Fikri *et al. J Big Data* (2019) 6:97
https://doi.org/10.1186/s40537-019-0260-x

Journal of Big Data

RESEARCH Open Access

An adaptive and real-time based architecture for financial data integration

Noussair Fikri*, Mohamed Rida, Noureddine Abghour, Khalid Moussaid and Amina El Omri

*Correspondence:
noussair.fikri@gmail.com
MOSMBD2C, Faculty of Science AinChock, University of Hassan II, Casablanca, Morocco

Abstract

In this paper we are proposing an adaptive and real-time approach to resolve real-time financial data integration latency problems and semantic heterogeneity. Due to constraints that we have faced in some projects that requires real-time massive financial data integration and analysis, we decided to follow a new approach by combining a hybrid financial ontology, resilient distributed datasets and real-time discretized stream. We create a real-time data integration pipeline to avoid all problems of classic Extract-Transform-Load tools, which are data processing latency, functional miscomprehensions and metadata heterogeneity. This approach is considered as contribution to enhance reporting quality and availability in short time frames, the reason of the use of Apache Spark. We studied Extract-Transform-Load (ETL) concepts, data warehousing fundamentals, big data processing technics and oriented containers clustering architecture, in order to replace the classic data integration and analysis process by our new concept resilient distributed DataStream for online analytical process (RDD4OLAP) cubes which are consumed by using Spark SQL or Spark Core basics.

Keywords: ETL, OLAP, Financial, Ontology, RDD, Real-time, Discretized, Stream, Apache spark, Kafka, Producer, Consumer, Containers

Introduction

The main goal of building an ETL is production data integration. Information systems have a various and large datasets of data following a relational schema in the most cases [1]. Companies use several tools for data preparation before loading into data warehouse, Microsoft SSIS, IBM Cognos, SAP Business Objects, Pentaho Data Integration and other tools are available for this type of tasks. Data warehouse provides a structured and cleansed data for reporting in order to provide metrics which allow firms to make the best decisions in business driving [2]. Financial reporting is based on financial metrics that allow internal and external regulatory and auditing bodies to evaluate the company's health. These metrics are extracted from financial data warehouse which contains datamarts of financial statements like statement of financial position. Companies are in obligation to financial statements delivery to government regulatory bodies [3], for example the general direction of taxes in Morocco requests mandatory statements delivery for every quarter then every year. Internal auditing needs also these financial statements to have an eagle eye on their business status. Financial data are seen as a large amount of various data, so there is several challenges in financial data processing. The

Springer Open

© The Author(s) 2019. This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

first challenge is volume and velocity of data, ETL must follow an optimal approach for fast processing, especially for financial data which needs complex calculations [4]. The second is variability, metadata varies between information systems, and ETL must deal with various business metadata to be adaptive with information systems [5]. And the third is real time data processing to deal with in-time report delivery for an up-to-date financial metrics [6]. Using a big data approach is the best way to improve large scale data processing. After Hadoop, the emergence of Apache Spark with its implementation of Hadoop programming model improved by the new concept of resilient distributed datasets [7], the programming interface of Spark process data 20× faster than Hadoop. The proposed ETL is designed for a clustered data processing using [8] RDD concept as micro-batches. Cluster building process becomes more and more easy now, choosing between virtual machines and containers was an easy task [9–11]. Containers provide an easy workflow of shipment and are handier in native resources management [9]. It allows us to define a dedicated control unit for each component of our architecture. Each company has its own information system, it can be a centralized solution with all modules (Accounting receivables, Accounting payables, Stock management…), and several companies' use SAP solutions, Microsoft Dynamics solutions (Dynamics AX and Dynamics CRM), or another customizable solutions. And the rest rather use their own builder solutions. The main issue is heterogeneity of these information systems relational schema. We need to have a unified business metadata base [12]. Using ontologies is the best approach to produce a universal model for metadata mapping between heterogeneous relational schemas [13]. We are focused on financial data integration, from which we choose financial industry business ontology (FIBO) for its rich content which is dedicated to financial contexts [14]. FIBO is used as global ontology and for each company, there is a local ontology which is defined for its information system metadata especially for financial modules. Most of the cases, companies handle their accounting folders in Sage as accounting system with manual process of seizures transfers from sub ledgers to general ledger. It allows us to define a common local ontology as information system local ontology (ISLO). By combining FIBO and ISLO we produce hybrid financial ontology (HFO) to have an enforcement of financial metadata. Connecting companies' information systems to a streaming unit is a good approach to provide an up-to-date metrics. Apache Kafka is the best framework for oriented HFO RDDs [15], and this, by using a discretized stream of Apache Spark. Both of producers and consumers are used for this approach [8, 16]. The producers are connected to companies' information system to convert inserted, updated and deleted records into HFO key/pairs. Then comes the publishing process, which is in charge of HFO key/pairs publication into specific Kafka topics. The consumers gets data from topics as RDDs and submit them into processing units. A series of transformations are applied on data to provide a prepared content for RDD oriented cubes. In this article we are going to make a study on ETL and data warehousing concepts, big data frameworks, streaming technics and the principle of containers in distributed computing, to define an optimal approach for adaptive and real time financial data integration. To resolve classic ETLs problems, reduce latency of data processing, we are going to use a big data dedicated technologies with an enhanced data processing pipeline using a based ontology metadata and resilient distributed datasets. Financial data are dispatched to cluster with embarked metadata information. Our

approach is based on data acquisition as key/value which are streamed as resilient distributed datasets using 2 types of streaming, the first is datasets stream and the second is entry stream, with a hybrid financial ontology based keys. It is based on data processing by using a series of transformations on a grouped datasets or entries by hybrid financial ontology key. Transformed datasets are sent to in-memory data warehouse which is consumed as a resilient distributed datasets based OLAP cube. We will present a state of art of all related concepts and researches, then, introduce our approach details, experiment it, compare it to a classic ETL, and finally discuss evaluations and limitations of our approach.

Literature review

Business intelligence

Business intelligence has become a crucial element in data analytics, especially [5] for report delivery by using a data driving decisions concept. Before using Business intelligence Tools in order to have a multi-dimensional data exploitation and produce an interactive data view for end-users, we have to extract production data from information systems which are stored in a many heterogeneous data sources, and [1] transform these data into a suitable format by removing corrupted and duplicated data, checking data integrity and formatting it, then involving corrections and cleansing, and finally applying joins, specific calculations and keys definition.

- Basic transformations: cleaning, deduplication, format revision and key restructuring
- Advanced transformations: derivation, filtering, joining, splitting data, validation, summarization, aggregation and integration.

The output of previously described process will be loaded into a data warehouse for analytics, report, a knowledge base or another reliable data source. The ETL process is defined by 3 layers: staging layer (for storing extracted data), data integration layer (representing data in transformation process) and access layer (entry point for reliable and callable data) [1, 2, 5]. After extracting, transforming data, an obligatory process is called for checking if data is ready to be loaded into data warehouse or not [12, 17]. This process is named "ETL Testing", and it's an involvement of the following tasks:

- Understanding a functional meaning of used data for a good reporting.
- Definition of source/target mapping, checking (format, integrity, accuracy).
- Data comparison between the source and the target system.

