Christopher Botts
DSC650
Bellevue University
Final Project

Objective

Leverage big data technology to analyze ecommerce transaction data and output a list of recommended add-on items for customers.

Requirements

The technology utilized must be capable of processing a large quantity of transactions.

Queries must be completed in real time, so that the list of recommended add-on items can be presented to consumers before the transaction is complete.

Transaction history should be used to generate a “frequently purchased with” list that improves conversation rate when compared with a randomly populated list of items.

Strategy

Create a NiFi flow within which data can be modified and analyzed.

Ingest transactions into NiFi via ConsumerKafka processor.

Modify transaction data as necessary to complete Solr queries.

Store data information in HDFS and log failures.

Publish query results to Kafka.

Methods

This system uses Apache NiFi, Solr, HDFS, and Kafka.
For proof of concept, a dataset of Ecommerce transactions was ingested into NiFi.

The image below shows the flow built in NiFi (Image 1).

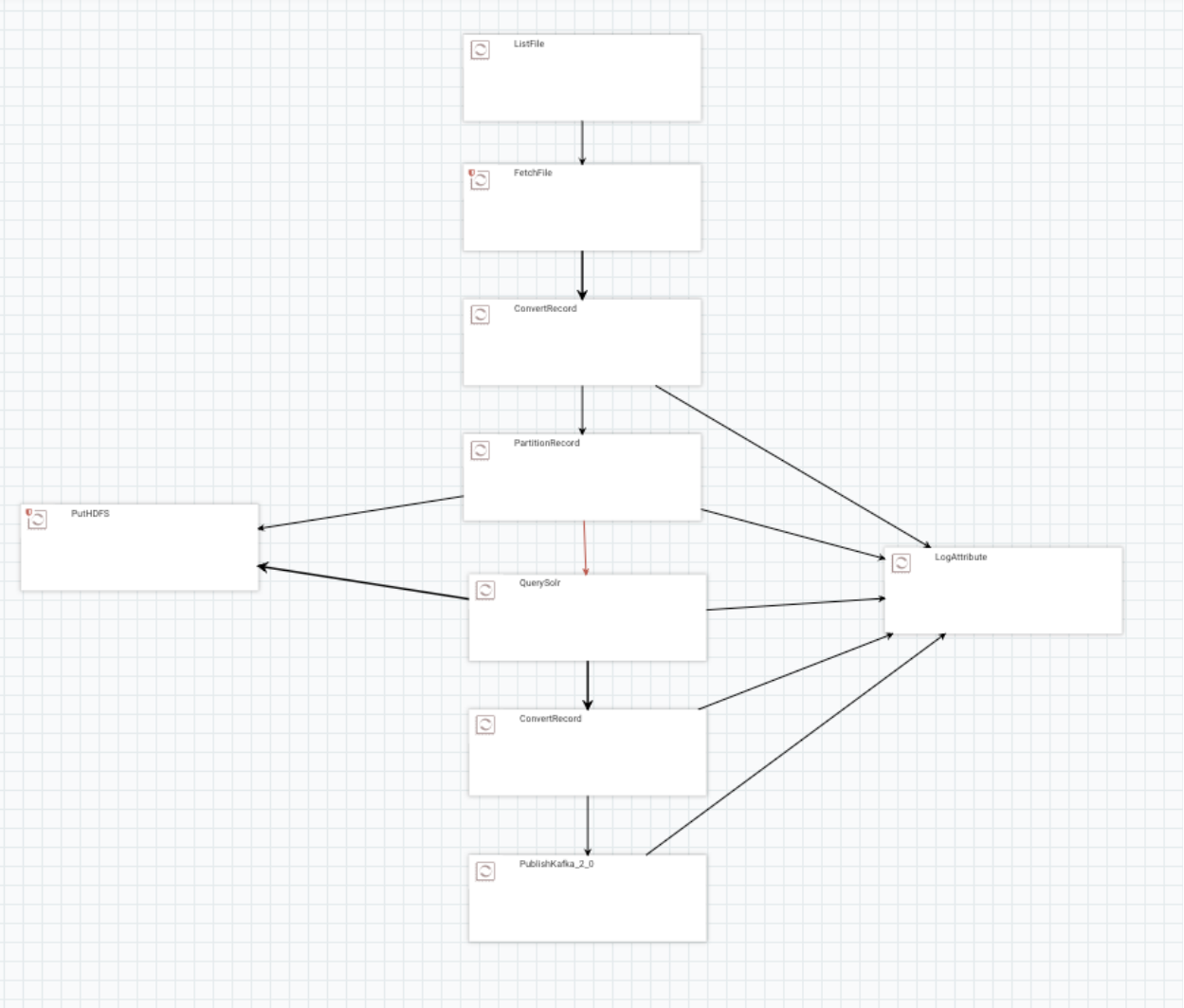

Image 1. NiFi flow used for this project

The processors used for this project will be explained from top to bottom.

Data Ingestion
A dataset of Ecommerce transactions was ingested for use in the project via the top two processors, ListFile(1) and a FetchFile(2) which found and accessed the csv document that was saved on the virtual machine.

