Machado *et al. Journal of Internet Services and Applications* (2019) 10:21
https://doi.org/10.1186/s13174-019-0121-z

Journal of Internet Services and Applications

RESEARCH Open Access

DOD-ETL: distributed on-demand ETL for near real-time business intelligence

Gustavo V. Machado*, Ítalo Cunha, Adriano C. M. Pereira and Leonardo B. Oliveira

Abstract

The competitive dynamics of the globalized market demand information on the internal and external reality of corporations. Information is a precious asset and is responsible for establishing key advantages to enable companies to maintain their leadership. However, reliable, rich information is no longer the only goal. The time frame to extract information from data determines its usefulness. This work proposes DOD-ETL, a tool that addresses, in an innovative manner, the main bottleneck in Business Intelligence solutions, the Extract Transform Load process (ETL), providing it in near real-time. DOD-ETL achieves this by combining an on-demand data stream pipeline with a distributed, parallel and technology-independent architecture with in-memory caching and efficient data partitioning. We compared DOD-ETL with other Stream Processing frameworks used to perform near real-time ETL and found DOD-ETL executes workloads up to 10 times faster. We have deployed it in a large steelworks as a replacement for its previous ETL solution, enabling near real-time reports previously unavailable.

Keywords: Near real-time ETL, Business intelligence, Big data

Introduction

Today, there is a dire need for organizations to find new ways to succeed in an increasingly competitive market. There is no simple answer on how to achieve this goal. One thing is patently true, though: organizations must make use of near real-time and reliable information to thrive in the global market.

Business Intelligence (BI) is a term used to define a variety of analytical tools that provide easy access to information that supports decision-making processes [1]. These tools perform collection, consolidation, and analysis of information, enabling analytical capabilities at every level inside and outside a company. Putting it another way, BI allows collected data to be unified, structured, and thus presented in an intuitive and concise manner, assisting organizations in corporate decision-making.

The Extract Transform Load (ETL) pipeline is a vital procedure in the Business Intelligence (BI) workflow. It is the process of structuring data for querying or analysis. ETL is made up of three stages, namely: data extraction, data transformation, and data loading where, respectively, data is extracted from their sources, structured accordingly, and finally loaded into the target data warehouse. Two processing strategies can be used in the ETL process: (1) Batch and (2) Stream processing. The difference between them resides in whether the source data is bounded, by known and finite size, or unbounded (arriving gradually over time).

The integration of production systems and BI tools, which is a responsibility of ETL processes, "is the most challenging aspect of BI, requiring about 80 percent of the time and effort and generating more than 50 percent of the unexpected project costs" [2]. For all that, ETL is deemed a mission-critical process in BI and deserves close attention. Getting current, accurate data promptly is essential to the success of BI applications. However, due to the massive amount of data and complex operations, current ETL solutions usually have long run times and therefore are an obstacle to fulfilling BI's requirements.

The main challenges of ETL lie on performance degradation at data sources during data extraction, and on performing complex operations on large data volumes in short time frames. The ideal solution has two conflicting goals: (1) cause no impact on data sources and (2) process data in near real-time as they are generated or updated. Ideal solutions should handle

*Correspondence: gustavovm@ufmg.br
Department of Computer Science, Universidade Federal de Minas Gerais, Belo Horizonte, Brazil

high-volume input data rates and perform complex operations in short time frames while extracting data with no operational overhead. The main characteristics of batch and near real-time ETL are summarized in Fig. 1.

Sabtu et al. [3] enumerate several problems related to near real-time ETL and, along with Ellis [4], they provide some directions and possible solutions to each problem. However, due to the complexity of these problems, ETL solutions do not always address them directly: to avoid affecting the performance of transaction databases, ETL processes are usually run in batches and off-hours (e.g., after midnight or during weekends). By avoiding peak hours, the impact on mission-critical applications is mitigated. However, in a context where the delay in extracting information from data determines its usefulness, BI tools and decision making are heavily impacted when the ETL process is executed infrequently.

In this paper, we proposed DOD-ETL, Distributed On-Demand ETL, a technology independent stack that combines multiple strategies to achieve near real-time ETL. DOD-ETL has minimum impact on the source database during data extraction, delivers a stream of transformed data to the target database at the same speed as data is generated or updated at the source, and provides scalability, being able to respond to data and complexity growth. We achieve all this by synergistically combining multiple strategies and technologies that are usually employed independently (e.g., in [5–8]): log-based Change Data Capture (CDC), stream processing, cluster computing, an in-memory data store, a buffer to guarantee join consistency along with efficient data partitioning and an unified programming model. DOD-ETL works in a distributed fashion and on top of a Stream Processing framework, optimizing its performance.

We have developed a DOD-ETL prototype based on Kafka [9], Beam [10] and Spark Streaming [11]. We evaluate DOD-ETL's performance executing the same workload on a Stream Processing framework with and without DOD-ETL. We have found that our solution, indeed, provides better performance when compared to an unmodified stream processing framework, being able to execute workloads up to 10 times faster while providing horizontal scalability and fault-tolerance.

We have also evaluated DOD-ETL in a large steelworks as a replacement for its previous ETL solution. DOD-ETL has sped up the ETL process from hours to less than a minute. This, in turn, enabled important near real-time reports that were previously impractical.

Our key contributions are: (1) a thorough study and bibliographic review of BI and the ETL process; (2) the design of a near real-time ETL solution; (3) its implementation in a general-use tool called DOD-ETL, using state-of-the-art messaging, cluster computing tools and in-memory databases; (4) and evaluation of DOD-ETL, including a real deployment in the steel sector.

The remainder of this work is organized as follows. First, we discuss the research problem and challenges of near real-time ETL in Section 1. Then, related work is presented in Section 1 and Section 1 presents DOD-ETL and the assumptions under which it has been designed, detailing its implementation and optimization. We evaluate performance, scalability, and fault-tolerance in Section 1. And finally, we summarize our main findings and propose future work in Section 1.

Research problem and challenges

Due to the increasing pressure on businesses to perform decision-making on increasingly short time frames, data warehouses are required to be updated in real-time or in the shortest possible interval. This requirement leads

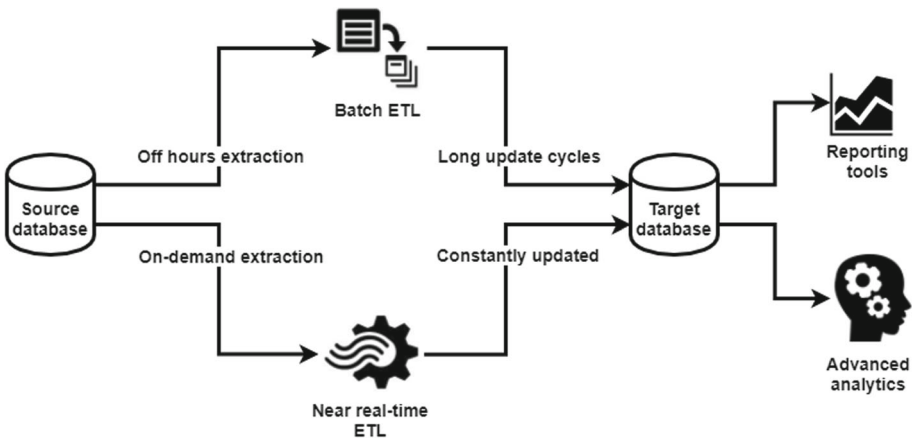

Fig. 1 Batch vs. Near real-time ETL

to the concept of near real-time ETL, a challenging and important topic of research, whose primary definitions, problems and concepts were defined by Vassiliadis and Simitsis [12].

Other emerging concepts related to near real-time ETL are active data warehouses [13], [14] and real-time BI [15–17]. Both describe a new paradigm for business intelligence, in which data is updated in near real-time and both decision-making and resulting actions are performed automatically.

Ellis [4] explains that dealing with near real-time ETL requires three key features to be addressed at each ETL phase: high availability, low latency, and horizontal scalability, which ensure that data will flow and be available constantly, providing up-to-date information to the business.

Both Wibowo [18] and Sabtu et al. [3] identify problems and challenges for developing near real-time ETL systems. Along with Ellis [4], they identify challenges and provide directions to approaching them. In particular, they identify integration with multiple and heterogeneous data sources, performance limitations, data inconsistencies, as well as backup and master data overhead as practical challenges that need addressing.