There are 2 type of data testing, ETL and Database testing, the first one involve validation and verification on data in the warehouse, and the second on transactional systems. After testing, we get a centralized database of cleansed and transformed data, that we call data warehouse, and it has specific features for decision making, this data container is a subject oriented [17, 18]. A data warehouse is mainly used by managers, business analysts or another user who has no technical skills, but a functional view of data, according to their experience, they need a simplified and on-demand data for analyzing business and reporting. An OLTP database contains a primitive and highly detailed

data, not ready yet for analysis: ETL's job. An OLAP data warehouse provides a summarized and multidimensional view of data for simple information get operations and accurate packet of information illustrated by 3 types of schemas [1]: Star Schema, Snowflake Schema and Fact Constellation Schema as galaxy schema. In a data warehouse, we need a road-map or a data abstraction to define warehouse objects and its directory for simplified access. This procedure is Metadata attributions, in order to have data about data with specific number of abstraction levels, depending on business contexts. We can also have a metadata repositories: business metadata, operational metadata, technical metadata or an advanced repository of an embarked algorithms of summarization etc. Data exposition is a crucial parameter in data warehouse organization, a cubic view can spare us too much efforts in reading process, a data cube is the optimal data structure, its multidimensional representation is based on entities as dimensions [1]. What is a Data mart? For each department, there is a part in data warehouse which is dedicated to it, for example in a specific organization, sales and distribution are considered as data marts, in real life, separately each of them is a department, we can see this as a way to impose access control strategy and enhance performance by making focus on a specific volume in data processing. A data warehouse rides several updates, its architecture must have a dynamic and extensible management system, and this feature is included among required elements for data warehouse delivery. A three-tier architecture is recommended to deal with BI's processes flow, staging layers and data flow. Starting from the bottom to the top, the first tier is the relational [1] database or another type of data source, and the layer where a data extraction, cleansing, loading and refreshing functions are performed. The second tier is the OLAP server, where outputting data is used by following 3 different models. A relational OLAP (ROLAP), multidimensional OLAP (MOLAP) and hybrid OLAP (HOLAP), a mixed approach, which combines advantages of ROLAP and MOLAP [19]. The manipulation of the OLAP multidimensional view of data amounts to 5 main operations; Roll-up, Drill-down, Slice, Dice, Pivot. For adjusting management of a data warehouse, involving its performance and having an easy procedure on backup and recovery, the best way is a data warehouse partitioning. A data warehouse, seen as system, must have several components, and each of them is managed by a handler which is a virtual entity who acts for system behavior monitoring.

Big Data and Apache Spark

The massive data processing or what we call Big Data challenges is the most coveted domain in this last decade. There are several technologies with an advanced tools and frameworks which are designed for satisfying technical requirements. These technologies are in the form of an ecosystem with an optimized algorithms and pipelines for a fast data processing. After Apache Hadoop, and its approach of structured and unstructured data processing, an emerged technology has resolved latency problem on processing time for a large amount of data, with a unified ecosystem, which is based on in-memory concept of data storage and processing as shown in Fig. 1, input data are distributed to multiple workers and stored in RAM of each worker depending on persistence strategy.

Apache Spark is considered as the fastest and the most scalable ecosystem for data analytics in academic and industrial area, and one of the important open source projects

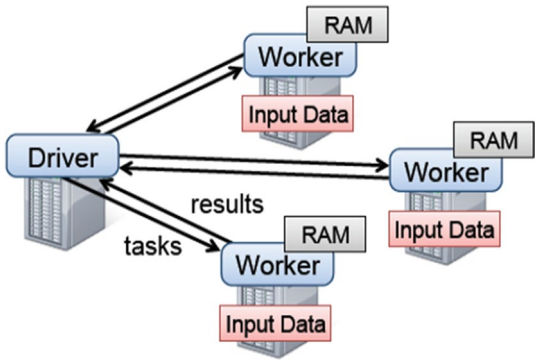

Fig. 1 Apache Spark, launching application on multiple workers

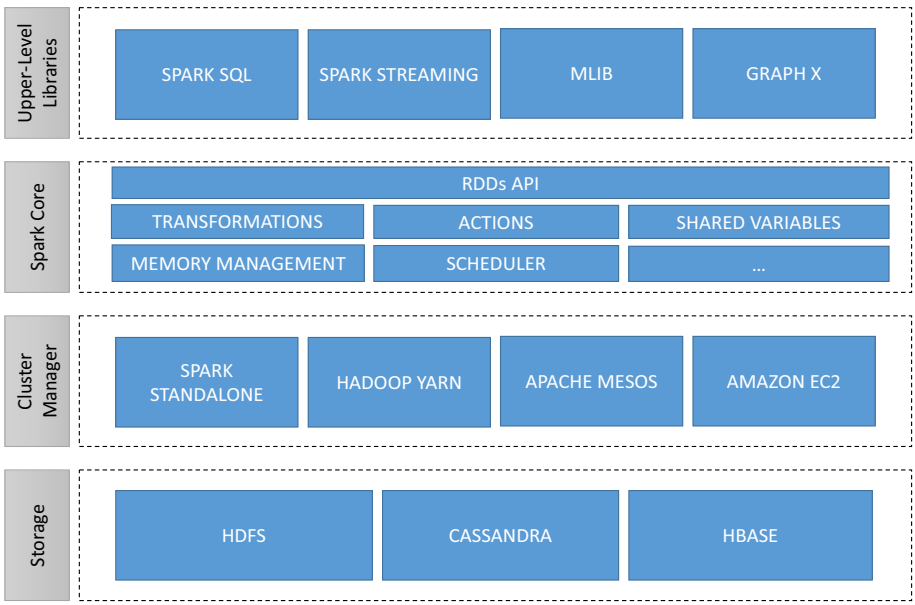

Fig. 2 Apache Spark Stack and libraries

in Apache Software Foundation. Figure 2 shows libraries which rotates around Spark, in other terms, the Apache Spark stack [8].

Spark Core, the basis of the whole system, it provides an essential panel of operations, like transformations, action and shared variables to prepare data for the next-level components, and the management components who take cares of memory management, scheduling process, shuffler, and interpreter integration, for the interaction with the low-level components like, the Cluster manager for distributed computing and Storage manager for an optimal I/O movements [7]. The top-level libraries of Spark ecosystem are based on Spark core essentials to deals with many workloads. Before introducing these libraries, let's go a little deep into Spark programming model, the *resilient distributed dataset* concept. It's a collection with fault tolerance capacity that can be paralyzed with a driver program's collection or an external data source like HDFS. It use the key-value

pair approach to dealing with transformation, actions and shuffling operations. Every dataset state across applied operations is saved in memory for a fast computing and data access by processing algorithms. Spark provides 2 datasets saving types, PERSIST and CACHE, and we can choose from them by storage levels: MEMORY_ONLY, MEMORY_AND_DISK, MEMORY_ONLY_SER, MEMORY_AND_DISK_SER, DISK_ONLY, MEMORY_ONLY_2 and MEMORY_AND_DISK_2. For an optimized cache management, the least-recently-used (LRU) states of dataset are removed. In a cluster mode Spark operations (map or reduce) are executed by using remote nodes, so for each variable which is used by these operations, there is copy, on each node. These shared variables are propagated from a node to another, and are divided into 2 types. Broadcast variables are cached on each node of the cluster with read-only status, and are managed by broadcast algorithms for an optimized inter-node communication. Accumulators are used for an associative and commutative operations through nodes, especially for Map and Reduce operations or sums, by using numeric types or custom types. The upper-level layer of Spark contains 4 main libraries.

Spark SQL: this component is dedicated to structured data, it provides a simple interface that combines data structure and its computation as extra information for an extra optimization. It's based on Datasets and DataFrames interfaces for data models, and SQL for data manipulation. By combining the resilient distributed datasets and Spark SQL's engine strength, the result is an object called Dataset which is easy and adapted for Map, FlatMap or Filter transformations. A Dataframe is an oriented column dataset, it follows the relational model of an RDBMS, in other words, a structured table with rows and columns. Building a dataset based on an RDD can be done by 2 technics; applying reflection on an RDD by inferring the schema with specific types of objects, or building a schema then project it on the current RDD. Excepting the predefined aggregations (count, average, sum) on a Dataframe, the user can define its own aggregate function. There is 2 types of user-defined aggregation functions, Untyped and Type-safe custom functions. The strength of Spark SQL is its capacity of handling relational transformations on various datasources, duplicate a Dataframe into temporary view for SQL queries that could be stored as persistent table and allows bucketing, sorting and partitioning. Spark SQL provides several features which are useful for building a good ETL pipeline, for semi-structured/unstructured data (with flat files formats like CSV, JSON etc.) or structured data (JDBC or others). This pipeline produce a ready content for analytics and processing, data is cleansed and curated by a sequence of transformations. Spark SQL is a good choice for building an ETL pipeline because of its flexibility and adaptation to many datasource types, moreover it can deal with corrupted or malformed data by using the high-order functions in SQL, interoperating with one or several Dataframes.

