Steppe Chorus

Platform for Advanced Analytics

White paper

Executive summary

It's a truism that companies delivering services need to do two things to stay in business: identify new marketing opportunities and adjust the existing service offerings to ensure the most cost-effective operation of the company. With rapidly changing customer behavior, and increasing ease of switching between service providers (churn), fast access to actionable information and insights is crucial for success of the service-based business. Advanced analytics helps companies both to come up with new offerings and monitor the performance of the existing services. Figure 1 illustrates application of advanced analytics to improve the process of selecting customers that are targets of marketing campaigns (both mass mailings and individual offers to at-risk customers).

Depiction of business process

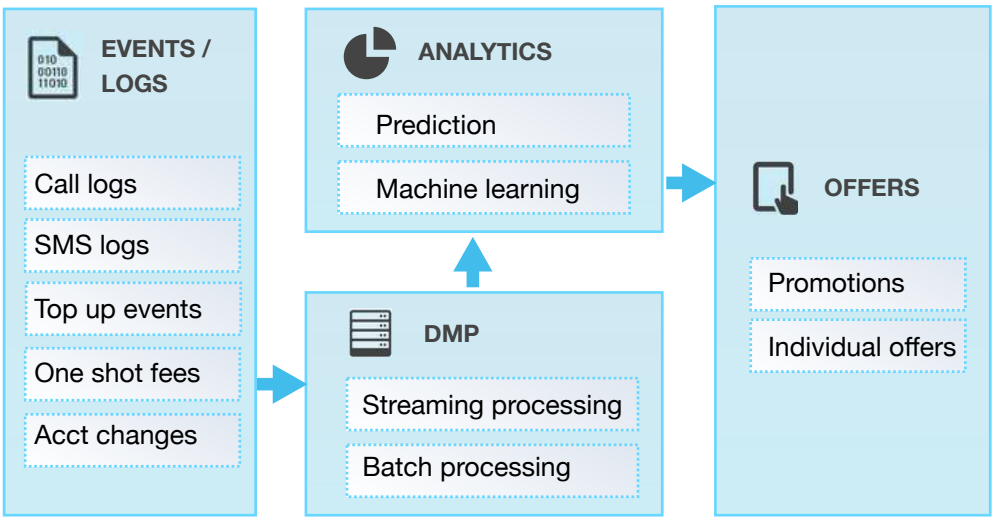

Figure 1. Depiction of business process

Table of contents

.

Introduction

Practically every company has some sort of reporting / analytics (business intelligence) solution already implemented. However, in many cases those solutions suffer from several problems:

- various departments have independent implementations of solutions (silos of information) – resulting in inconsistent and hard to combine data and information derived from them
- information is not delivered in timely manner
- hard-to-change structure of the system

The reason for silos:

- historical – early Business Intelligence solutions were based on data extracted from departmental operational systems and aggregated for use by that department.
- some analytical systems were built based on one of two Data Warehousing methodologies: Bill Inmon's Corporate Information Factory or Ralph Kimball's EDW Bus Architecture. Ideally, both solutions would have produced consistent data and information. In reality, different components were developed asynchronously and with less than perfect cooperation between entities, resulting in inconsistent data stores.

The reason for high latency of information:

- practically any analytical system developed prior to 2011 (release of Storm) was based on batch processing. Processing requiring multiple steps (with each step taking hours in order to process a daily batch, for example) takes hours or days.
- updating large volumes of data stored in relational DBMS is a costly process, preferred to be done in times of low demand (at night).

The reason for hard to change structures:

A lot of analytics solutions are based on relational DBMS. RDBMS systems do not, in general, handle nested and repeated data structures. In order for changes in structure of input data to be used in the processing, the entire chain of ETL (Extract, Transform, Load) processes, as well as all subsequent processing within the Data Warehouse needs to be modified.

Problem definition

Problem 1 – In the past users were willing to wait for hours or even days for records of events to appear in the reporting data repositories. Now users are not willing to wait more than few minutes for records of events to become visible in Data Repository, yet legacy systems are not able to cope with this requirement.

Problem 2 – repositories built for high access speed and based on relational technologies often lock entire tables for the duration of **DELETE** and **UPDATE** operations. Data from those tables cannot be queried during such operations. Therefore, users have to wait for those operations to complete before they can receive results of queries.

