CareFirst/SBP: Real-time monitoring of claims inventory

….with a particular focus on Cloudera and MicroStrategy implementation best practices

Narendra Kambam
Director, Healthcare Analytics

Fajun (George) Shi
Business Intelligence Architect
Healthcare providers transmit millions of claims to payers every day. While most claims are adjudicated, up to one in five claims are rejected or denied, representing unpaid services — and lost or delayed revenue to the division. Understanding the underlying reasons for these and resolving them takes significant time and resources; therefore, it’s critical for businesses to identify causes, monitor the inventory of all open claims (pended, denied, rejected, and other status) in near real-time, and prevent them. Unfortunately, most payer organizations lack the ability to extract the true value of this information.

This presentation provides an overview of how we solved this problem using MicroStrategy by implementing cutting-edge big data streaming technologies and providing near-time operational insights to the Claims Management team.

The session will also go behind the scenes of building an end-to-end solution using Flume, Spark Streaming, Apache Kudu, and a live Impala connection using MicroStrategy.
Federal Employee Health Benefits Program

Near real-time requirements and Architecture

Integration between MicroStrategy and Cloudera

Lessons Learned

Note: Content is relevant at the time of material preparation
What is FEHBP?

- The Federal Employees Health Benefits Program (FEHBP) provides group coverage to employees, retirees, and their eligible family members
- Employer-offered benefit
- Choice of 230 health plans
- Individual choice market for about 8M enrollees and dependents (average contract = two members)
- Federal government pays 72% of average premium of FEP Plans towards total cost of each member’s premium; no more than 75% of total premium of any FEHBP Plan
- The FEHBP offers Fee-for-Service plans, plans offering a Point-of-Service product, Health Management Organizations (HMOs), High-Deductible Health Plans, and Consumer-Driven Health Plans
- is administered by the Office of Personnel Management (OPM)
Known as FEP; formally called Government-wide Service Benefit Plan

One of the first six original carriers; named in federal legislation that initiated FEHBP

Originally two separate associations; BCBSA formed to provide stronger solutions to market and manage the license and the brand

Thirty five independent companies with a unified purpose: to be the healthcare provider of choice

All BCBS Plans participate in FEP – size of business varies
Also known as Federal Employee Program Operations Center (FEPOC)

We manage enrollment processing, claims processing, and online e-solutions for FEP Plans and members

We develop innovative solutions to support member management efficiencies and customer satisfaction
Federal Employee Health Benefits Program

Near real-time requirements and Architecture

Integration between MicroStrategy and Cloudera

Lessons Learned
Near real-time (NRT) requirements to solution

- BCBSA Plans need NRT open claims inventory to manage daily operations
- Secure data to all 35 BCBS Plans
- Capture messages in NRT
- Scale to millions of transactions and should be inserted, updated, and deleted
- Exactly once processing (no duplicate records)
- NRT access to data, high throughput scans and a live connection since the data changes rapidly

Cloudera and MicroStrategy platform
Kudu + Sentry
IBM Qrep + Flume + Spark Streaming + Kudu
43 million messages per day – per thread per partition
One partition per feed
MicroStrategy + Impala + Kudu
Kudu and Impala integration meets the requirements

- DDL: Create/Alter/Drop Table
- DML: Data Insert/update/delete
- Data can be inserted into Kudu tables in Impala
- Impala supports UPDATE and DELETE SQL Commands to modify existing data in Kudu tables
- Flexible Partitioning (Hash, Range, Multi-level)
- Kudu allows dynamically pre-split tables by Hash and/or Range
- Supports parallel scans
- High-efficiency queries
- Impala push down predicate evaluation to Kudu
NRT high level architecture

Mainframe DB2

Transaction logs

QPublish program

WebSphere MQ

QPublish XML Parser

Apache Flume Agent

JMS Source

Kafka Channel

Apache Kafka

Kafka NRT topic

Kafka NRT error topic

1 partition

subscribes to the NRT topic

Spark Streaming

Kudu

Kudu API

UPSERT and DELETE statements

Config properties

MicroStrategy

Impala

HDFS

MicroStrategy
NRT ingestion metrics

“JMS source batch size” of 300

Processes ~500 QPublish XML messages per second
NRT dashboard running on Kudu with key metrics
Federal Employee Health Benefits Program