Spark Streaming: before introducing this component, there is a quick overview of data streams. An unbounded sequence of data which comes every interval of time is called a data stream, the window between two data arrivals may vary. These package of data follows a basic structure of key-value pair, like relational databases or a complex view like graphs. Nevertheless, some differences exist between a data stream and dataset, a static data state against a continuous sequence of data, in the form of blocks of data, where each processed stream is discarded for memory optimization. A dataset can be stored in memory, but storing a data stream is an impossible operation because of continuous

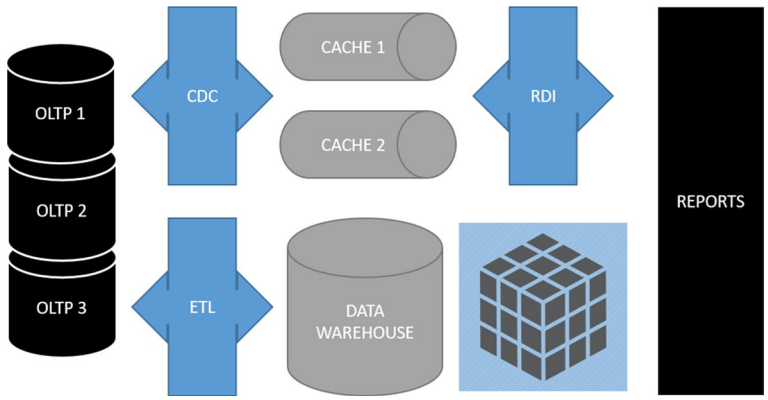

Fig. 3 SOA architecture using CDC

data injection. The interval between two arrivals must allow data processing to avoid queuing. A system of data streams management has to consider also the concept of data drift which is an unexpected mutation that may affect the distribution. Spark Streaming is established for live data processing, where data may come from many sources or TCP sockets for a continuous data stream. It uses a discretized stream approach or DStream for a continuous stream of an RDD's represented as a sequence of data for input receivers. Over transformations, the RDD content changes and produces a sliding status, from which a windowed DStream are defined [8, 20]. This operation needs two parameters; Window length, which is the period of a window and sliding interval which defines the moment when window starts and window ends. This component can be used for algorithms streaming, like logistic regression streaming for a real time learning. It also uses the classic Spark approach of RDD caching and persisting, and for a continuous and fault tolerant streaming, it uses a Checkpoint of every system status variables, by two types of Checkpointing; Metadata Checkpointing for elements which rely to configuration, operations and scheduled tasks (Batchs), and Data Checkpointing for elements which concerns the RDD's.

Real-time data integration

There are many proposed solutions for near real-time data warehouse and architectures for concrete real-time data warehouse system, like service-oriented architecture (SOA) based on change data capture (CDC) for real time data acquisition on changes events [6, 21]. SOA is used to separate data consumers from data controllers, in other words a self-contained architecture and can be accessed remotely. There is a lot of implementation of this concept like Web services based on Web Service Description Language (WSDL) and representational state transfer (REST) which are the most used SOA implementation. CDC is used for resources optimization, instead of full data reload, this approach allows update on target data in hot database exploitation.

Figure 3 shows how change data capture is integrated in service-oriented architecture. Changed data are captured from OLTP data sources submitted to caches in order to avoid full execution of ETL from the beginning. A great gain in processing time.

Apache Kafka is a distributed publish-subscribe messaging system initially developed by LinkedIn [15, 16]. The goals of using it are generally for real-time data exchanges between systems or applications, and real-time Applications which depends on data volatility. It's based on 4 main core APIs:

- Producer API: streams of records publication from applications to Kafka topics.
- Consumer API: streams of records subscription from Kafka topics to applications.
- Streams API: acts like producer API and consumer API, get streams of data as input, processing it and produce streams of transformed data as output.
- Connector API: dedicated to a group of producers and consumers for Kafka topics to applications or data systems connections.

Virtual machines and containers

The virtualization concept is based on emulation strategy, where the host server is virtualized, in other words, the same hardware resources are shared between virtual machines with limitations and quotas. And each VM has its own operating system with binaries and libraries, which is monitored by a hypervisor [10]. It manages VMs from its creation to running, and acts between hardware layer and VMs layer. Virtualization allows companies to avoid expensive hardware costs and increases a separated applications running efficiency [9]. The main advantages of virtualization is that security and management technics are already defined and there is no need of extra configuration for a good processing. Containerizing concept uses a lightweight strategy, instead of hardware virtualization, the host operating system is virtualized. Hypervisor is replaced by containers manager which sits on the top of OS, where binaries and libraries are shared in read-only mode. This approach allows a multiple workloads running on one kernel. Linux containers (LXC) is the first container technology which allows an isolated linux systems to runs on a single host, then Docker Project comes with several updates on LXC containers to add portability and flexibility for a best containers management like containers shipment and running [9–11]. The use of containers provides several benefits like using native performance, startup time and memory space requirement.

Financial reporting

IFRS: international financial reporting standards are defined to update the older standards of IAS and covers transparency of financial activities in European Union and several countries in Asia and South America but not in United States which use General accepted accounting standards [22].

IFRS established these mandatory rules to handle business practices [3, 18, 23]:

- Statement of financial position: IFRS influences how components of balance sheet are reported. It includes:
  - Assets: current and non-current.
  - Liabilities: current and non-current.
  - Equity: shareholders' equity and retained earnings.

- Statement of comprehensive income: includes
  - Profit statement.
  - Loss statement.
  - Other income statement.
  - Property and equipment statement.
- Statement of changes equity: Is a documentation of earning benefits for a given financial period
- Statement of cash flow: Is a company's summarization of operations, investing and financing as transactions entries.
- Statement of profit and loss: includes
  - Gross profit.
  - Operating profit.
  - Profit before income tax.
  - Profit from continuing operations.
  - Profit for the period.
  - Earnings per share ratios.

A company's information system includes largely a financial module which contains a summarization of its financial activity, in the form of transaction system called general ledger. All financial activities are sourced from it as statements following IFRS or GAAP standards. The main goal of using general ledger is to centralize all financial activities into a common data structure, defined as COA: chart of account, to have a representation of all business segments like business units, cost centers, products, projects, and accounts, in order to provide a classification of financial measures. Business transactions details are recorded in a sub transaction systems of general ledger called sub-ledgers. The main purpose of having sub ledgers is granular level of financial activity, and the well-known of them are:

- Sales orders.
- Corporate debt.
- Accounts payable invoices.
- Purchase orders.
- Customer receivables.

The process of posting to general ledger is an aggregation of the financial activities recorded in sub-ledgers and journal entries using the segments of chart of accounts. There are 2 cases of accounting systems, an integrated and a non-integrated components in enterprise resource planning system, depending on how company handle its accounting activity [24]. Some firms use an auxiliary applications in order to track their financial activities. Then comes the famous closing process, which gets, for a specific period all transactions from sub-ledgers and post them to general ledger. For each accounting period, no more entry is permitted on general ledger after closing

process. In this stage, financial statement can be sourced to reporting entities, who's in charge of repots delivery.

Among financial report delivery complications, the main issue is accessing data [25]. The need of on-demand financial data produces several questions about building a robust financial data warehouse. General ledger and sub-ledgers are widely separated, so retrieving data for detailed and accurate report is an intelligence challenge based on chart of accounts segments. To avoid this problem, we need to consolidate general ledger and sub-ledgers in a common repository, in other words, a structured and detailed data warehouse. There are several requirements linked to what kind of data should be retrieved? Who is allowed to retrieve it? And what's the appropriate dimensional model to use?

The main purpose of having a real time reporting system is applied updates on financial metrics for on-demand report delivery. Not only for in-time report delivery to regulatory bodies but also for firm's internal decision-making. Another purpose [25] including:

- Reporting efficiency which requires substantial time and resources intensions, a self-service report consumption as much as flexible and real-time collaboration between different business units.
- Reporting process automation, for example, the posting process to general ledger or financial consolidation between two business units.
- Real-time data benefits: based on Aberdeen group survey made by Nick Castellina, using real-time approach showed time optimization and quality efficiency in report delivery.

Tables 1, 2 and Fig. 4 shows examples of real-time reports with dynamic values. Top 5 customers and top 10 products tables reload values on each order. The same process is applied on Sales by product type graphic.

Ontologies

Using ontology-based approach becomes useful technic in World-Wide Web for categorization [13, 14, 26]. The W3 consortium developing a dedicated framework for knowledge base builds, the Resource Description Framework. These technics are used now in many domains such as Medicine, and other scopes with large vocabularies.