The main research question that we answer in this work is: *how to enable a near real-time Business Intelligence approach?* We posit that the answer is not trivial, since such a solution imposes multiple requirements:

- High availability;
- Near real-time latency; and
- Horizontal scalability.

Each of these requirements adds complexity, and each is negatively impacted by complexity (imposed by the requirements themselves), leading to a conflict. We present and evaluate a design that strikes positive trade-offs and satisfies these three requirements. Section 1 describes the solutions we combine in our design. In particular, our main insights are (1) a new method to partition different information sources into minimal sets that minimizes data duplication and allows parallel computation, and (2) an in-memory cache to summarize data to support incremental, scalable generation of reports.

Related work

This section is organized into five topics. The first three presents published works that try to solve the problems described in Section 1: data source integration, master data overhead, performance degradation and backup. The fourth focus on publications on Stream Processing frameworks, an important factor for near real-time ETL solutions. As for the last section, we compare the differences between DOD-ETL and related works.

Data source integration

Mesiti et al. [5] take advantage of the Stream Processing paradigm, where processors can work concurrently, to integrate multiple and heterogeneous sensor data sources. This work takes the integration challenge to a more severe domain, where heterogeneity and multiplicity of data sources are accentuated: sensors as data sources. Naeem et al. [6] address near real-time ETL by proposing an event-driven near real-time ETL pipeline based on push technology and with a database queue.

Zhang et al. [7] proposes a new framework to process streaming data in health-care scientific applications. Their framework (1) enables integration between data from multiple data sources and with different arrival rates and (2) estimates workload so it can plan for computational resources to scale appropriately. They extended the Hadoop [19] Map-Reduce framework to address the variable arrival rates of streamed data. To estimate the unknown workload characteristics, they consider two methods to predict streaming data workload: smoothing and Kalman filter.

Waas et al. [20] propose a different approach for ETL with streaming. Their approach first imports raw records from data sources and only executes transformation and data cleaning when requested by reports. In other words, they reorder ETL into ELT and, hence, data is transformed and processed at different times, on demand. To enable this ELT, they developed a monitoring tool to alert all connected components about new incoming data.

The strategy of reordering ETL into ELT can benefit from database replication techniques, which are embedded and orchestrated by the database itself. In this case, the replicas could be used to perform the *transformation* step of the ELT. Data replication is a complex topic with multiple approaches. Wiesmann et al. [21] developed a framework to compare techniques across multiple axes, including synchronous vs asynchronous, distributed vs centralized, or data storage backend. We pay closer attention to asynchronous replication on distributed databases [22, 23]: these solutions are able to provide strong consistency without significantly degrading performance using asynchronous transactional replication.

Master data overhead

Naeem et al. [6] also tried to minimize master data overhead by designing a new approach that manages them during the transformation stage: they divided data into master data, which are more static, and transactional data, which changes more often, and stored that master data on a repository. This strategy made its use more efficient during the transformation step.

Many works focused on the master data overhead problem, where joins between the data stream and master data can lead to performance bottlenecks. To minimize

these bottlenecks, they proposed optimization strategies to these joins [24–27].

Performance degradation and backup

According to Vassiliadis and Simitsis [12], one of the most effective ways of extracting data from a database with minimum overhead is through Change Data Capture (CDC). They also argue that some of the techniques to perform CDC extraction are:

- *Triggers* are created in the source database to notify about each data modification.
- *Timestamps* of transactions in the database allow periodic extraction of new records (i.e., records with timestamps larger than that of the previous extraction).
- *Logs* store changes into the database and can be later processed by systems that depend on the data.

While Jain et al. [8] compared CDC proposals using two metrics to identify data source overload, Shi et al. [28] proposed a CDC framework to evaluate its performance. Both concluded that log-based CDC was the most efficient, having minimal impact on database performance and minimizing data loss. Therefore, performing the extraction stage using logs allows it to provide low latency without degrading performance. It is important to point out that the CDC can be enabled in the replica database, as shown previously. That is, it is possible to combine both to reduce performance degradation and increase backup resilience.

Stream processing frameworks

The above-mentioned publications propose strategies to overcome the challenges related to the three main features of near real-time ETL solutions (high availability, low latency and horizontal scalability). Each work focuses on a specific problem in isolation. However, the Stream Processing paradigm appears as a common denominator among most solutions, which is consistent with near real-time ETL requirements.

In this stream-based application context, multiple stream processing frameworks facilitate the development of such applications or are used directly to solve near real-time ETL requirements. Most of these frameworks are based on the record-at-a-time processing model, in which each record is processed independently. Examples of stream processors frameworks that use this model are Yahoo!'s S4 [29] and Twitter's Storm [30].

In contrast to this record-at-a-time processing model, another possibility is to model it as a series of small deterministic batch computations, as proposed by Zaharia et al. [11] in the Spark Streaming framework. This way, among other benefits, integration with a batch system is made easy and performance overhead due to record-at-a-time operations is avoided. Flink [31] is currently competing with Spark Streaming as the open-source framework for heavyweight data processing. It also merges, in one pipeline, stream and batch processing and it has features such as flexible windowing mechanism and exactly-once semantics.

Besides these open-source frameworks, there are those offered by cloud providers as services such as Google Dataflow [32] and Azure Stream Analytics [33]. By using these services, it is possible to avoid the installation and configuration overhead and the resources allocation to horizontal scalability gets simpler.

These open-source frameworks and stream processing services are both designed for general use. Due to their one-size-fits-all architectures, they lack strategies and optimization that are used by DOD-ETL. Besides, while the above-mentioned papers propose solutions to a specific problem or challenge, DOD-ETL combines solutions to all challenges in a single tool. Finally, we note that the extensions we propose for DOD-ETL are general and can be integrated into any stream processing framework.

DOD-ETL vs. previous works: improvements and trade-offs

We identified that it is imperative for any near real-time ETL solution to have three key features [4]: high availability, low latency, and horizontal scalability. However, we also found that there are some challenging problems to solve to achieve these features [3, 18].

Table 1 sums up some of the main problems concerning near real-time ETL, key works that address them, and the respective solutions used to address the problems.

Table 1 Near real-time ETL problems and solutions

Problem	Work	Solution
Multiple and heterogeneous data sources	Mesiti et al. [5]	Stream processing
	Naeem et al. [6]	
	Zhang et al. [7]	
	Waas et al. [20]	
Performance degradation and backup data	Vassiliadis and Simitsis [12]	Log CDC*
	Jain et al. [8]	Trigger CDC
	Shi, JinGang, et al. [28]	Timestamp CDC
Master data overhead	Naeem et al. [6]	Master data repository
	Polyzotis, Neoklis, et al. [24]	Join optimizations
	Bornea, Mihaela A., et al. [26]	
	Naeem, M. Asif, et al. [27]	
	Zhang et al. [7]	

Precisely, it highlights the main problems for ETL: (1) multiple and heterogeneous data sources, (2) performance degradation and backup data, and (3) master data overhead problems; and corresponding existing approaches: (1) stream processing, (2) CDC, (3) and master data repository or join optimizations.

DOD-ETL combines multiple strategies, which were previously used separately, to achieve near real-time ETL. Notably, DOD-ETL employs most of the strategies mentioned in Table 1: it extracts changes from the CDC using stream processing and also employs an in-memory database to cache master data. Besides, DOD-ETL takes advantage of cluster computing frameworks features and data replication on stream processing to guarantee fault-tolerance and prevent data loss.

That said, DOD-ETL employs strategies different from previous works to improve its performance even further. For instance, unlike Naeem et al. [6], which uses a central repository, DOD-ETL uses in-memory databases, deployed on each processing cluster node, to provide master data. This strategy enables master data to be accessed by the processing nodes promptly. DOD-ETL also supports integration between data with varying arrival rates as Zhang et al. [7] but with a different strategy. While Zhang et al. [7] uses adaptive algorithms to provision resources and cache out of sync data, DOD-ETL uses a buffer with all late data and their timestamps. Regarding on-demand transformation capabilities, while Waas et al. [20] loads data first and only triggers transformation when requested, DOD-ETL takes advantage of CDC to process data at the same pace and only when changes occur. This enables data to be provided for consumption faster when compared to ETL previous solutions that perform data transformation when processing a request.

Each of the points above will be further described in detail in Section 1.