Problem 3 – legacy systems are able to return quickly results of pre-aggregated data or data that are otherwise indexed. However, indexing of relational systems is expensive (slow), thus slowing down ingestion of low-latency data.

Problem 4 – legacy systems were designed in the days when computing resources (processing, memory and storage) were very expensive, so the systems were designed to work smarter, not harder (using indexing, for example). However, maintenance of indices on large number of fields and large number of records is very expensive. With the drastic reduction in the cost of computing resources, brute force makes it economical to unearth facts stored in non-indexed fields by quickly scanning large volumes of data.

High-level description of the system

DMP consists of 3 major components: ingestion, computing, and output. Figure 2 depicts those components:

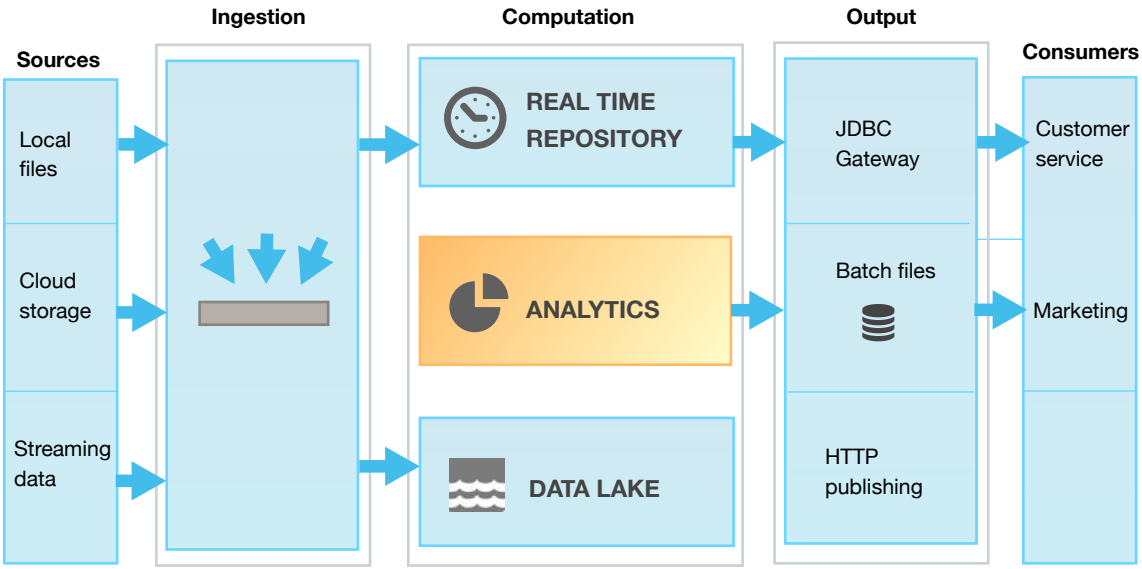

Figure 2. High-level view of the system

Ingestion component

The ingestion component consists of several adapters to retrieve data from various types of sources and a data distribution hub (Apache kafka cluster). The adapters are capable of reading data from the following source types: files from local file systems, files from Cloud Storage, streaming data from streaming feeds. The adapters send data to the distribution hub (kafka).

Computational component

At high level, implementation of the computational component is similar to Lambda Architecture described in Nathan Marz' book 'Big Data'.

Two clusters comprise the component:

- a standard Apache Spark™ cluster that enables large-scale processing of data and pre-computation of required aggregations (batch layer), and
- a low-latency cluster that captures data seconds after they have been made available for input (speed layer). The speed layer is implemented as Apache Solr™ cluster.

The same Solr cluster is used to serve queries that require response in near-real-time. Such queries combine pre-calculated aggregates with the newly-arrived data to return up-to the minute results in few seconds.

The full power of Spark on the main Spark cluster enables users to perform advanced analytical calculations such as Machine Learning, pre-calculation of aggregates, etc. Such calculations are not sensitive to recently arrived data.

Output component

The output component consists of a query gateway for querying data, as well as various adapters for pushing data to desired destinations in various formats: files, streaming data (such as HTTP requests, etc.)

Solution details

Exact products used by DMP and the flows of data and control between them are shown in Figure 3 below.