Table 1 Real-time report of top 5 customers

Top 5 customers	
MELLAL Younes	3,600,000 MAD
EDDIHI Anas	1,970,000 MAD
LECHHEB Saad	1,450,000 MAD
LOUAFDI Mehdi	800,000 MAD
BOUJOUF Annouamane	190,000 MAD

Table 2 Real-time report of top 10 products

Top 10 products	
Plasma monitor 10/XGA	15,920,000 MAD
Client ISO CP-S	12,660,000 MAD
Sound card STD	12,002,000 MAD
Laptop 16/8/110	8,507,500 MAD
Laptop 16/8/111	7,440,900 MAD
Manual-vision OS/2.x	3,290,000 MAD
ESD bracelet/clip	2,281,000 MAD
CD writer, read only, speed 8x	1,177,700 MAD
CD writer, read only, speed 8x	1,177,700 MAD
Ergonomic keyboard 113 E/EN	930,100 MAD

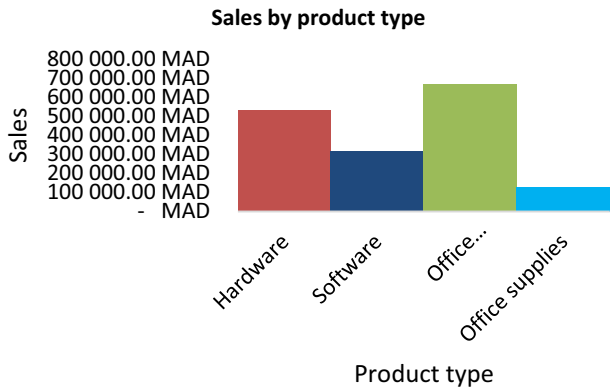

Fig. 4 Sales by products example

The appropriate approach for building an ontological knowledge base, is based on the following steps:

- Defines the scope of covered terminologies.
- Makes this ontology reusable.
- A wide enumeration of vocabularies.
- Defines the class tree.
- Defines properties (slots) of each class
- Defines facets of each slot.
- Finally using instances of class

What is FIBO? The role of financial industry business ontology is to centralize vocabularies of the financial industry [14]. In most of the implementations which rest on the ontology, there are three types of different ontological approaches which can operate separately or together and are used for the explicit description of semantics of the information resulting from heterogeneous databases. Figure 5 shows the hybrid ontological approach [13, 26, 27].

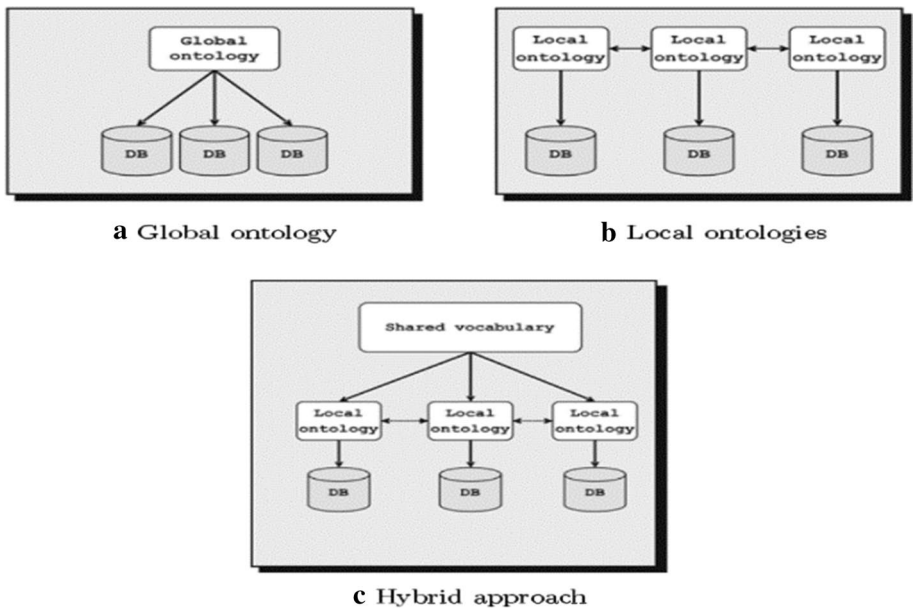

Fig. 5 Hybrid ontology approach

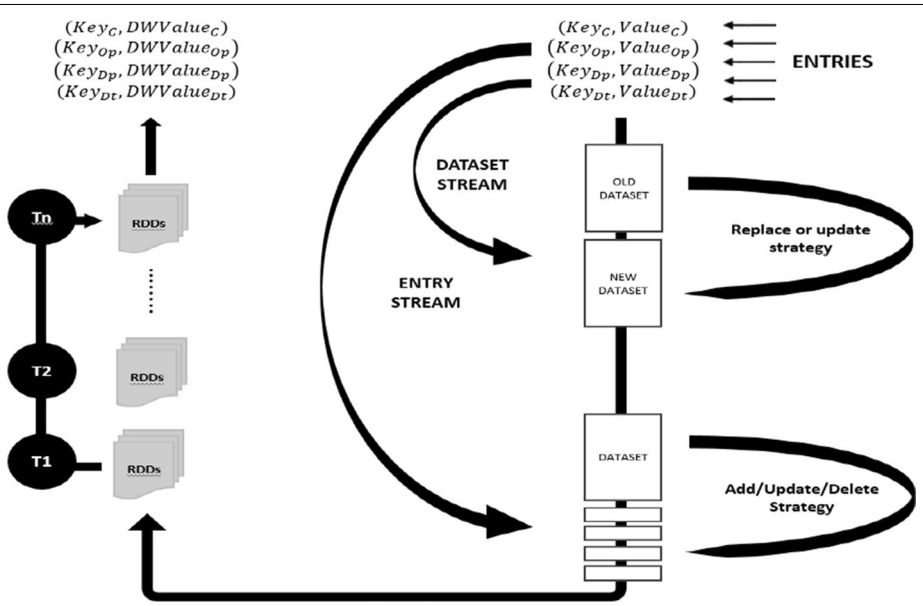

Fig. 6 Data streaming and integration strategies through RDDs

Methods

Our approach is based on watching all new information system data and make them streamed in the form of dataset stream with replace or update strategy or entry stream with add, update and delete strategy, then submitted to a serial transformations (T1, T2,…,Tn) through resilient distributed datasets of key/value pairs (RDDs1, RDDs2, …. RDDsN), and finally stored in financial data warehouse (Fig. 6).

Input side

By using hybrid approach, on input side, we have local ontology (ISLO) which is current information system metadata, then financial industry business ontology (FIBO) as global ontology which contains standardized financial metadata The goal is to produce a hybrid financial ontology (HFO) defined as the following notation:

$$HFO = \text{FIBO} \cup \text{ISLO}$$

where

$$HFO[\text{C}, \text{Op}, \text{Dp}, \text{Dt}]$$

where C is the Class or SubClass; Op is the object properties; Dp is the Data properties; Dt is the data types

The hybrid financial ontology (HFO) has two separated layers, input layer which contains information system local ontology (ISLO) represented by yellow color and FIBO metadata with black color. This HFO mixed metadata are provided as containers or keys for input data in RDDs. Then output layer with the same concept for outputted data to data warehouse. These data are served as RDDs of pairs to build reporting OLAP cubes. As shown on the ontology tree, the symmetry of the 2 layers explains the mapping between them. The main source of financial data is the ERP database or an exclusive accounting system (Fig. 7).

- SI = [HFO, CSL]/SI => Information system
- CSL: Collection of Sub ledgers
- $\text{HFO} = [(C|C \in \{ERP, Accountingsystem, \ldots\}),$
  $(Op|Op \in \{Integratedin, \ldots\}),$
  $(Dp|Dp \in \{Isopen, \ldots\}),$
  $(Dt|Dt \in \{System, Modul \ldots\})]$

A company information system is composed of sub-ledgers which are interfaced between them:

- SL = [HFO, CDS]
- CDS: Collection of dataset

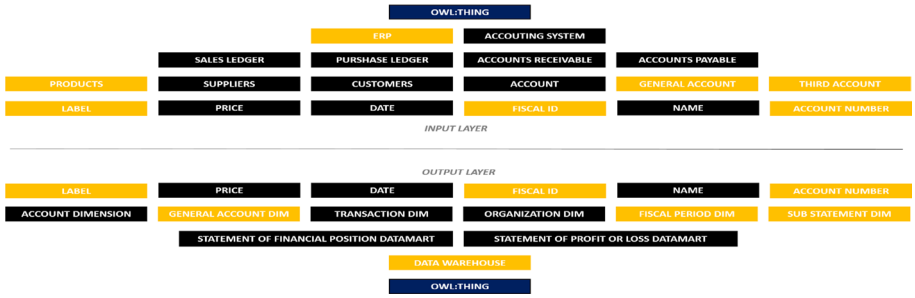

Fig. 7 Simple model of hybrid financial ontology tree

- HFO $= [(C|C \in \{Acc.payable, Acc.receiv, \ldots\}),$
  $(Op|Op \in \{Belongs\ to, \ldots\}),$
  $(Dp|Dp \in \{Narrative, Qualitative \ldots\}),$
  $(Dt|Dt \in \{Sub\ ledger, General\ ledger\})]$