DOD-ETL

DOD-ETL relies on an on-demand data stream pipeline with a distributed, parallel and technology-independent architecture. It uses Stream Processing along with an in-memory master data cache to increase performance, a buffer to guarantee consistency of join operations on data with different arrival rates, and a unified programming model to allow it to be used on top of a number of Stream Processing frameworks. Besides, our approach takes advantage of (1) data partitioning to optimize parallelism, (2) data filtering to optimize data storage on DOD-ETL's master data cache and (3) log-based CDC to process data on-demand and with minimum impact on the source database. Therefore, DOD-ETL has: minimum impact on the source database during extraction, works in near real-time, can scale to respond to data and throughput growth and can work with multiple Stream Processing frameworks.

The insights that enable DOD-ETL to achieve the features mentioned above are: *On-demand data stream*—as data changes on the source database, they are processed and loaded into the target database, creating a stream of data where the ETL process handles, in a feasible time frame, only the minimum amount of necessary data. *Distributed & parallel*—perform all steps in a distributed and parallel manner, shortening processing time and enabling a proper use of computing resources. *In-memory Cache*—perform data transformations with no look-backs on the source database, providing all required data to execute calculations in the In-memory Cache which avoids expensive communications with the data source. *Unsynchronized consistency*—a buffer guarantees that data with different arrival rates can be joined during transformation. *Unified programming model*—a programming model used to build data pipelines for both batch and streaming in multiple Stream Processing frameworks.

DOD-ETL merges into a single solution multiple strategies and techniques that have never been integrated to achieve near real-time ETL: log-based Change Data Capture (CDC), stream processing, cluster computing, and an in-memory data store along with efficient data partitioning (c.f. Section 1). Although these techniques have indeed been used before (e.g., in [5–8])), they have not been previously integrated into a single solution. By synergistically combining these strategies and techniques DOD-ETL can achieve all the three key features needed to perform near real-time ETL: high availability, low latency and horizontal scalability. The next sections present a detailed explanation of DOD-ETL's architecture and how each module contributes to solving near real-time ETL challenges.

Architecture

DOD-ETL has the following workflow (Fig. 2): (1) it tracks changes on each source system's database table and (2) sends these changes as messages to a (3) message queue, following a pre-configured partitioning criteria. Then, (4) these messages are pulled from the message queue and sent to the In-memory Cache or (5) transformed to the target reporting technology format, and, finally, (6) loaded into the target database. DOD-ETL's workflow can be grouped into two central modules: Change Tracker and Stream Processor.

All steps depend on configuration parameters to work properly. Thus, during DOD-ETL's deployment, it is imperative to go through a configuration process, where decisions are made to set the following parameters: *tables to extract*—define which tables will have data extracted from; *table nature*—from the defined tables, detail which ones are operational (constantly updated) and which ones are master data (semi-static); *primary key*—for each

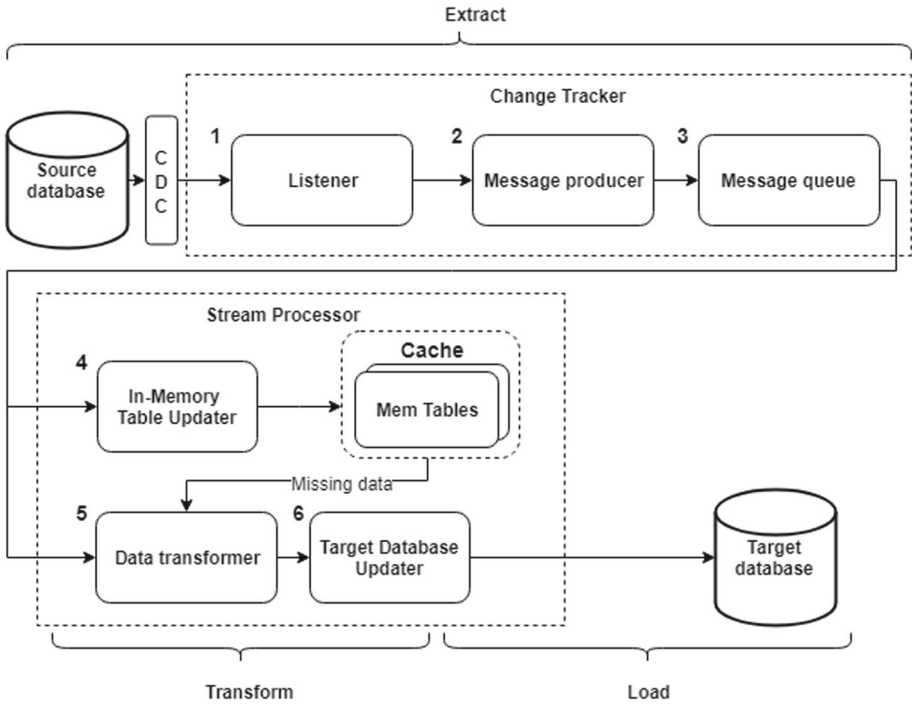

Fig. 2 DOD-ETL workflow step by step

table, the set of columns that contains each row's unique identifier; *business key*—for each table, the set of columns that define how to partition or filter data by a domain-specific key.

Change tracker

The Change Tracker makes available to the Stream Processor module, instantaneously, as events occur, any data altered and added to the source database. This module makes use of CDC, a standard database pattern that contains all operations carried out over time and their respective values. Thus, whenever a record is inserted, altered, or removed from a table, the CDC writes that event, together with all of the record's information.

CDC can take many forms, ranging from log files to structured tables in the database. They can be generated either by the database itself or by an external system. In this work, in agreement with Jain et al.'s recommendations [8], log-based CDC was used. However, as explained later, the CDC reading step in DOD-ETL is performed by the Listener; thanks to DOD-ETL's modular architecture, new CDC implementations can be supported by creating different Listeners.

The Change Tracker works in three stages called Listener, Message Producer, and Message Queue. The Listener stage listens to the CDC changes and, with each new record, extracts its data to be further processed by Message Producer. This stage was built to extract data independently from tables, allowing it to be parallelized. As a prerequisite, then, the Listener expects the CDC log to support this feature. The Listener step has minimum impact on the source database's performance due to two factors: (1) only new and altered records are extracted, minimizing data volume, and (2) queries are performed in log files only, which takes the pressure off the database and production tables. As the Listener step has been designed to be decoupled from the other two steps, it can be replaced or extended to meet the specific characteristics of the source database, such as different technologies and CDC implementations.

The Message Queue works as a message broker and uses the publish/subscribe pattern with partitioning features, in which each topic is partitioned by its key. On DOD-ETL, a topic contains all insertions, updates and deletions of a table. The topic partitioning criteria vary by the table nature that the topic represents (master or operational). When dealing with master data tables, each topic is partitioned by its respective primary key and, when dealing with operational data, each topic is partitioned by the business key. This partitioning occurs at the Message Producer, before it publishes each extracted datum, based on the aforementioned configuration parameters. Therefore, Message Producer builds messages from data extracted by the Listener and publishes them in topics on the Message Queue according to the pre-configured parameters. These two partitioning strategies (by primary key and by business key) have the following objectives:

- *Primary key*: Since the primary key is the topic partition key, the Message Queue guarantees that the consumption of messages from a specific table row will happen in the right sequence. Morever, as new messages for a primary key overwrite the previous message, it suffices to retrieve the last message for each partitioning key (primary key) to reconstruct a snapshot of the database.
- *Business key*: The Stream Processor transformation process is parallelized based on the number of partitions defined by operational topics and their business keys. Therefore, the partitioning criteria has a direct impact on DOD-ETL's performance. In this sense, it is paramount to understand in advance the extracted data and the nature of operations that will be performed by the Stream Processor. The goal is to figure out the partitioning criterion and its key, because they may vary depending on the business nature.

Analyzing the main near real-time features and challenges for data extraction, we conclude that the use of log-based CDC enables low latency without performance degradation and high availability due to processing only data changes, and the ability to restore snapshots when required. Since Change Tracker extracts data from each configured table independently, it guarantees data extraction scalability with the number of available partitioning keys. The partitioning criteria used by the Message Queue (business key and primary key) guarantees the horizontal scalability of data consumption. In this case, it enables the Stream Processor to scale.