Near real-time requirements and Architecture

Integration between MicroStrategy and Cloudera

Lessons Learned
Environment Configuration

- MicroStrategy
  - GA 10.4, Hotfix 3
- Cloudera
  - CDH 5.11.0
  - Impala 2.8.0
  - Kudu 1.3.0
  - Impala ODBC Connector 2.5.37
- Dell Hardware
- Linux Operating System
Functions - Impala supports MicroStrategy functions

- Impala 2.x supports most of MicroStrategy functions
  - Basic Functions
  - Date Functions
  - String Functions
  - Math Functions
  - OLAP Functions
  - Statistics Functions

- Why it is important?
  - If Impala doesn’t support these functions, we have to use pass through functions in MicroStrategy. These functions will render complex transformations and manual work.
Data Types - Impala supports all major MicroStrategy data types

- Impala supports the following data types
  - String data type
  - Numeric data Type
  - Date, Time, and Timestamp
- Int64 is an improvement in MicroStrategy and is the same size as BIGINT
- Why it is important?
  - If you use multi-pass SQL, and temporary tables are created, MicroStrategy I-Server will look at the datatype mapping between MicroStrategy and Impala. If it doesn't match, the reports will fail.

### Impala and MicroStrategy datatype mapping

<table>
<thead>
<tr>
<th>Impala Datatype</th>
<th>MicroStrategy Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHARACTER</td>
<td>Char</td>
</tr>
<tr>
<td>NUMERIC</td>
<td>Numeric</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>Decimal</td>
</tr>
<tr>
<td>INTEGER</td>
<td>Integer</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>Integer</td>
</tr>
<tr>
<td>FLOAT</td>
<td>Float</td>
</tr>
<tr>
<td>REAL</td>
<td>Float</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>Double</td>
</tr>
<tr>
<td>DATE</td>
<td>Date</td>
</tr>
<tr>
<td>TIME</td>
<td>Time</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>TimeStamp</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>VarChar</td>
</tr>
<tr>
<td>BIGINT</td>
<td>Int64</td>
</tr>
<tr>
<td>TINYINT</td>
<td>Integer</td>
</tr>
</tbody>
</table>
Kudo Data Types – Best practices

- Kudu stores date, time, timestamp as BIGINT. This is a feature of Kudu. To accommodate this feature, we need to define the object definition using the following functions to convert BIGINT to timestamp or date.
  - Use `FROM_UNIXTIME(BIGINT, FORMAT_STRING)` to convert to timestamp values in string
  - Use `UNIX_TIMESTAMP(STRING)` to convert to date or timestamp
- Note: We requested MicroStrategy to enrich the functionality of the `TO_STRING` and `TO_DATETIME` to minimize the manual conversion. These two are currently available only on the analytic engine.
Schema design – Adaptation of denormalized schema on HDFS platform

- Denormalizing the data structures on HDFS optimizes I/O performance by writing to contiguous sections of disk drives
- The goal is to minimize disk seeks joins, when feasible
- Consider trade-offs and find a “Goldilocks” zone on the amount of denormalization on Big Data platform

- Why it is important?
  - When data structures on HDFS are denormalized, it impacts the MicroStrategy platform because MicroStrategy prefers normalized structures with lookup tables
  - To achieve the best performance from a single table that contains all the relationships and all the attribute forms, avoid the self join in the last pass
Performance Tuning: Element browsing strategies

- Challenges for Denormalized Schema
  - No looking up tables
  - Large number of elements
  - Element browsing using big tables can cause slow response

- Element cache strategy
  - Enable object and element caching
  - Increase the object and element caching to 1GB
  - Create Visual Insights dashboard and put the attributes on the filter
  - Make the style as checkbox or radio button
  - Schedule to run daily