A sub-ledger is a subject oriented aggregation (Ex: Sales) of structured data:

- DS = [HFO, CC]
- CDS: Collection of columns
- HFO $= [(C|C \in \{Customers, Invoices, \ldots\}),$
  $(Op|Op \in \{Default\ requirement, \ldots\}),$
  $(Dp|Dp \in \{Is\ callable, \ldots\}),$
  $(Dt|Dt \in \{Table, View \ldots\})]$

A dataset is a column based structure:

- C = [HFO]
- HFO $= [(C|C \in \{Label, Date \ldots\}),$
  $(Op|Op \in \{Designate, Include \ldots\}),$
  $(Dp|Dp \in \{Variable, Constant \ldots\}),$
  $(Dt|Dt \in \{String, Integer, Bool \ldots\})]$

Transformation pipeline

Each entry has a key/value pair where key is a composite HFO keys or single, where some of those can be null or empty collection, defined as:

$$K_M = K_{SI} \cup K_{SL} \cup K_{DS} \cup K_{CC},$$
$$\text{where} \quad M \in \{SI, SL, DS, CC\}$$

And entries with:

- Class or subclass key are stored among business metadata.
- Object properties are stored among technical metadata.
- Data properties are stored among technical metadata.
- Datatypes are stored among technical metadata.

T is the summation of all minor transformation (t):

$$T = \sum_{i=0}^{N} t_i | t \in \{Map, Reduce, Filter, Group \ldots\}$$

where

$$t = t(C_E) + t(Op_E) + t(Dp_E) + t(Dt_E)$$

with

$$E \in \{SI, SL, DS, CC\}$$

T-Function acts on key, value pair depending on HFO rules, shown on the following expressions:

$$t\big((Key_C, Value_C)\big) = \big(Key_C, DWValue_C\big)$$

$$t\big((Key_{Op}, Value_{Op})\big) = \big(Key_{Op}, DWValue_{Op}\big)$$

$$t\big((Key_{Dp}, Value_{Dp})\big) = \big(Key_{Dp}, DWValue_{Dp}\big)$$

$$t\big((Key_{Dt}, Value_{Dt})\big) = \big(Key_{Dt}, DWValue_{Dt}\big)$$

T-pipeline is handled by master node which is a spark master instance running on container. The minor transformation t_i are dispatched to nodes which are spark workers nodes, as N collection of minor transformation on N nodes.

Output side

The step after transformation is final results storage in data warehouse, so the goal is to produce a HFO based warehouse. Transformed data has datamarts as targets:

- DW = [CDM]/DW => Data warehouse
- CDM: Collection of datamarts

And datamarts is composed of dimensions:

- DM = [HFO, CDIM]
- CDIM: Collection of dimensions
- HFO $= [(C|C \in \{Statem.of\ financial\ position,$ $Statements\ of\ profit\ and\ loss, \ldots\})]$

Dimensions are columns based structure too:

- DIM = [HFO, CC]
- CC: Collection of columns
- HFO $= [(C|C \in \{AccountDim, Transac.Dim,$ $OrganizationDim, FiscalPeriodDim \ldots\})]$

With:

C = [HFO]

- HFO $= [\big(C|C \in \{Label, Date \ldots\}\big),$ $\big(Op|Op \in \{Designate, Include \ldots\}\big),$ $\big(Dp|Dp \in \{Variable, Constant \ldots\}\big),$ $\big(Dt|Dt \in \{String, Integer, Bool \ldots\}\big)]$

RDDs based OLAP cube

After transforming data and serving them as RDDs of pairs, we are going to build RDDs based OLAP cubes, by retrieving dedicated pairs to form sub-cube of data. On each pair, the key is composed of datamart key, cude key, dimension key and finally column key, where each key if a HFO metadata. As shown on Fig. 8, retrieved data are represented by cube faces which build the sub-cube of data.

The main architecture

The main control unit, running on a container as Spark Master, is in charge of handling sparker the streaming control unit, the metadata control unit and processing units.

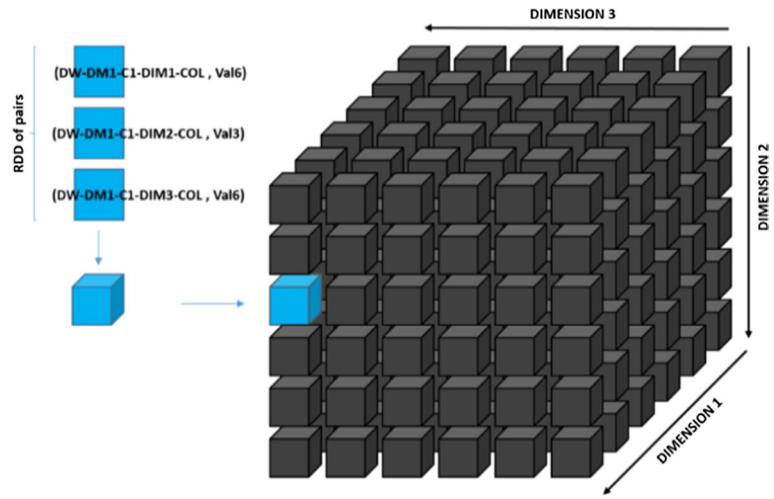

Fig. 8 Resilient distributed dataset based OLAP cube

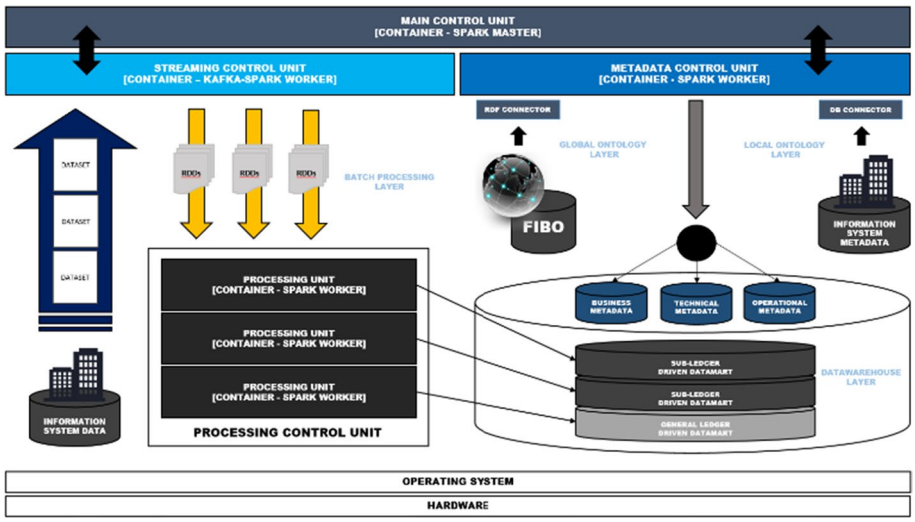

Fig. 9 Our main architecture

Starting from targeted information system, data are streamed by streaming control unit using Kafka Connector API, which is a Kafka-Spark worker running on a container, then dispatched as RDDs to another Spark Workers which are data processing units. Key-value pairs are submitted to a serial transformations, from a Spark Worker to another then resulting RDDs are loaded to data warehouse. The metadata control unit retrieves data from FIBO using an RDF/OWL connector and targeted information system metadata to build HFO keys for RDDs used by streaming and processing control units (Fig. 9).

The metadata control unit use and RDF/OWL connector to retrieve FIBO content as global metadata, and simple database connector to get ISLO content as local metadata. It uses the hybrid ontology approach to produce HFO rules. After extracting needed

metadata, the current unit store produced HFO metadata into business and technical metadata component of data warehouse.