Figure 3 provides an overview of Change Tracker's APIs. We note that both Listener and Message Producer were built as interfaces to facilitate extension and substitution with different functionality, for example, to adjust to different databases and Message Queue restrictions. The figure also shows that all modules are parameterized, e.g., to allow configuration of how many parallel data extraction tasks are executed, or what are the partitioning keys of produced messages.

Stream processor

The Stream Processor receives data from the Listener by subscribing to all topics published on Message Queue, receiving all recently changed data as message streams. The Stream Processor comprises three steps: (1) In-memory Table Updater, (2) Data Transformer and (3) Target Database Updater.

The *In-memory Table Updater* prevents costly lookbacks on the source database by creating and continuously updating distributed in-memory tables that contains supplementary data required to perform a data transformation. Data flowing from topics representing master data tables go into the In-memory Table Updater step. The In-memory Table Updater only saves data related to the business keys assigned to its corresponding Stream Processor node, filtering messages by this key. By doing so, only data from keys that will be processed by the node are saved in its in-memory table, reducing memory pressure. In case of node failures, data loss, or both, the Stream Processor retrieves a snapshot from the Message Queue and repopulates in-memory tables that went down. This is possible due to the way each Message Queue's master data topic is modeled: it is partitioned by the table's

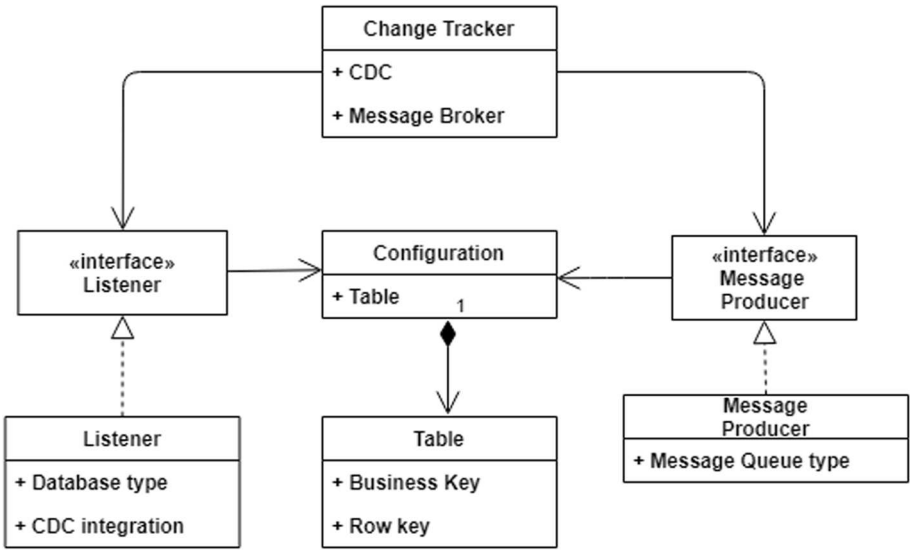

Fig. 3 Change Tracker API UML diagram

primary key, allowing the In-memory Table Updater to retrieve an exact snapshot of this topic table from the Message Queue.

The *Data Transformer* receives data and performs operations to transform it into the required BI report format. The Data Transformer can run point-in-time queries on the in-memory tables to fetch missing data necessary to carry out its operations, joining streaming and static data efficiently. Each partition is processed in parallel, improving performance. The operations executed in Data Transformer rely on the operators of the cluster computing framework (e.g., map, reduce, filter, group) and are business-dependent. Like the In-memory Table Updater, not all messages go through this module, only messages from tables configured as operational. For the events of master data arriving after operational data (master and operational messages are sent to different topics and by different Listener instances), the Data Transformer uses an Operational Message Buffer to store the (early) operational messages for reprocessing after receipt of the (late) master data. At the arrival of new master data, Data Transformer checks the buffer for pending operational messages and reprocesses them, along with the new master data. To optimize performance, Data Transformer only reprocesses buffer messages with transaction dates older than the latest transaction date from the In-memory Cache, which avoids reprocessing operational messages that are still missing master data. As shown in more detail in Section 1, DOD-ETL's performance is highly dependant on the data model and the complexity of transformation operations. That is, the data and how it is transformed, in this case by Stream Processor, is a key factor in DOD-ETL's processing rate.

The *Target Database Updater* translates the Data Transformer's results into query statements and then loads the statements into the target database. For performance, the loading process also takes place in parallel and each partition executes its query statements independently.

Due to stream processing and publish/subscribe approaches employed by the Stream Processor and the Message Queue, data from multiple and heterogeneous sources can be processed simultaneously. In addition, the use of stream processing frameworks along with an efficient partitioning strategy enables high availability and horizontal scalability. The use of an In-memory Cache and an Operational Message Buffer enables DOD-ETL to process data with low latency, since master data is available promptly. These solutions combine to provide all three key features presented in Section 1 in DOD-ETL.

As shown in the Stream Processor API UML (Fig. 4), the Stream Processor has an interface called Message Consumer that implements the particularities of the selected Message Queue. The Target Database Updater module also has an interface to allow similar flexibility. These two modules are able to adapt to the deployment environment or software stack. Since Data Transformer transformations vary from business to business, it also has an interface, called Business Domain, to incorporate the business model and transformations.

Scalability, fault-tolerance and consistency

DOD-ETL takes advantage of the adopted cluster computing framework's and the message queue's native support for scalability and fault-tolerance. For fault-tolerance, both DOD-ETL modules can handle node failures, but with different strategies: while the Change Tracker focuses

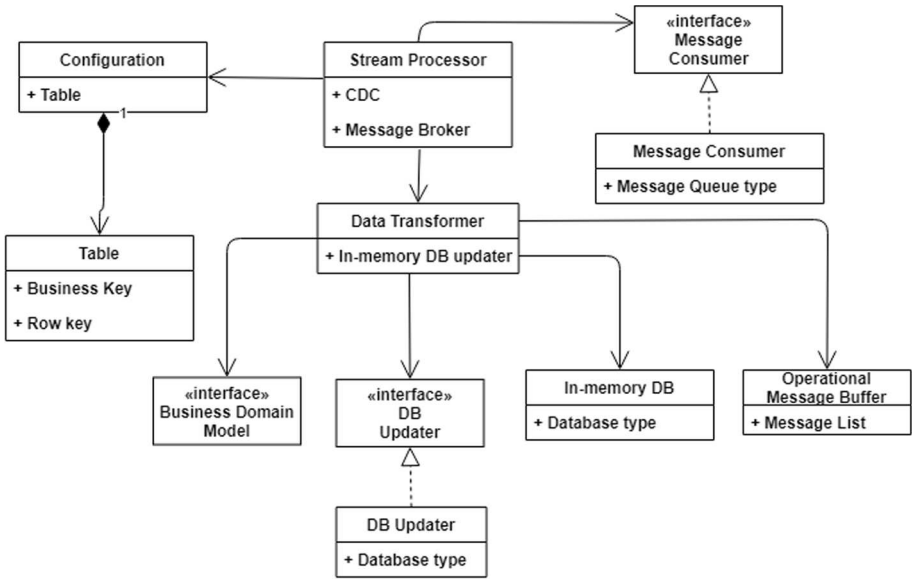

Fig. 4 Stream Processor API UML diagram

on no message loss, the Stream Processor concentrates on no processing task loss. As for scalability, both modules depend on efficient data partitioning to scale properly.

Despite these inherited capabilities, some features had to be implemented on the Stream Processor so fault-tolerance and scalability could be adherent to DOD-ETL's architecture: the In-memory Cache and the Operational Message Buffer have to be able to retrieve data previously stored on failed nodes or data from new nodes. Regarding the In-memory Cache, we implemented a trigger that alerts the In-memory Table Updater when the Data Transformer's assigned business keys changes. On a change event, the In-memory Table Updater resets the In-memory Cache, dumps the latest snapshot from the Message Queue and filters it by all assigned business keys. By doing so, the In-memory Cache keeps up with the Stream Processor reassignment process on failure events or when the cluster scales up or down. As for the Operational Message Buffer, it uses a distributed configuration service, built into the Message Queue, to save its configuration: when a note fails, other Stream Processor instances can pick up the processing tasks. That is, the Operational Message Buffer saves all operational messages with late master data and, at each new message, Data Transformer tries to reprocess this buffer by checking at the In-memory Cache if its respective master data arrived, solving the out-of-sync message arrival problem.