Data Transformation
The ConvertRecord (3) processor utilized a CSV Reader and a JSON record writer to convert the dataset from its original format to JSON for use downstream. At this point, the data is stored within one record, with each line-item sale represented a row in the record.

The PartitionRecord (4) processor grouped line-item sales by transaction number. This aggregation enabled a reindexing where transaction number served as an index for later queries. This processor then split the dataset along the transaction number index, so that transactions were sent downstream as separate records.

The original record was sent to a PutHDFS (9) processor (Image 2). HDFS was chosen because it is fault tolerant, scalable, and cost effective as a means of storage.

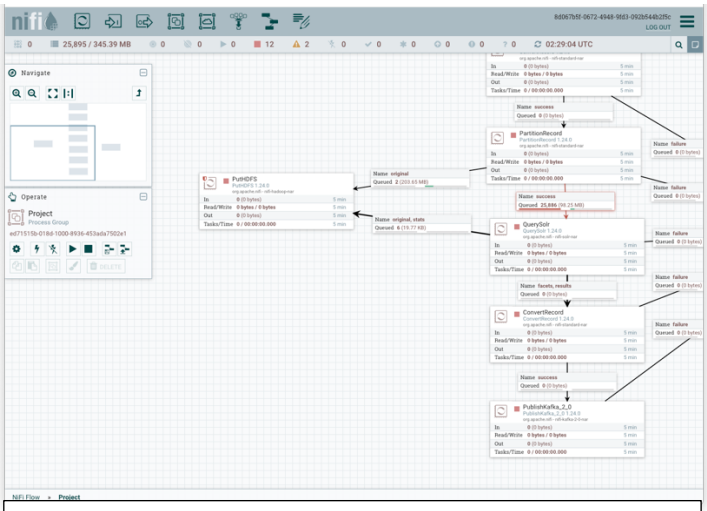

Image 2. Processed records are stored in HDFS

Data Analysis

Transaction records were then sent to Solr with the QuerySolr (5) processor. A simple query utilizing faceting was run on the transaction data Solr was chosen for this because its fast and advanced querying capabilities enable us to analyze a large amount of data in real time. Solr can filter transactions by the items in Customer X's cart and then use faceting to generate a count of the number of items other customers have purchased alongside the items Customer X's cart. The top five items most frequently purchased with the items in Customer X's cart should be returned in descending order.

The original records and stats were sent to the PutHDFS (9) processor for fault tolerant storage (Image 2).

The images below show faceting from the Solr user interface(Image 3) and the results of the QuerySolr processing in NiFi (Image 4). "Facet" displays the item code for individual items sold by the retailer in our example dataset.

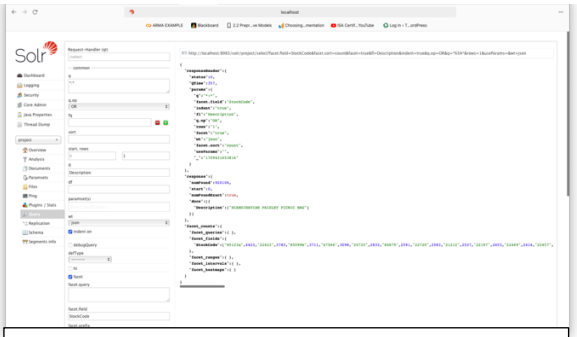

Image 3. Query in Solr UI

Image 4.SolrQuery output in NiFi

Record Publishing

Faceting results were sent to another ConvertRecord (6) processor to write the JSON records in an AVRO format for use with Apache Kafka.

The PublishKafka_2_0 (7) processor published the query results to a Kafka topic. The image below shows the results of the query from a Kafka consumer terminal (Image 5).

Image 5. PublishKafka results viewed from a kafka consumer

Displayed in the Kafka terminal are the item codes of frequently purchased items, shown in descending order by number of units sold. In application, the online store should use these codes to present the customer with the list of frequently purchased items that correspond with the item code.

Failures

Processing failures from all processors were logged with a LogAttribute (8) processor (Image 6).