The streaming control unit is composed of producer and consumer which process a continuous discretized stream of RDDs using a structured streaming model based on HFO keys. RDDs are streamed as HFO key/value pairs. The producer gets data from information system data sources as datasets using available strategies (replace or update strategy and add/update/delete strategy). Then producer converts collected datasets to key/value pair and publish them into associated topics following HFO rules. RDDs production is triggered by transaction event on data source (Ex: inserts, updates …). The consumer is in charge of subscription. It consumes published RDDs from topics and submit it into processing units. Processing units executes jobs (T-functions) where each job has a chained steps (minor t transformation) and uses operational metadata component of data warehouse for logging and monitoring. After processing data HFO RDDs are routed to associated dimensions on dedicated datamarts following HFO rules.

Experiment

To perform producing and consuming actions, we are going to use Apache Kafka for continuous publishing and subscribing of RDDs. This streaming API needs Apache Zookeeper in order to insure distributed coordination. The Kafka producer is written in Java, based on JDBC connector, file input stream and excel file readers or other data sources reader. It has also a data source listener for event notification depending on streaming strategies. And finally a publisher which submit data into topics for consumption. The Kafka consumer is written in Scala and use Spark core with Spark streaming libraries for discretized stream manipulations. And finally processing units, which are written in Scala due to its flexibility and simplicity of code integration and recommendations of using Scala on Apache spark. The producer has data sources watchers, for flat files, database and others. In our case, we're going to test producing process by using as data source a CSV files of invoices, customers, products, orders, promotions. The producer watchers detects inserts, updates, deletes and changes on file entries by using the checking differences between old chunks and new chunks. In each case the producer generates a key/value pairs as shown here:

- Inserts case:

The new row is: INV0015261;CUST40010014;Company 14;14/12/2018;;;4320,00;MAD
The produced HFO Key/Value pairs are:
*(**INSERT-SI-AR-INV-14963-INVOICE_ID**, INV0015261)*
*(**INSERT-SI-AR-INV-14963-CUST_ID**, CUST40010014)*
*(**INSERT-SI-AR-INV-14963-CUST_NAME**, Company 14)*
*(**INSERT-SI-AR-INV-14963-INV_DATE,** 14/12/2018)*
*(**INSERT-SI-AR-INV-14963-DUE_DATE**, null)*
*(**INSERT-SI-AR-INV-14963-TRS_NUM**, null)*
*(**INSERT-SI-AR-INV-14963-AMOUNT**, 4320,00)*
*(**INSERT-SI-AR-INV-14963-CURRENCY**, USD)*

- Change case:

On previous inserted pairs we changed due date and transaction number triggered by customer payment:

*(**CHANGE-SI-AR-INV-14963-DUE_DATE**, 31/12/2018)*

*(**CHANGE-SI-AR-INV-14963-TRS_NUM**, 98888*)

- Delete case:

The removed pairs are the previous inserted and changed pairs

The produced HFO Key/Value pairs are:

*(**DELETE-SI-AR-INV-14963-INVOICE_ID**, null)*

*(**DELETE-SI-AR-INV-14963-CUST_ID**, null)*

*(**DELETE -SI-AR-INV-14963-CUST_NAME**, null*

*(**DELETE -SI-AR-INV-14963-INV_DATE,** null)*

*(**DELETE -SI-AR-INV-14963-DUE_DATE**, null)*

*(**DELETE -SI-AR-INV-14963-TRS_NUM**, null)*

*(**DELETE -SI-AR-INV-14963-AMOUNT**, null)*

*(**DELETE -SI-AR-INV-14963-CURRENCY**, null)*

The consumer gets each producer key/value pair as consumer record. In all cases the record structure is the same, it's composed of topic, partition, offset, created time, checksum for integrity, key and value size, and finally the key/value data as shown on the following record:

ConsumerRecord
(
topic = finance,
partition = 0,
offset = 83,
CreateTime = 1543957056448,
checksum = 4200369887,
serialized key size = 36,
serialized value size = 4,
*key = **CHANGE-SI-AR-INV-14963-DUE_DATE**,*
value = 31/12/2018
)

In the processing unit, the transformations pipeline on the top are done by Spark worker or precessing units. It gets the Consumer records as RDDs then maps it to HFO Key/Value pairs to apply changes. By mapping key/value pairs contained by streamed RDDs, the key is parsed using the following rules:

***CHANGE – SI – AR – INV* – 14963– DUE_DATE**
1 2 3 4 5 6

Part 1: Is the process type which is applied on intermediate datasets. In shown key, it's an insert process.

Part 2: <*SI*> is the Information system key on HFO.

Part 3: <*AR*> is the Accounting receivables key on HFO.

Part 4: <*INV*> is the Invoices key on HFO.

Part 5: Shows order id of the differences in chunk

Part 6: <*DUE_DATE*> is targeted column.

The processing units get consumer records then dispatch them to Spark Workers, the transformation pipeline is composed of 1 transformation on customers RDDs:

1. Replacing **(,)** by **(.)** in **AMOUNT**
2. Converting **AMOUNT** to MAD by **AMOUNT*** ***USDtoMAD*** value which is currently: **9,62**

After processing transformations on pairs, the outputted key/value pair for ***CHANGE – SI – AR – INV –*** * **– AMOUNT** is mapped as:

CHANGE – DW – AR – INV – **14963– AMOUNT**					
1	2	3	4	5	6

Part 1: Is the process type which is applied on intermediate datasets. In shown key, it's an insert process.

Part 2: <*DW*> is the Data Warehouse key on HFO.

Part 3: <*AR*> is the accounting receivable datamart key on HFO.

Part 4: <*INV*> is the Invoices key on HFO.

Part 5: Shows order id of the differences in chunk

Part 6: <*AMOUNT*> is targeted column.

After transformations the produced RDDs are structured following hybrid financial ontology tree. The HFO key contains the process type (INSERT–CHANGE–DELETE) the data warehouse, the datamart and finally the dimension which RDD key/value pair is assigned. The persistence type can be specified in Spark configuration. It can be stored on memory only, memory and disk (lack of memory case) or disk only.

To retrieve invoice lines of a specific datamart (account receivable), a specific dimension (invoice dimension) then a specific order id (14963 in this example), and simultaneously following HFO tree:

```
currentDW = testETL.groupByKey('DW)
currentDM = currentDW.groupByKey('AR')
currentDIM = currentDM.groupByKey('INVDIM')
invoiceLines = currentDIM.groupByKey('14963')
Or
invoiceLines = testETL.groupByKey('14963')
                      .groupByKey('AR')
                      .groupByKey('INVDIM')
                      .groupByKey('14963')
```

To apply sum operation on retrieved invoice lines:

```
invoiceTotal = invoiceLines.reduceByKey((total,value)=> total + value)
```

To get total value of this invoice (related to order id: 14963)

```
invoiceTotal.collect ()
```

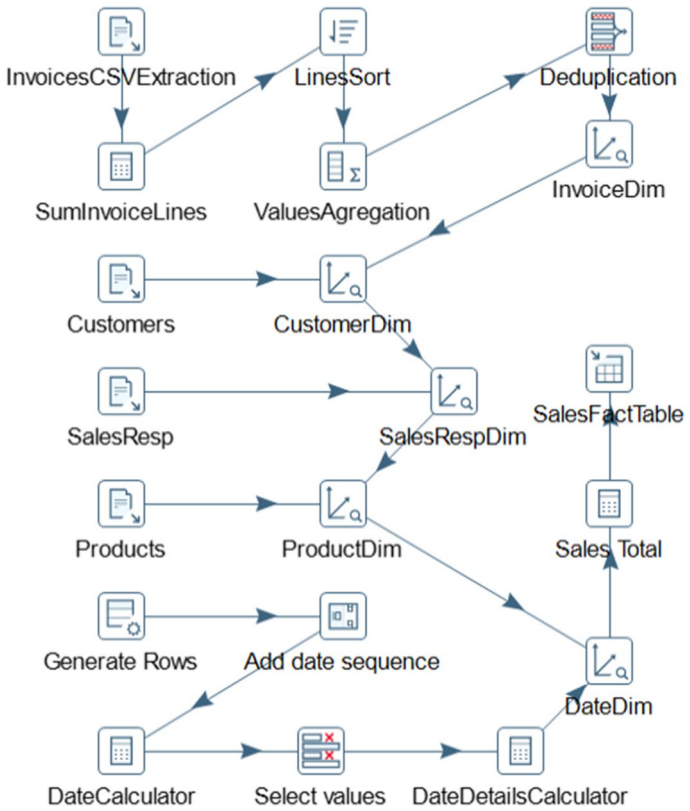

Fig. 10 Simple data integration dataflow on Pentaho Data Integration

Table 3 Pentaho data integration and our solution benchmark

	PDI	Our approach
Business metadata	Manual	Semi-automatic
Technical data	Automatic	Automatic
Distributed processing	Not integrated	RDD approach
Data streaming	Not integrated	Discretized stream
Data processing	Tuples	Key/value pairs
Data warehouse type	Database	HFO based datasets
OS compatibility	OS with JVM	OS with JVM