Since DOD-ETL focuses on delivering data to information systems, not operational mission-critical systems, transactional atomicity was not a requirement and it was left out of the scope of this work: rows of different tables added/altered in the same transaction can arrive at slightly different time frames. However, due to its late operational messages buffer, DOD-ETL can guarantee that only data with referential integrity will be processed and that those that are momentarily inconsistent will eventually be processed. As stated before, the Data Transformer reprocesses all messages from the Operational Message Buffer when all needed data arrives on the In-memory Cache.

Implementation

DOD-ETL is a tool with many parts: CDC, Listener, Message Queue, Stream Processor. While the Listener was built from scratch, the CDC and the Message Queue are simply out of the shelf solutions, and the Stream Processor is a stream processing framework with customizations and optimizations built on top of a unified programming model. Its implementation and technologies are explained next.

All data exchanged between DOD-ETL modules are serialized and deserialized by the Avro system [34]. The Message Queue role is performed by Kafka [9], an asynchronous real-time message management system, whose architecture is distributed and fault-tolerant. Due to Kafka's dependency on Zookeeper [35], it is also used by DOD-ETL.

The Stream Processor was implemented with Beam [10], a unified programming model that can be used on top of stream processing frameworks such as Spark Streaming and Flink. Its steps, Data Transformer, In-memory Table Updater and Target Database Updater, were all encapsulated together to make communication between them easy. The Data Transformer takes advantage of the Zookeeper dependency to store its late operational messages buffer. It does so to guarantee that, in any failure event, another Stream Processor node could keep processing those late messages.

Regarding the In-memory Cache, H2 [36] was used and deployed as an embedded database on the Stream Processor. To support DOD-ETL's needs, H2 was configured to work in-memory and embedded so, for each Spark worker, we could have an H2 instance with fast communication. Our prototype and its modules are publicly available[1].

Since DOD-ETL modules were designed to be decoupled, each one can be extended or replaced without impacting other modules. Adding to this its technological-independent features, all selected technologies on each module can be replaced, provided that its requirements are satisfied.

Evaluation

We used, as a case study, a large steelworks and its BI processes. In this case, a relational database, powered by both its production management system and shop floor level devices, was used as the data source. This steelworks has the following characteristics: total area of 4,529,027 m^2, a constructed area of 221,686 m^2, 2238 employees, and it is specialized in manufacturing Special Bar Quality (SBQ) steel.

This steelworks uses OLAP reports [37] as its BI tool. DOD-ETL was used to provide near real-time updates to these reports. As a comparison point, reports were updated twice a day prior to DOD-ETL's deployment. DOD-ETL's purpose was to extract data from the source database and transform them into OLAP's expected model, known as *star schema* [38]. This transformation involves calculations of Key Process Indicators (KPIs) of this steelworks' process. For this case study, the Overall Equipment Effectiveness (OEE) [39], developed to support Total Productive Maintenance initiatives (TPM) [40], and its related indicators (Availability, Performance, and Quality) were chosen as the steelworks KPIs.

TPM is a strategy used for equipment maintenance that seeks optimum production by pursuing the following objectives: no equipment breakdowns; no equipment running slower than planned; no production loss.

[1] https://github.com/gustavo-vm/dod-etl

OEE relates to TPM initiatives by providing an accurate metric to track progress towards optimum production. That is, the following KPIs can quantify the above three objectives: *availability*—measures productivity losses, tracking the equipment downtime vs. its planned productive time; *performance*—tracks the equipment actual production speed vs. its maximum theoretical speed; *quality*—measures losses from manufacturing defects. These three indicators together result in the OEE score: a number that provides a comprehensive dimension of manufacturing effectiveness.

DOD-ETL extracted only the data needed to calculate these indicators. For this industry context, we grouped them into the following categories: *production data*—information of production parts; *equipment data*—equipment status information; *quality data*—produced parts quality information. During the DOD-ETL configuration process, two decisions were made: we defined the nature of each table (operational and/or master data) and decided which table column would be considered as the Stream Processor business partitioning key. Regarding the table nature, we considered the production group as operational and equipment and quality as master data. Due to this decision, all equipment and quality data sent to Stream Processor will be stored on its In-memory Cache while production data will go straight to the Data Transformer step of the Stream Processor.

As for the business key, we considered the production equipment unit identifier, since all KPIs are calculated for it. We set, then, the column that represents and stores the related equipment unit code on each table as the business key in the configuration. This column will be used by operational topics for partitioning and by the In-memory Cache as filter criteria.

For this industry context, Data Transformer splits data as a requisite to support OLAP multidimensional reports. For the above mentioned KPIs, we defined the splitting criteria as its intersections in the time domain, in which the lowest granularity represents the intersection between all the data obtained for the equipment unit in question: Production, Equipment status and Quality. Figure 5 shows an example of this intersection analysis and data splitting. In this case, Data Transformer searches for intersections between equipment status and production data and breaks them down, generating smaller data groups called fact grain. In the example in Fig. 5, these fact grains can be divided into two groups: (1) equipment with status "off" and (2) equipment with status "on" and production. As stated before, after the splitting process is completed, the Data Transformer performs the calculations.

Experiments

To evaluate our DOD-ETL prototype's performance, we used the Spark Streaming framework [11] as the baseline. We generated a synthetic workload, simulating the data sent by the steelworks equipment, and executed Spark with and without DOD-ETL on top of it. We have also performed experiments to check if DOD-ETL achieved Ellis' key features [4] (high availability, low latency and horizontal scalability) and, as a differential of our work, we executed DOD-ETL with production workloads from the steelworks to check its behavior in a complex and realistic scenario.

In sum, we evaluated (1) DOD-ETL vs Baseline, checking how Spark performs with and without DOD-ETL on top of it; (2) horizontal scalability, analyzing processing time when computational resources are increased; (3) fault tolerance, evaluating DOD-ETL's behavior in the event of failure of a compute node; (4) DOD-ETL in production, comparing its performance against real workloads and complex database models from the steelworks.

We used Google Cloud and resources were scaled up as needed, except for the fifth experiment that used the steelworks's computing infrastructure. All Google Cloud Platform instances had Intel Haswell as their CPU platform and hard disk drives for persistent storage. To represent the three data categories cited before, we used one table per data model group on experiments 1, 2 and 3. Regarding the fourth experiment, we used a more complex data model based on the ISA-95 standard [41]. The following hardware and software and configurations were used (Table 2):

Also for experiments 1, 2 and 3, as mentioned above, we built a sampler to insert records on each database table. This sampler generates synthetic data, inserting

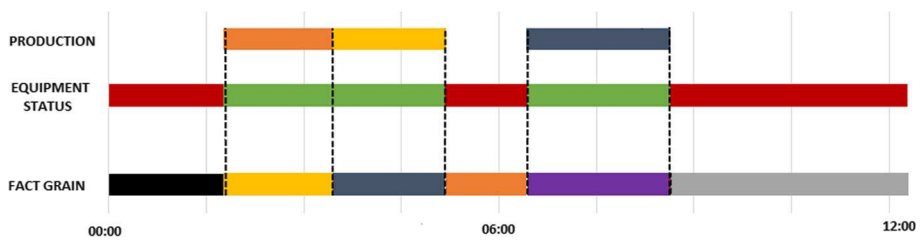

Fig. 5 Data splitting working on metals industry context

Table 2 Experiments hardware and software configuration

Module	Software	Hardware	Instances
Database	MySQL database	8-core 10 GB	1
Sampler	Python script	20-core 18 GB memory	1
Change Tracker	Python script	20-core 18 GB memory	1
Message Queue	Kafka	one core and 2 GB memory	3
	Zookeeper	one core and 2 GB memory	3
Stream Processor	Spark Streaming DOD-ETL job	one core and 2 GB memory	From 1 to 20

20,000 records on each table, simulating the steelworks operation. To minimize the impact of external variables in these experiments, the Listener started its data extraction after the sampler finished its execution. To avoid impact on the results, Change Tracker extracted all data before the execution of Stream Processor, so the Message Queue could send data at the same pace as requested by the Stream Processor. Since the Listener depends on the CDC log implementation, its data extraction performance also depends on it. We used MySQL as the database and its binary log as the CDC log.

DOD-ETL vs. baseline

Since DOD-ETL is comprised of out-of-the-shelf components, modules combine in a decoupled architecture. To evaluate DOD-ETL's performance, each module needs to be analyzed separately.