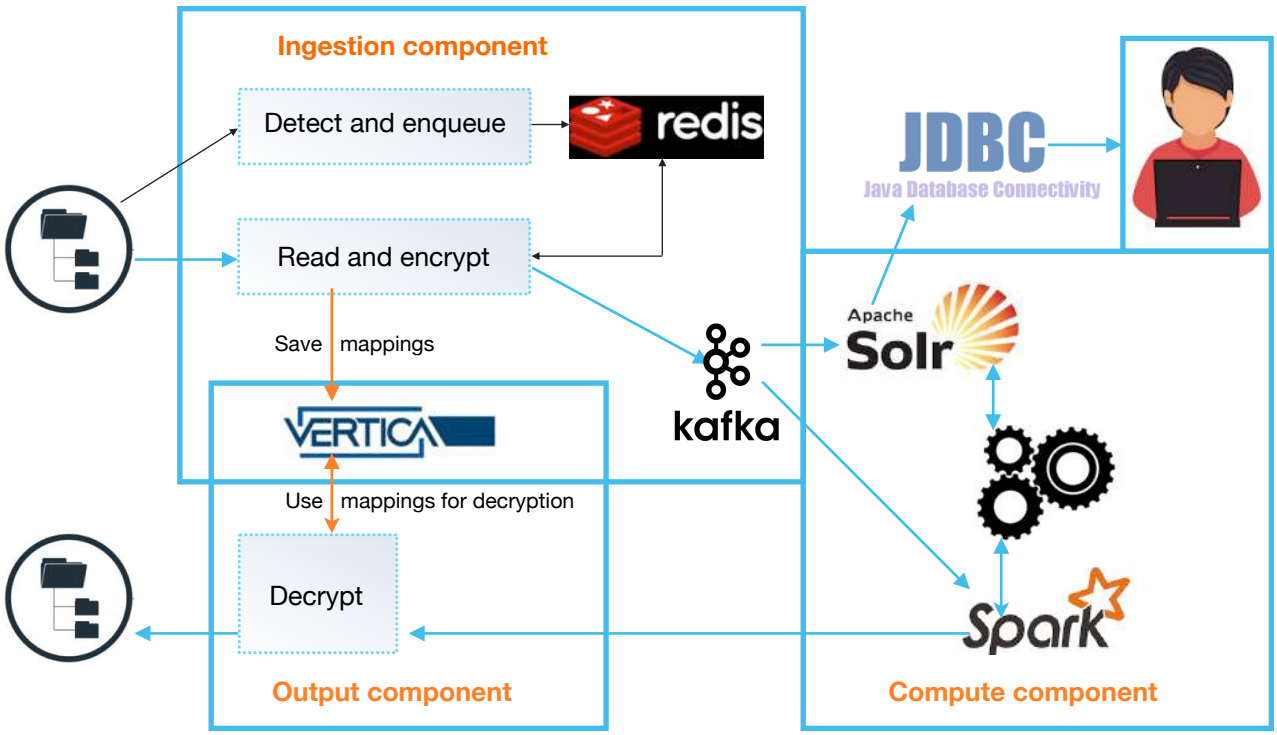

Figure 3. Data Flow diagram

Ingestion component

Ingestion component consists of three types of sub-components:

- adapters for each type of input (local files, streaming inputs, etc.)
- pre-processing modules that perform tasks required prior to sending data to computation component (e.g. obfuscation of fields containing Personally Identifiable Information (PII))
- a scalable and fault-tolerant Apache kafka cluster.

The role of each type of adapter is to ensure all data made available to the DMP are:

- queued for processing
- pre-processed (if required)
- sent to the kafka cluster

The current version of DMP contains a single type of adapter – one used to read local files. However, the architecture allows for addition of other types of adapters.

Queueing of batches (local files) for processing is done by a dedicated process. The process performs the following actions**:**

- scans a set of directories known to contain input data;
- detects files in the directories that have been added since the previous scan;
- adds new files to the list of files to be processed (the names of new files are added to MySQL DB, as well as to Redis servers).

Redis servers are used as queue managers for batches of data to be processed. Redis is an in-memory database with built-in list-processing primitives (push and pop). These primitives make queue management easy. In addition, as an in-memory DB, Redis is capable of handling much higher volume of queries than MySQL. After restart, Redis database are initialized based on the combination of content of input directories and data stored in MySQL.

Pre-processing step is optional. Two types of pre-processing are implemented within the ingestion component: obfuscation of fields containing PII data and 'enrichment' of logs with data available from other sources (such as CRM data).