Results

Our tests are done by using Ubuntu 18.04 + Docker containers for worker nodes, running on a physical machine with Intel i5 2.2 GHZ 4 CORE processor and 8 GB of RAM. For evaluation of our system on add operations of X records [10 k, 50 k, 100 k, 250 k, 500 k] in invoices.csv. The producer of streaming unit, use a file watcher and producer which are in charge to get events on files and produce chunks as key/value pairs records. Then consumer gets streamed key/value pairs as RDD then submit it to processing units using the following code. The used ETL tool for evaluating processing technics of our solution is Pentaho Data Integration. We used flat file reader to read *invoices.csv* content and

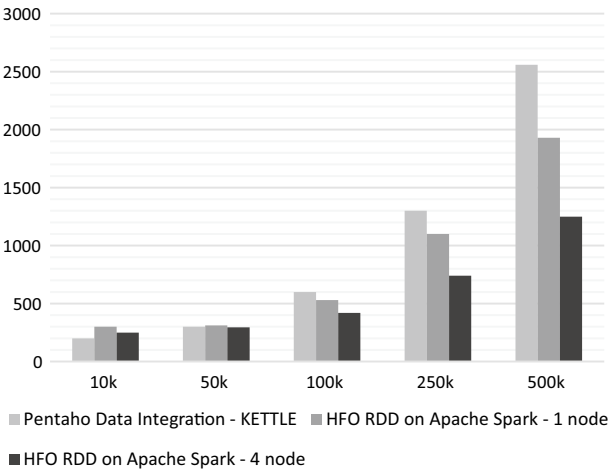

Fig. 11 Processing time comparison

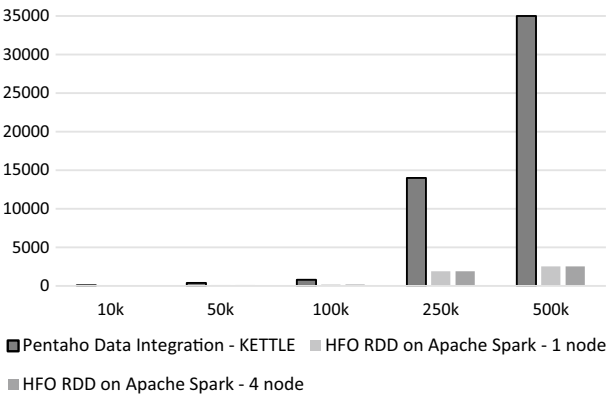

Fig. 12 Memory consumption comparison

Dimension Lookup to load data in Transaction dimension of our testing Data Warehouse. For our transformations we used aggregation by SUM operation of invoice lines by invoice and grouping them by invoice id then deduplication. The invoices CSV files are splited into 5 partitions for tests We had created a Kettle job with the same transformations, and run it every file changing event. Figure 10 shows the implementation of PDI data processing flow of all transformations by using 3 additional flat files (Products, Salesman, Customers).

Table 3 shows main differences between our system and PDI (Pentaho data integration) architecture. In PDI business metadata are manually mapped by user, while, in our approach metadata are assigned using HFO tree. In our approach transformations are done through RDDs (distributed computing) to enhance processing performance and data is streamed using discretized stream instead of classic data extracting from data sources in PDI.

Figure 11 shows the processing time (in millisecond) of the same operations on input invoices done by our system and PDI. In figure, we used the same transformation flow on PDI and HFO RDDs. In Spark, we used 2 different configurations to show distributed computing performances.

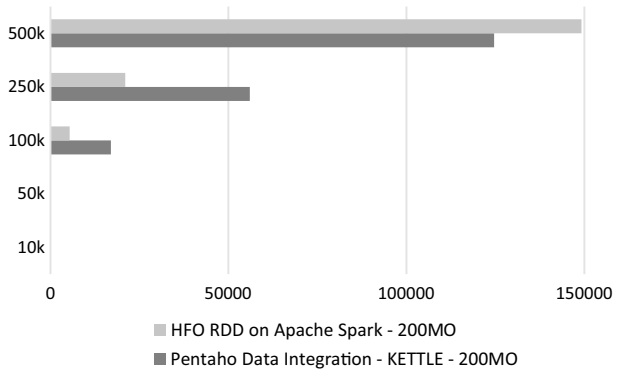

Fig. 13 Records transformation failure due to lack of memory

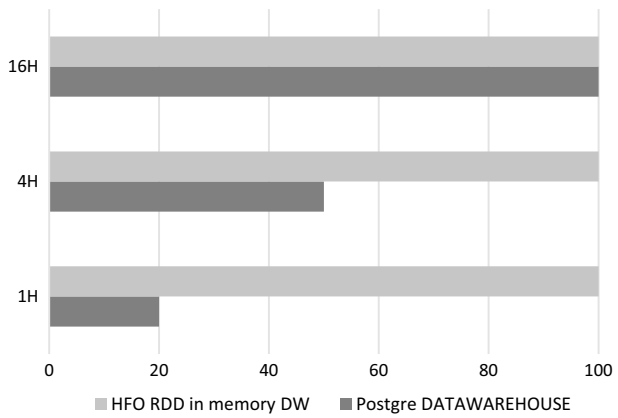

Fig. 14 Up-to-date data available in datawarehouse

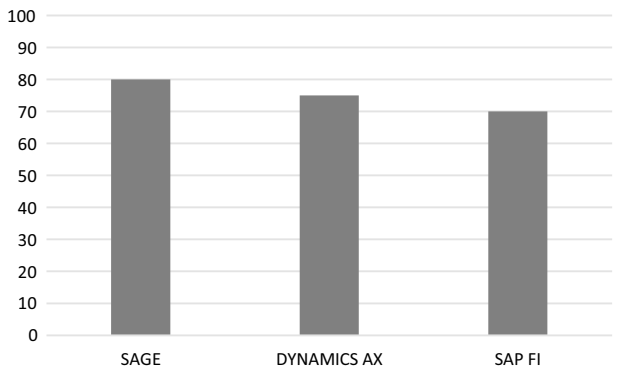

Fig. 15 Covered data of FIBO and information system metadata

Figure 12 shows the consumed memory (in kilobytes) by the same operations on input invoices done by our system and PDI.

Figure 13 shows failures in records processing due to lack of memory by the same operations on input invoices done by our system and PDI. In Apache configuration we used a memory only persistence strategy to test memory failure on large datasets.

Figure 14 shows up-to-date data percentage between a classic data warehouse and HFO RDD in memory data. The main goal of using real-time data integration pipeline using HFO RDD is to delivery up-to-date reports. Classic ETL tools are not the best solution for short time frame reports delivery especially when there is a large amount of data in processing.

Hybrid financial ontology is considered as a financial metadata knowledge base. Figure 15 shows the percentage of covered metadata between chosen local ontology which is based on Information Systems (Sage–AX–SAP FI Module) and FIBO (global ontology) as financial metadata.

Discussion

Some real-time ETL architectures are more focused on data storage strategies and their optimization to support real-time processing and analysis. We have seen this in a preprocessing framework implemented as a dynamic storage area [28]. And also in proposed near-time architecture which are mainly based on change data capture (CDC) [6, 29].This architecture offer an amalgam of concepts for data integration especially for financial data due to its ontology-based metadata management layer. For large scale data generation, this system is made for real time data acquisition and transformation then structured model of data delivery. It cover the following features:

- Business intelligence: This architecture has an added value on classic BI Tools: Real time data acquisition using Apache Kafka, rich programming interface and clustered computing of Apache Spark for transformation and finally an adaptive metadata models for financial data structures using a hybrid approach of ontologies by combining Financial industry business ontologies (FIBO) and Information system local ontologies (ISLO).
- Real-time data integration: Using Apache Kafka consumers and producers topics for data extraction at real-time.
- Virtual machines and containers: For data processing acceleration, we used a clustered architecture for distributed data transformation. Real time processing for real time delivery.
- Financial reporting: By connecting our system to a popular ERP and accounting tools (Sage–Dynamics AX–SAP FI Module), we see the hybrid financial ontology (HFO) build from available metamodel to specific metamodel of extracted data.