As said before, the Listener is highly dependant on the used database and CDC implementation and its performance is tied to them. Therefore, the Listener has the CDC throughput as its baseline. Since the complete flow from writing to the database to copying it to the binary incurs a lot of I/O, the Listener will always be able to read more data than CDC can provide.

Since the Message Queue is an out-of-the-shelf solution and can be replaced by any other solution, provided that its requirements are satisfied, its performance can benefit from improvements in existing solutions or development of new ones. As for now, Message Producer and Message Queue are instances of Kafka producers and Kafka brokers, respectively. Kreps et al. [9] already demonstrated its performance against other messaging systems.

The Stream Processor includes substantial customizations (Data Transformer, In-memory Table Updater and Target Database Updater) on top of the Spark Streaming framework. We evaluated its performance executing the same workload against Spark Streaming with and without DOD-ETL. We used a fixed number of Spark worker nodes (Table 3): a cluster with ten nodes.

Table 3 Experiments results (records/s)

Baseline		Fault Tolerance		Production	
DOD-ETL	Spark	Normal	Failure	Simple Wkld	Real Wkld
10,090	1230	5063	2216	10,090	230

As shown in Table 3, DOD-ETL was able to process 10,090 records per second. In contrast to the 1230 records processed by Spark Streaming alone, which represents ten times fewer records. Looking at the Spark job execution log, that shows the processing time in milliseconds of each Spark Worker task (Fig. 6), it is possible to identify an initialization overhead. This overhead is due to the In-memory Cache startup: when a new business key is assigned it dumps from the Message Queue all the respective master data. In the case of this experiment, the initialization overhead costs 40 s for each Spark worker.

Throughout these experiments, we were able to demonstrate that DOD-ETL customizations make Spark significantly faster. Although it has an initialization overhead, due to the In-memory Table Updater data dump from Message Queue, it is minimal and negligible considering the volume of processed messages. In other words, DOD-ETL is able to process data at a higher rate than the baseline, providing low latency, one of the three features of near real-time ETL, better than the baseline alone.

The next experiments will show that DOD-ETL customizations do not negatively impact Spark Streaming's fault-tolerance and scalability. This is due to the synergy between the use of the efficient data partitioning strategy, Operational Message Buffer, Message Queue and In-memory Cache.

Scalability

We evaluated the scalability of DOD-ETL's Change Tracker and Stream Processor modules. We performed two different experiments to evaluate the Listener's performance: (1) *Same number of input and output tables*, where the number of input tables (where data is extracted from) and output tables (where data is inserted into) varied from 1 to 16; and (2) *Fixed number of output tables*, where data were inserted into 16 tables and number of tables data was extracted from was increased from 1 to 16.

As shown in Fig. 7, where the number of records inserted per second was plotted against the number of tables, the Listener achieved different performance on each experiment: When the number of input and output tables is the same, the Listener's performance increases as a sub-linear function and then saturates at 18,600 records per second for eight tables. When using a fixed number of output tables, performance increased linearly until it also saturated when extracting simultaneously from eight tables, with a throughput of 10,200 records per second.

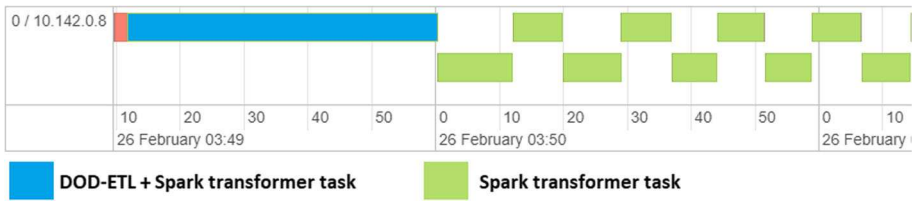

Fig. 6 DOD-ETL initialization overhead

This behavior is directly related to MySQL's CDC implementation, as it writes changes from all tables on the same log file, so each Listener instance has to read the whole file to extract data from its respective table. Throughput is higher in the first experiment compared to the second experiment because of the difference in log file size: while in the experiment with a fixed number of output tables the CDC log file had a fixed size of 320,000 records, in the varying experiment with variable number of output tables the CDC log file varied from 20,000 records to 320,000 records. Therefore, going through the whole log file took less time until it matched at 16 tables. The saturation point is a function of MySQL performance and we conjecture it will vary across different databases and CDC implementations.

As already stated, the Message Producer and the Message Queue are instances of Kafka producers and Kafka brokers, respectively. Kreps et al. [9] already demonstrated that each producer is able to produce orders of magnitude more records per second than the Listener can process. Regarding its brokers, their throughput is dictated more by hardware and network restrictions than by the software itself, also enabling it to process more records.

To evaluate the Stream Processor's scalability, we varied the number of available Spark worker nodes from one to twenty and fixed the number of partitions on the operational topic at twenty. To maximize parallelization, the number of equipment units (partition keys) from the sampled data followed the number of topic partitions: sampled data contained 20 equipment unit identifiers, used as partition keys. We used the Spark Streaming Query Progress' metric *average processed rows per second* at each of its mini-batch tasks. As shown in Fig. 8, where the number of processing records per seconds was plotted against the number of Spark Workers.

DOD-ETL modules are scalable: both Change Tracker and Stream Processor can absorb growth by adding more computational resources, despite their difference in throughput and scalability factor. While the Change Tracker scales proportionally to the number of tables, Stream Processor scales with the number of partition keys on operational tables.

Regarding Change Tracker, we saw that the Listener and the Message Producer can process tables independently and that it can scale up as the process incorporates new ones, provided that the database CDC log supports it. As for the Message Queue, it also scales linearly but based on multiples variables: the number of available brokers, extracted tables (topics), partitions, and keys.

Stream Processor's scalability is proportional to the number of partitions at the operational table topics and the number of partitioning keys that, on this steelworks case, are the total number of production equipment units. Data partitioning plays a vital role here, so, it is imperative to understand functionally and in advance all extracted data in order to find partitioning criterion and its key, which varies from business to business. Since the Message Queue supports throughput orders of magnitude higher than the Listener and the Stream Processor, it is possible to integrate multiple database sources and use multiple Stream Processor instances, each performing different transformations.

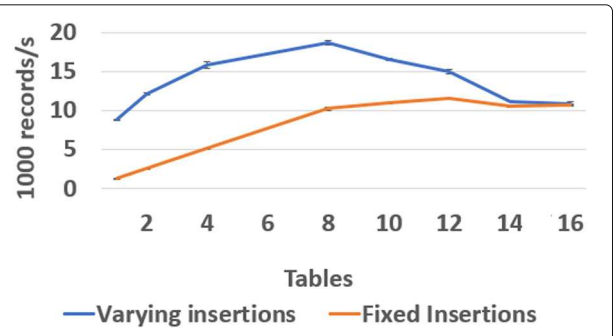

Fig. 7 Scalability: Listener experiment result

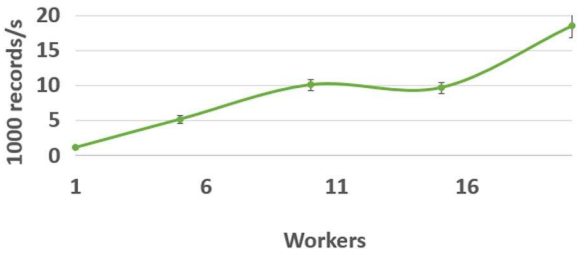

Fig. 8 Scalability: Stream Processor experiment result

Fault tolerance

We have executed DOD-ETL in a cluster with five worker nodes and, midway through the experiment, we shut down two of the worker nodes to evaluate DOD-ETL's fault tolerance. We measured the rate of processed messages before and after the shutdown and performed a data consistency check on the result. We used the same performance metric as the scalability experiment (Table 3).

The Stream Processor went from processing 5060 messages per second to 2210, representing a processing rate decrease of 57%. After each execution, we checked the consistency of all messages processed by this module and did not find any error: it processed all messages correctly, albeit at a reduced rate. This result indicates that DOD-ETL is fault-tolerant, which significantly increases its robustness.

While the number of available clusters was changed from 5 to 3, a 40% decrease, the performance decrease was more significant (57%). By analyzing the Spark execution log, we found that the In-memory Cache also impacts failover performance: when a node becomes unavailable and a new one is assigned, the In-memory Cache has to dump all data from the newly assigned partition keys, which impacts performance.