Improve element browsing performance via Visual Insights
Object creation – Attribute Character Datatype

- MicroStrategy converts Impala Character type to String
  - Fixed length character data type is mapped as CHAR(32747) when imported into MicroStrategy
  - Variable character data type is mapped as VARCHAR(334) when imported into MicroStrategy

- Why it is important?
  - It is important because it takes more memory on I-server for the cube and also for the view evaluation. This is a known discrepancy and the way to address this is to modify the DT mapping.pds when doing the next upgrade

- For all char datatypes, specifically trim the spaces within MicroStrategy. Note that you are not dealing with relational databases and everything is stored as a file on HDFS
Object creation - Metric Datatype

- Count Metric column alias data type
  - Must use Big Integer

- Why it is important?
  - The default is Integer. The SQL will fail if the Big Integer is not chosen on Big Data platforms
Intermediate Table Type default option is Permanent table. The Permanent table option in HIVE produces high latency of creating tables and slows performance of inserting records.

Change that to Derived Table or Common Table Expression to achieve high performance.

Why it is important?
- This is to support the non-transactional database, such as HIVE.
Multi-Pass SQL Support- HIVE Permanent table options

- When derived table is not a valid option, fallback to Permanent Table
  - Stored Data as Parquet (Create Post String) for performance gain
  - Collect Stats on Intermediate Tables (Pre/Post Statements)
  - New Drop table syntax
    - DROP Table TAB_NAME PURGE
    - New Patch from MicroStrategy needed

- Challenges
  - Performance overhead of insert records into Permanent tables
  - No option to create partition on Intermediate Table
  - No VLDB Settings to create Primary Index or Secondary Index
  - Kudu Table might be better options
    - MicroStrategy does not support Kudu Create Syntax
There is a limitation in Impala when it comes to supporting group by complex expression.

To overcome in MicroStrategy, when there is complex calculation in Attribute ID form, select the GROUP BY ALIAS choice.

Why it is important?
- When attribute is built using complex expression, the report will fail. To avoid that, we need to change the GROUP BY ALIAS setting.

VLDB Setting: Group by ID attribute
Impala Count Distinct Limitation – VLDB Customization

- The current limitation of Impala is every SQL pass can only have one count distinct expression
- The current default option is Level 4 and it tries to combine the separate SQL passes into one
- When you have more than two count distinct expressions in Impala, choose level 2 to force MicroStrategy to choose different SQL passes for count distinct metrics
- If data sets have only one count distinct metric, increase SQL Optimization level to Level 4
- Why it is important?
  - It is going to fail if there are more than two count distinct expressions

Set project level SQL Global Optimization default to level 2

![SQL Global Optimization](image)
Performance tuning – Cloudera side

- Multi-Level Partition
  - Cloudera Supports Multi-Level Partition
  - Partition Table by Popular Attributes
  - Choose partitioning granularity based on actual data volume
  - Find Optimal File Size (256 MB to 2 GB) for Partition columns
- Make partition columns as part of Join clause will improve performance
- Compute Warehouse Table Statistics
- Create Summary Tables
- Use the Kudu Java API for best throughput and error handling.
- Change the number of tablets per table. Recommendation is 10 or more tablets per host for large tables.
Federal Employee Health Benefits Program

Near real-time requirements and Architecture

Integration between MicroStrategy and Cloudera

Lessons Learned
Lessons Learned

- Pick a problem that addresses a recognized business need
- Doable technically, financially, organizationally, politically – not too complicated or difficult
- Pilot in, rather than big bang
- Build the necessary infrastructure
- Skillset around Cloudera and MicroStrategy integration
- Bring Cloudera and MicroStrategy Account teams and R&D together to improve the integration touchpoints
- Lot of Trial and Error – Learn and Change
- Changed our migration path to hit performance early in the development lifecycle
- Pad time for performance optimization in your project plan
Acknowledgement

- Thanks to MicroStrategy Consulting Services
- Thanks extended to Cloudera Consulting Services