At the current level, our approach is not as mature as well to be integrated in a hot production environment. We need a reviews in data retrieving strategies on HFO metadata and RDDs OLAP cubes. We need also to enhance memory management, the solution has several failures on multiple node processing.

Conclusion

Using a Big data processing approach combined with a hybrid financial ontology is the best approach among existing real-time architectures. This approach is more suitable for real-time data integration, especially in large datasets processing. It shows a good performance and decrease latency in a short time frame report delivery. It also use Apache

Spark programming interface which provides a large panel of transformations that could be applied on RDDs. The hybrid financial ontology provides a rich knowledge base of metadata which assists metadata mapping and enhance key/value retrieving using our defined HFO tree. After building a big data and real-time based ETL using RDDs, we are now focused on optimizing memory consumption. Using resilient distributed datasets in Apache Spark needs more memory than other programming interfaces in data processing. And distributed computing needs a considerable number of nodes to handle task dispatching. The next works are around reducing the number of slave nodes and developing a better resources management algorithm for master node.

Abbreviations

BI: business intelligence; SQL: Structured Query Language; SSIS: SQL Server Integration Services; ETL: Extract-Transform-Load; RDD: resilient distributed dataset; FIBO: financial industry business ontology; ISLO: information system local ontology; HFO: hybrid financial ontology; OLAP: online analytical processing; OLTP: online transaction processing; ROLAP: relational online analytical processing; MOLAP: multidimensional online analytical processing; HOLAP: hybrid online analytical processing; DBMS: Database Management System; I/O: input/output; HDFS: Hadoop Distributed File System; LRU: least recently used; RDBMS: relational database management system; CSV: comma-separated values; JSON: JavaScript Object Notation; JDBC: Java Database Connectivity; DStream: discretized stream; TCP: Transmission Control Protocol; SOA: service-oriented architecture; CDC: change data capture; API: Application Programming Interface; IFRS: International Financial Reporting Standards; GAAP: Generally Accepted Accounting Principles; COA: chart of accounts; W3: World-Wide Web; PDI: Pentaho Data Integration; ERP: enterprise resources planning; SAP: systems applications and products; AX: Axapta.

Acknowledgements

Not applicable.

Authors' contributions

Mr RIDA Mohamed worked on existing accounting system schemas and UML diagrams. Mr ABGHOUR Noureddine was in charge of choosing the right Big Data tools, Mr MOUSSAID Khalid worked on FIBO and metadata of choosen ERPs Then Ms EL OMRI Amina helped on performing the programming patterns. All authors read and approved the final manuscript.

Funding

Not applicable.

Availability of data and materials

The datasets generated and analyzed during the current study are confidential and cannot have a public status.

Competing interests

The authors declare that they have no competing interests.

Received: 24 July 2019 Accepted: 18 October 2019
Published online: 11 November 2019

References

1. Chaudhuri S, Dayal U, Narasayya V. An overview of business intelligence technology. Commun ACM. 2011;54(8):88–98.
2. Chaudhuri S, Dayal U. An overview of data warehousing and OLAP technology. ACM Sigmod Rec. 1997;26(1):65–74.
3. Lessambo FI. Financial statements: analysis and reporting. London: Palgrave Macmillan; 2018.
4. 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); 2015. p. 345–50.
5. Su J, Tang Y. Business intelligence revisited. In: 2017 fifth international conference on advanced cloud and big data (CBD); 2017. p. 1–6.
6. Obalı M, Dursun B, Erdem Z, Görür AK. A real time data warehouse approach for data processing. In: 2013 21st signal processing and communications applications conference (SIU); 2013. p. 1–4.
7. Zaharia M, et al. Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: présenté à presented as part of the 9th {USENIX} symposium on networked systems design and implementation ({NSDI} 12); 2012. p. 15–28.
8. Salloum S, Dautov R, Chen X, Peng PX, Huang JZ. Big data analytics on Apache Spark. Int J Data Sci Anal. 2016;1(3):145–64.
9. Chae M, Lee H, Lee K. A performance comparison of linux containers and virtual machines using Docker and KVM. Clust Comput. 2017;22(1):1765–75.

10. Joy AM. Performance comparison between Linux containers and virtual machines. In: 2015 international conference on advances in computer engineering and applications; 2015. p. 342–6.
11. Senthil Kumaran S. Practical LXC and LXD: linux containers for virtualization and orchestration. Berkele: Apress; 2017.
12. Thomsen E. Olap solutions: building multidimensional information systems. 2nd ed. New York: Wiley; 2002.
13. Gulla JA, Brasethvik T. A hybrid approach to ontology relationship learning. In: International conference on application of natural language and information systems; 2008. p. 79–90.
14. Petrova GG, Tuzovsky AF, Aksenova NV. Application of the financial industry business ontology (FIBO) for development of a financial organization ontology. J Phys Conf Ser. 2017;803:012116.
15. Nguyen CN, Lee J, Hwang S, Kim JS. On the role of message broker middleware for many-task computing on a big-data platform. Clust Comput. 2018;22(1):2527–40.
16. Mittal M, Balas VE, Goyal LM, Kumar R. Big data processing using spark in cloud. Springer Singapore: Singapore; 2019.
17. Aufaure MA, Zimányi E. Business intelligence: second European summer school, eBISS, Brussels, Belgium, July 15–21, 2012, tutorial lectures. Berlin: Springer; 2012. p. 2013.
18. Page M, Spira LF. Special issue on "business models, financial reporting and corporate governance". J Manag Gov. 2016;20(2):209–11.
19. Reddy GS, Srinivasu R, Rao MPC, Rikkula SR. Data warehousing, data mining OLAP and OLTP technologies are essential elements to support decision-making process in industries. J Comput Sci Eng. 2010;2(9):9.
20. Luu H. Beginning apache spark 2: with resilient distributed datasets, spark SQL, structured streaming and spark machine learning library. New York: Apress; 2018.
21. Jun L, ChaoJu H, HeJin Y. Application of Web services on the real-time data warehouse technology. In: 2010 international conference on advances in energy engineering. 2010. p. 335–8.
22. Bensadon D, Praquin N. IFRS in a global world: international and critical perspectives on accounting. New York: Springer International Publishing; 2016.
23. Li Y, Li R. Research on financial statements system based on enterprise resource. In: Proceedings of 2013 4th international Asia conference on industrial engineering and management innovation (IEMI2013); 2014. p. 389–99.
24. Cherny J. Financial statements are messages that need a context to be better understood. Int J Discl Gov. 2014;11(2):161–76.
25. Real-time financial reporting: the need for speed July. https://www.google.co.ma/search?source=hp&ei=iXZpXJivL4OmaLCVoNAE&q=REAL-TIME+FINANCIAL+REPORTING%3A+THE+NEED+FOR+SPEED+July%2C&btnK=Google+Search&oq=REAL-TIME+FINANCIAL+REPORTING%3A+THE+NEED+FOR+SPEED+July%2C&gs_l=psy-ab.3…1771.1771..2150…0.0..0.601.601.5-1……0….2j1..gws-wiz…..0.yjmCb7-G2nE. Consulté le: 17-févr-2019.
26. de Heij D, Troyanovsky A, Yang C, Scharff MZ, Schouten K, Frasincar F. An ontology-enhanced hybrid approach to aspect-based sentiment analysis. In: Web information systems engineering—WISE 2017; 2017. p. 338–45.
27. An extended hybrid ontology approach to data integration—IEEE conference publication. https://ieeexplore.ieee.org/document/5304734. Consulté le: 24-févr-2019.
28. Li X, Mao Y. Real-time data ETL framework for big real-time data analysis. In: 2015 IEEE international conference on information and automation, Lijiang, China; 2015. p. 1289–94.
29. Mohammed Muddasir N, Raghuveer K. Study of methods to achieve near real time ETL. In: 2017 international conference on current trends in computer, electrical, electronics and communication (CTCEEC), Mysore, India; 2017. p. 436–41.

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Submit your manuscript to a SpringerOpen journal and benefit from:

- ► Convenient online submission
- ► Rigorous peer review
- ► Open access: articles freely available online
- ► High visibility within the field
- ► Retaining the copyright to your article

Submit your next manuscript at ► springeropen.com