Obfuscation of fields containing PII prior to sending data to Computation component ensures that users working with data within the component cannot attribute any records to (or search for records of) any individual.

The obfuscation is based on SHA-1 hashing. The mappings between obfuscated and clear text values are kept in a separate database (Vertica).

These mappings are used when clear text values are required in outputs delivered to Customer Support or Marketing departments.

'Enrichment' of logs with data available from other sources (such as CRM data) enables faster delivery of output results by eliminating the need to perform joins during or at the end of the computing phase. The logs contain numerical IDs of textual values available in dictionary tables (those tables are stored in a CRM DB). During the ingestion of every record new fields are added to the record. The new fields contain text values associated with the numerical IDs.

In order to perform enrichment, contents of the dictionary tables are cached in memory of every ETL server that processes logs. The caches are kept up-to-date as the values in underlying dictionary tables change.

Sending of data to kafka is done via standard kafka publisher API. Once the data have been accepted by a kafka broker, the input data are marked as processed.

In the case of input data arriving as files, the files are moved into a 'processed' directory. Data received from streaming sources are discarded.

Computation Component

Computation component consists of two subcomponents:

- Spark cluster running on top of YARN and using HDFS for storage and
- Cloudera Search (Solr) cluster.

Spark cluster is used for long-term storage of data and a variety of computational tasks (calculation of various aggregates, calculation of features used for execution of models, data exploration, feature engineering, development of models for prediction of user behavior, etc.).

Solr cluster is used for serving interactive queries requiring fast response time, as well as for ensuring that recently arrived data are included in the results of those queries.

Processes running within Spark cluster

Two types of processing are performed on Spark cluster:

- R&D (design of behavioral models and associated features) and
- Production (execution of the models developed during R&D processing).

Research and Development activities can be classified as follows:

- Data discovery
- Feature engineering
- Model development

Production processes are broken up into two buckets:

- Calculation of features
- Execution of models

The outputs of production processes are delivered to two possible destinations:

- Solr cluster
- Files that can be collected for further processing

Features delivered for near-real-time consumption are copied to Solr cluster. When end-users request values of such features, the returned results will include up-to-the minute data. Such inclusion is achieved by combining pre-calculated values with the latest data that are pushed into the Real-Time Repository as part of Ingestion of processing (*see Figure 3*). Batch outputs (files) are sent via Egress processing to file servers accessible to end-users. Egress processing is described in Output component section.

Output component

Output component exposes to the outside world the results of work performed by Computation component.

The results are made available in three different ways:

- Via JDBC gateway – for consumers using SQL to access data on demand
- Via files – for consumers that need bulk data (such as for bulk marketing campaigns)
- Via HTTP streaming – for real-time campaigns

Access via JDBC gateway

Results of the work residing within the Computation component can be accessed on-demand by submitting requests to a Thrift JDBC server. The server accepts requests in Hive Query Language (HQL) – a subset of SQL.

Data stored in Spark cluster within HDFS are accessible via standard Hive access methods.

Data in Solr collections are accessed via a custom SerDe (Serializer/DeSerializer) Hive adapter.

Access via local files

Results that are meant to be consumed in their entirety are made available as local files. Prior to making files available for consumption, data in fields containing PII are converted from the obfuscated (hashed) values back into clear text. The task is performed by Egress process. The process is an inverse of Ingest process. The files to be processed are queued for processing, then processed. Conversion of obfuscated data into clear text format is accomplished by querying mapping tables generated during the Ingestion of data.

Since data in output files might contain PII, care is exercised to restrict access to the files.

Access via HTTP streaming

In some cases, the output needs to be delivered in seconds, rather than waiting for the next scheduled execution of batch jobs. An example is an advice to offer a user a different package in order to save on

overage charges. For such cases Output component enables push of results to HTTP servers via various interfaces (SOAP or REST).

Summary

DMP takes care of time-consuming tasks (data imports, exports, on-demand access, dealing with PII, etc.) in a robust and scalable manner. In addition, DMP enables customers to perform large volumes of computations in scalable environments.

DMP frees customers to focus on business-related issues, rather than dealing with technical problems.

Get in touch:

Leo Gelman
Head of Data Engineering
leo.gelman@steppechange.com

Ina Goldberg
Head of Analytics Solutions
ina.goldberg@steppechange.com