Since the Change Tracker and the Stream Processor were built on top of Kafka and Spark, respectively, both modules can resist node failures. Due to their different purposes, each module uses distinct strategies: while Kafka can be configured to minimize message loss, Spark can be configured to minimize interruption of processing tasks.

DOD-ETL in production

We executed DOD-ETL with real workloads from the steelworks. This workload is generated by both a production management system and shop-floor devices and its data model is based on the ISA-95 standard, where multiple tables are used to represent each category (production, equipment and quality). We compared DOD-ETL results on previous experiments, where synthetic data was used with a simpler data model (a single table for each category of data), with DOD-ETL executing on real and complex data (Table 3).

Both synthetic and production experiments used the same configuration: a cluster with ten Spark worker nodes. While DOD-ETL can process 10,090 records per second for the data source with simple model complexity, this number decreases to 230 records per second for the complex data model. It is possible to state, then, that data model complexity impacts directly on DOD-ETL performance. Since it depends on the In-memory Cache to retrieve missing data, when this retrieval involves complex queries, this complexity impacts on the query execution time and, therefore, on DOD-ETL performance.

This steelworks currently uses an ETL solution to perform the same operation performed by DOD-ETL. It adopts a sequential batch approach, comprises a series of procedures ran within the relational database server, and relies on a twelve core /32 GB memory server. While DOD-ETL takes 0.4 s to process 100 records, this solution takes one hour. Although it is not a fair comparison (a streaming distributed and parallel tool vs. a batch and legacy solution), it is important to demonstrate that DOD-ETL can be successfully used in critical conditions and can absorb the steelworks data throughput, providing information in near real-time to its BI tools.

Considering the author's experience in developing mission-critical manufacturing systems and its knowledge in the ISA-95 standard, his opinion regarding these systems data modeling is that the drawbacks of using a standardized and general data model, that seeks a single vision for all types of manufacturing processes, far outweigh the benefits. Manufacturing systems that use generalized data models get way more complex when compared with process-specific models. These systems' performance, maintenance and architecture are severely impacted in order to comply by a generic model.

Therefore, in this industry context, a more straightforward data model could be used in the production management systems and shop-floor without drawbacks. With this, DOD-ETL (and possibly other factory systems) would perform even better.

Conclusion and future work

DOD-ETL's novelty relies on synergistically combining multiple strategies and optimizations (that were previously only used separately) with an on-demand data stream pipeline as well as with a distributed, parallel, and technology-independent architecture.

In this work we address the main research question, which is: *how to enable a near real-time Business Intelligence approach?* Near real-time ETL systems need to have three key features: high availability, low latency, and scalability. DOD-ETL has been able to achieve all three key features and address these challenges by combining log-based Change Data Capture (CDC), stream processing, cluster computing, an in-memory data store, a buffer to guarantee join consistency along with efficient data partitioning, and a unified programming model.

We have been able to demonstrate, by performing independent experiments on each of its main modules, that DOD-ETL strategies and optimizations reduce ETL run time, outperforming a modern Stream Processing framework. Through these experiments, we showed that DOD-ETL achieves these results without sacrificing scalability and fault-tolerance. We have also found that data source model complexity heavily impacts the transformation

stage and that DOD-ETL can be successfully used even for complex models.

Due to its technology-independence, DOD-ETL can use a number of Stream Processor frameworks and messaging systems, provided that requirements are satisfied. This allows DOD-ETL to adapt and evolve as new technologies surface and avoids technology lock-ins.

Instantiating DOD-ETL requires customizing the Data Transformer step: each required transformation is translated as Spark operators which, in turn, are compiled as a Java application. This requires DOD-ETL's users to know how to program, restricting its use somewhat. To overcome this, on future work, DOD-ETL will be adapted to integrate a friendly user interface with an intuitive and visual configuration of Data Transformer transformation operations.

Also on future work, we will study the impact on DOD-ETL performance when lightweight Stream Processing frameworks are used, such as Kafka Streams and Samza, by performing new experiments. By doing so, we will be able to compare and evaluate the trade-offs between these two types of frameworks (lightweight vs. heavyweight) and its impact on DOD-ETL's strategies and optimizations.

In sum, through this work, we were able to achieve near real-time ETL by combining multiple strategies and technologies, to propose a general-use tool and to evaluate it in the metals industry context.

Abbreviations

CDC: Change Data Capture; DOD: Distibuted On-Demand; ETL: Extract Transform Load; ISA: International Society of Automation; KPIs: Key Process Indicators; OEE: Overall Equipment Effectiveness; OLAP: Online Analytical Processing; TPM: Total Productive Maintenance initiatives

Acknowledgements

The authors want to thank Accenture, FAPEMIG, CNPq, and CAPES for partially supporting this paper.

Authors' contributions

GV played the most important role in this paper. Other authors have contributed equally. All authors read and approved the final manuscript.

Authors' information

Gustavo V. Machado received his bachelor degree in Control and Automation Engineering in 2014 at UFMG and his MSc. in 2018 in Computer Science. He worked as a Consultant at Accenture for industrial systems in mining and siderurgy companies and now works as a Data Engineer in financial markets at Itaú. He is interested in Data Science, Business Intelligence, IoT and Manufacturing Executing Systems.
Leonardo B. Oliveira is an Associate Professor at UFMG, Researcher at Brazil's National Scientific Council and, currently, a Visiting Associate Professor at Stanford. Leonardo has managed projects funded by companies like Google, Intel Labs, and Microsoft. He is interested in IoT/Cyber-Physical Systems and Data Science.
Adriano C. Machado Pereira is an Associate Professor in Computer Science Department at Federal University of Minas Gerais (DCC / UFMG), Brazil. He received his bachelor degree in Computer Science at UFMG in 2000, his MSc. in 2002, and his Ph.D. in 2007. During his Ph.D. he had worked with Professor Paulo Góes as a visitor researcher at the Business School of University of Connecticut. He also had performed a Post-Doc research in electronic markets in 2008-2009, working with data characterization, personalization and anomaly detection, with a sponsorship of UOL Corporation. His research interests include e-Business, e-Commerce, Workload Characterization, Distributed Systems, Web 2.0, Social Networks, Performance of Computer Systems, Web Technologies, Business Intelligence, Data Science, Financial Markets and Algorithmic Trading. Before Academy, he had worked for eight years for a start-up company, in areas such as e-commerce, dynamic pricing, retail revenue management, and performance analysis of computer systems. His experiences with industry and applied research include projects with Universo OnLine S/A (UOL Corp. – since 2008), TeamQuest (2001-2005), CGI.br (Brazilian Internet Management Committee – since 2009), Smarttbot (www.smarttbot.com – since 2013), W3C (www.w3c.br - since 2009), United Nations (ONU/PNUD – 2007-2010), and some Brazilian Government projects (Ministry of Science and Technology). He is also a member of the Brazilian National Institute of Science and Technology for the Web - InWeb (www.inweb.org.br).
Ítalo Cunha is an assistant professor at the Computer Science Department at UFMG, Brazil, since 2012. He developed his Ph.D. under the French CIFRE Program for cooperation between industry and academia. He developed his Ph.D. research at Technicolor Research and Innovation, and graduated from UPMC Sorbonne in 2011. His research focuses on improving network performance, reliability and security. His contributions provide better visibility on Internet topology and routing dynamics; help network operators troubleshoot failures and performance problems; and empower other researchers. Ítalo has implemented novel network measurement tools like DTrack, built distributed global-scale Internet monitoring systems like LIFEGUARD and Sibyl, and deployed state-of-the-art research and experimentation platforms like PEERING. He holds a research fellowship from Brazil's National Science and Technology Foundation (CNPq), and his contributions have been published in conferences like ACM SIGCOMM and USENIX NSDI. His research is funded by federal grants and private grants from technology companies. Ítalo has served on the technical committee of flagship networking conferences such as ACM IMC and ACM SIGCOMM; he serves as a member of the National Brazilian Research and Education Network (RNP) Monitoring Work Group.

Funding

This work had no funding.

Availability of data and materials

Our prototype and its modules are publicly available at https://github.com/gustavo-vm/dod-etl.

Competing interests

The authors declare that they have no competing interests.

