Monthly Peer Reviewed & Referred Journal |

|| Volume 9, Issue 3, March 2021 ||

| DOI: 10.15680/IJIRCCE.2021.0903222 |

The Spark process benefited significantly from increasing the cluster size from 5 to 10 nodes, resulting in a super-linear speedup (see Figure 8). For the no-aggregation scenario, evaluated on a 53 GB dataset, the duration decreased from 9,080 to 3,914 seconds (see Figure 5a). In the other two scenarios, which were evaluated on a 30 GB dataset, the duration dropped from approximately 6,900 to 3,400 seconds (see Figures 6a and 7a). These super-linear speedups, particularly pronounced in the first scenario, were primarily due to the insufficient RAM on the 5-node clusters to store all data in memory.

For clusters with at least ten nodes, the RAM limitation was no longer an issue across all three scenarios. However, as the number of nodes increased, the speedup began to decline. The no-aggregation scenario showed better speedup compared to the aggregation scenarios when using up to 40 nodes, likely due to the simpler data flow. However, when the number of nodes increased to 60, performance actually worsened compared to using 40 nodes (see Figure 5a). To determine whether this was specific to this cluster size, we conducted experiments with the first ETL scenario using 100-node and even 200-node clusters. The results were even worse, likely due to the significant communication and synchronization overhead. This confirms that cluster size must be optimized for the data load, and that Big Data approaches are not well-suited for problems involving small amounts of data. From all the experiments and results, we can also conclude that there is a physical limit to how fast the ETL process can be completed. While we were able to reduce execution time to under one hour for all ETL scenarios, further increasing the number of nodes did not lead to additional reductions in execution time.

Although some metrics calculated during the ETL process were not informative for the churn prediction problem, they proved valuable for other business analytics. Features related to the number of dropped packets per operating system and user session provided critical insights into production issues with mobile applications, leading to the discovery of memory management issues and the sequence of actions that caused them. Additionally, these metrics were useful for predicting cancellation reasons, which were explored in a subsequent phase that classified predicted churners in a multi-label, multi-class setting. This, combined with the explained predictions using LIME, was particularly beneficial for the sales teams.

VI. CONCLUSION

In this paper, we have proposed a cloud-based architecture for efficient ETL of Big Data. The extract and transform phases are performed by Spark, and then the results are loaded into a data warehouse using distributed load agents (DLAs) that utilize the processing resources of the cluster slaves (edge nodes) reducing the workload on the database server. To that end, the ETL process utilizes 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 showed that the ETL process could be effectively used for feature extraction of non-trivial features that require processing of variable periods of past data. The proposed approach leverages the established ad-hoc, analytical, and integration capabilities of traditional data warehouses. In scenarios such as processing computer-generated logs, the number of rows can be significantly reduced through aggregations while still retaining sufficient information for a variety of ad-hoc queries. Pure Big Data solutions often present challenges in these cases, including longer development times, compatibility issues with visualization and reporting tools, and significant overhead when executing Mapreduce or Spark jobs. The true advantage of the proposed method lies in its combination of Big Data technologies for heavy lifting tasks like ETL and aggregation, with traditional data warehousing technologies for data exploration, such as analytics, reporting, and visualization. We demonstrated that this system offers substantial benefits to business applications, such as churn prediction, by incorporating feature extraction within the ETL process. Our experiments showed that churners can be identified with an AUC score of over 0.98, while also providing explanations of the classification that assist sales teams in launching personalized retention campaigns. Future work could involve integrating the proposed management system for Distributed Load Applications (DLAs) with YARN, and expanding the algorithms to handle hybrid aggregation scenarios during ETL. Further research could explore approximate database engines and their ability to quickly capture information, providing rapid, preliminary insights that allow users to visualize and identify data segments of interest efficiently. This would enable smooth navigation through data with effective roll-ups and drill-downs, while still allowing for exact queries when necessary. Another promising direction is the integration of machine learning algorithms within the ETL process in a non-blocking manner. This would facilitate online model recalibration to address concept drift, enhancing traditional analytics with real-time predictions.

REFERENCES

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 7.488 || A Monthly Peer Reviewed & Referred Journal |

|| Volume 9, Issue 3, March 2021 ||

| DOI: 10.15680/IJIRCCE.2021.0903222 |

1. Asuncion, H., & Taylor, R. N. (2016). Software engineering for big data systems. IEEE/ACM 38th International Conference on Software Engineering, 19-31.
2. Hu, H., Wen, Y., Chua, T. S., & Li, X. (2014). Toward scalable systems for big data analytics: A technology tutorial. IEEE Access, 2, 652-687.
3. Nargesian, F., Zhu, E., Miller, R. J., Pu, K. Q., & Dong, X. L. (2019). Data Lake management: Challenges and opportunities. Proceedings of the VLDB Endowment, 12(12), 1986-1989.
4. Abadi, D. J., Boncz, P. A., Harizopoulos, S., Idreos, S., & Madden, S. (2013). The design and implementation of modern column-oriented database systems. Foundations and Trends in Databases, 5(3), 197-280.
5. Shvachko, K., Kuang, H., Radia, S., & Chansler, R. (2010). The Hadoop Distributed File System. 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), 1-10.
6. Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., & Stoica, I. (2010). Spark: Cluster computing with working sets. Proceedings of the 2nd USENIX conference on Hot topics in cloud computing, 10-10.
7. Almeida, J. P., Oliveira, L. R., & Guimaraes, D. C. (2018). Big Data ETL on Cloud Environments: A Data-Driven Approach. Journal of Cloud Computing: Advances, Systems and Applications, 7(1), 1-17.
8. Zeng, X., Wu, X., Xiong, W., & Jiang, X. (2019). An effective cloud-based ETL framework for big data integration in cloud environment. Future Generation Computer Systems, 95, 128-139.
9. Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., ... & Zaharia, M. (2016). MLlib: Machine learning in Apache Spark. Journal of Machine Learning Research, 17(1), 1235-1241.
10. Diouf, D., Ndiaye, S., & Niang, I. (2021). Towards the Scalability of ETL Workflows on Cloud Computing Environments. Procedia Computer Science, 184, 625-632.