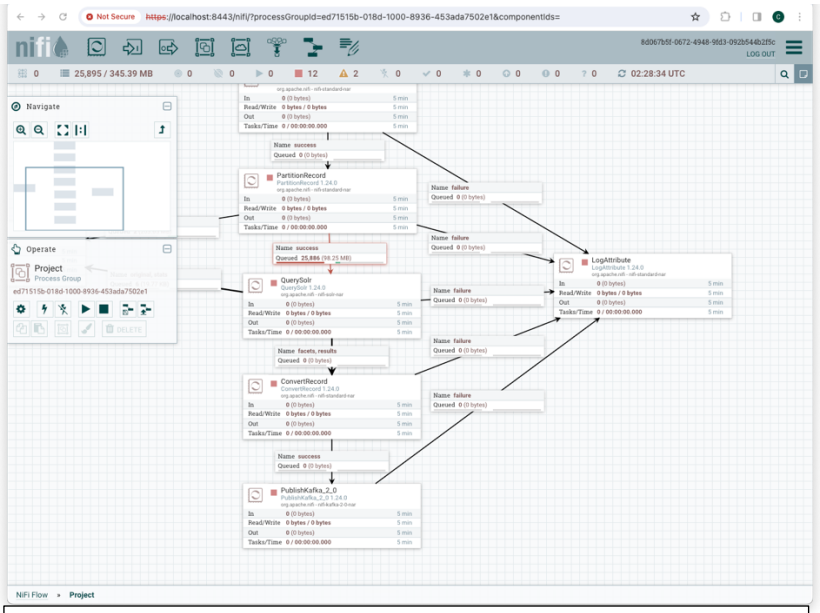

Image 6. LogAttribute process was used for logging failures

Modifications necessary for real time use

A dataset was downloaded for proof of concept and retrieved from the cloud system using the ListFile and FetchFile Nifi processors. However, for use in a real time production system, these two processors would be unnecessary. Instead, I would recommend data collection by Kafka and ingestion into NiFi via the ConsumeKafka processor. Kafka is ideal for this data collection because it can stream from multiple data from multiple sources, so it can capture transaction data from POS systems in various brick and mortar stores as well as online transaction data.

Likewise, the PartitionRecord processor was necessary because of the method of data entry into the downloaded dataset used for proof of concept. This processor would likely be unnecessary in a real time system because transactions would stream separately.

Below is an image of the proposed flow for a real-time system (Image 3).

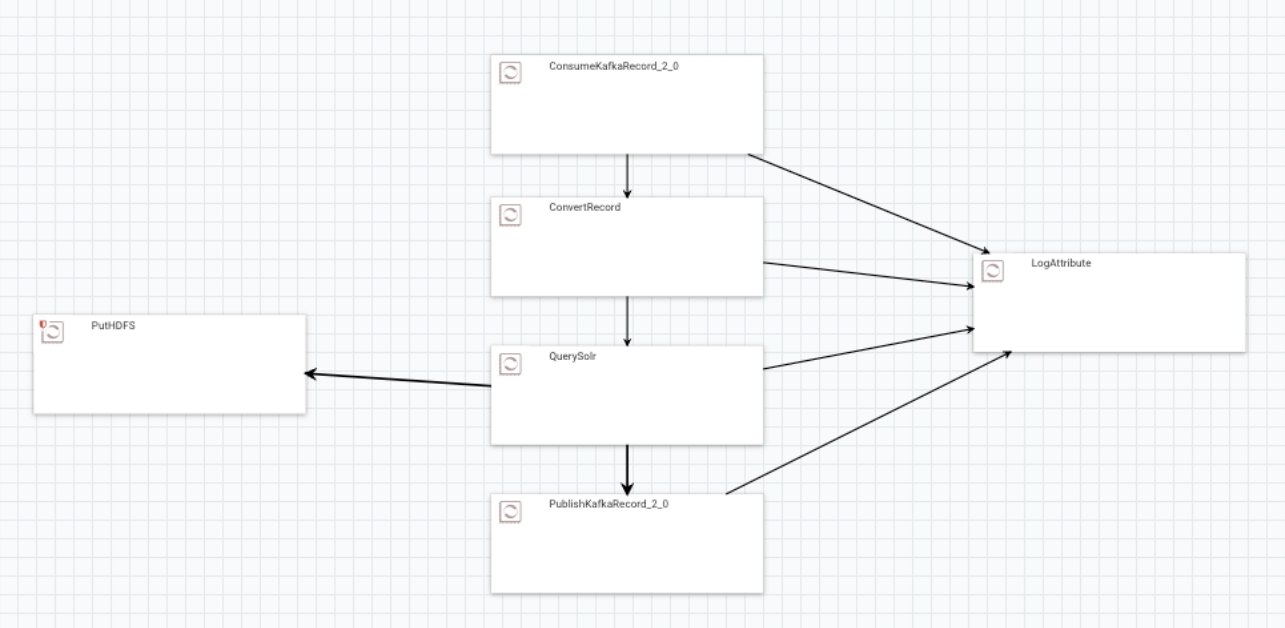

Image 7. Proposed NiFi flow for use with real-time transaction streaming.

Improvements for the future

After completing this project, I would suggest three improvements.

1. The results of this project would be stronger with a more effective use of Solr querying. The query used for this project was rudimentary and could be improved with a better understanding of Solr syntax.
2. The formatting of the results could be improved. The results seen in this project contain unnecessary information. Ideally, a clean list of five item codes would be published to the Kafka topic, which would be easier for an online store to access. The addition of a JoltTransformJSON processor might function for this purpose.

3. To further improve the conversion rate, Apache Spark could be integrated into the flow. Spark uses MLlib to apply machine learning algorithms that could predict which other products a consumer is likely to purchase.

Resources

Carrie1 (2017). E-Commerce Data. Retrieved from: https://www.kaggle.com/datasets/carrie1/ecommerce-data