Received: 6 December 2018 Accepted: 30 October 2019
Published online: 20 November 2019

References

1. Malhotra Y. From information management to knowledge management: beyond the'hi-tech hidebound'systems. Knowl Manag Bus Model Innov. 2001;115–34. https://doi.org/10.4018/978-1-878289-98-8.ch007.
2. Watson HJ, Wixom BH. The current state of business intelligence. Computer. 2007;40(9):96–9. https://doi.org/10.1109/mc.2007.331.
3. Sabtu A, Azmi NFM, Sjarif NNA, Ismail SA, Yusop OM, Sarkan H, Chuprat S. The challenges of extract, transform and loading (etl) system implementation for near real-time environment. In: 2017 International Conference On Research and Innovation in Information Systems (ICRIIS). IEEE; 2017. p. 1–5. https://doi.org/10.1109/icriis.2017.8002467.
4. Ellis B. Real-time Analytics: Techniques to Analyze and Visualize Streaming Data. Konstanz: Wiley; 2014.
5. Mesiti M, Ferrari L, Valtolina S, Licari G, Galliani G, Dao M, Zettsu K, et al. Streamloader: an event-driven etl system for the on-line processing of heterogeneous sensor data. In: Extending Database Technology. Konstanz: OpenProceedings; 2016. p. 628–31.
6. Naeem MA, Dobbie G, Webber G. An event-based near real-time data integration architecture. In: 2008 12th Enterprise Distributed Object Computing Conference Workshops. IEEE; 2008. p. 401–4. https://doi.org/10.1109/edocw.2008.14.
7. Zhang F, Cao J, Khan SU, Li K, Hwang K. A task-level adaptive mapreduce framework for real-time streaming data in healthcare applications. Futur Gener Comput Syst. 2015;43:149–60.

8. Jain T, Rajasree S, Saluja S. Refreshing datawarehouse in near real-time. Int J Comput Appl. 2012;46(18):24–9.
9. Kreps J, Narkhede N, Rao J, et al. Kafka: A distributed messaging system for log processing. In: ACM SIGMOD Workshop on Networking Meets Databases. New York; 2011. p. 1–7.
10. Apache. Apache Beam. 2015. https://beam.apache.org/. Accessed 22 Mar 2019.
11. Zaharia M, Das T, Li H, Shenker S, Stoica I. Discretized streams: An efficient and fault-tolerant model for stream processing on large clusters. HotCloud. 2012;12:10.
12. Vassiliadis P, Simitsis A. Near real time etl. In: New Trends in Data Warehousing and Data Analysis. New York: Springer; 2009. p. 1–31.
13. Thalhammer T, Schrefl M, Mohania M. Active data warehouses: complementing olap with analysis rules. Data Knowl Eng. 2001;39(3):241–69.
14. Karakasidis A, Vassiliadis P, Pitoura E. Etl queues for active data warehousing. In: Proceedings of the 2nd International Workshop on Information Quality in Information Systems. ACM; 2005. p. 28–39.
15. Azvine B, Cui Z, Nauck DD, Majeed B. Real time business intelligence for the adaptive enterprise. In: E-Commerce Technology, 2006. The 8th IEEE International Conference on and Enterprise Computing, E-Commerce, and E-Services, The 3rd IEEE International Conference On. New York: IEEE; 2006. p. 29.
16. Sahay B, Ranjan J. Real time business intelligence in supply chain analytics. Inf Manag Comput Secur. 2008;16(1):28–48.
17. Nguyen TM, Schiefer J, Tjoa AM. Sense & response service architecture (saresa): an approach towards a real-time business intelligence solution and its use for a fraud detection application. In: Proceedings of the 8th ACM International Workshop on Data Warehousing and OLAP. New York: ACM; 2005. p. 77–86.
18. Wibowo A. Problems and available solutions on the stage of extract, transform, and loading in near real-time data warehousing (a literature study). In: 2015 International Seminar On Intelligent Technology and Its Applications (ISITIA). New York: IEEE; 2015. p. 345–50.
19. Dean J, Ghemawat S. Mapreduce: simplified data processing on large clusters. Commun ACM. 2008;51(1):107–13.
20. Waas F, Wrembel R, Freudenreich T, Thiele M, Koncilia C, Furtado P. On-demand elt architecture for right-time bi: extending the vision. Int J Data Warehous Mining (IJDWM). 2013;9(2):21–38.
21. Wiesmann M, Pedone F, Schiper A, Kemme B, Alonso G. Understanding replication in databases and distributed systems. In: Proceedings 20th IEEE International Conference on Distributed Computing Systems. New York: IEEE; 2000. p. 464–74.
22. Thomson A, Diamond T, Weng S-C, Ren K, Shao P, Abadi DJ. Calvin: fast distributed transactions for partitioned database systems. In: Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data. New York: ACM; 2012. p. 1–12.
23. Sovran Y, Power R, Aguilera MK, Li J. Transactional storage for geo-replicated systems. In: Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles. New York: ACM; 2011. p. 385–400.
24. Polyzotis N, Skiadopoulos S, Vassiliadis P, Simitsis A, Frantzell N-E. Supporting streaming updates in an active data warehouse. In: Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference On. New York: IEEE; 2007. p. 476–85.
25. Polyzotis N, Skiadopoulos S, Vassiliadis P, Simitsis A, Frantzell N. Meshing streaming updates with persistent data in an active data warehouse. IEEE Trans Knowl Data Eng. 2008;20(7):976–91.
26. Bornea MA, Deligiannakis A, Kotidis Y, Vassalos V. Semi-streamed index join for near-real time execution of etl transformations. In: 2011 IEEE 27th International Conference On Data Engineering (ICDE). New York: IEEE; 2011. p. 159–70.
27. Naeem MA, Dobbie G, Weber G, Alam S. R-meshjoin for near-real-time data warehousing. In: Proceedings of the ACM 13th International Workshop on Data Warehousing and OLAP. New York: ACM; 2010. p. 53–60.
28. Shi J, Bao Y, Leng F, Yu G. Study on log-based change data capture and handling mechanism in real-time data warehouse. In: Computer Science and Software Engineering, 2008 International Conference On. New York: IEEE; 2008. p. 478–81.
29. Neumeyer L, Robbins B, Nair A, Kesari A. S4: Distributed stream computing platform. In: 2010 IEEE International Conference On Data Mining Workshops (ICDMW). IEEE; 2010. p. 170–7.
30. Toshniwal A, Taneja S, Shukla A, Ramasamy K, Patel JM, Kulkarni S, Jackson J, Gade K, Fu M, Donham J, et al. Storm@ twitter. In: Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data. New York: ACM; 2014. p. 147–56.
31. Carbone P, Katsifodimos A, Ewen S, Markl V, Haridi S, Tzoumas K. Apache flink: Stream and batch processing in a single engine. Bull IEEE Comput Soc Tech Comm Data Eng. 2015;36(4):28–38.
32. Google. Google Dataflow. 2015. https://cloud.google.com/dataflow/. Accessed 23 Mar 2019.
33. Microsoft. Azure Stream Analytics. 2015. https://azure.microsoft.com/en-us/services/stream-analytics/. Accessed 23 Mar 2019.
34. Cutting D. Apache Avro. 2009. https://avro.apache.org/. Accessed 10 Aug 2019.
35. Hunt P, Konar M, Junqueira FP, Reed B. Zookeeper: Wait-free coordination for internet-scale systems. In: USENIX Annual Technical Conference, vol. 8. Boston: USENIX; 2010. p. 9.
36. Mueller T. H2 Database. 2012. http://www.h2database.com/. Accessed 10 Aug 2019.
37. Codd EF, Codd SB, Salley CT. Providing olap (on-line analytical processing) to user-analysts: An it mandate. Codd Date. 1993;32:24.
38. Giovinazzo WA. Object-oriented Data Warehouse Design: Building a Star Schema. Upper Saddle River: Prentice Hall PTR; 2000.
39. Stamatis DH. The OEE Primer: Understanding Overall Equipment Effectiveness, Reliability, and Maintainability, 1 pap/cdr edn: Productivity Press; 2010. http://amazon.com/o/ASIN/1439814066/. Accessed 12 Aug 2018.
40. Ljungberg Ö. Measurement of overall equipment effectiveness as a basis for tpm activities. Int J Oper Prod Manag. 1998;18(5):495–507.
41. International Society of Automation. Enterprise-control system integration American national standard ; ANSI/ISA-95.00. Research Triangle Park: ISA; 2001.

Publisher's Note