# A State-of-the-Art Business Intelligence Platform

## For Ad Performance Analysis and Optimization—With Predictive Capabilities

By Fabien Coppens, Director of Egineering, PDG Consulting

PDG

At PDG Consulting, we work with a wide variety of clients, including the major players in the entertainment industry. They trust our technology and domain expertise, and frequently bring us in to help them with their toughest, most innovative, high value endeavors.

Recently, our client, a leading TV distributor for households, hired us to build a critical, ambitious Business Intelligence (BI) platform that would provide their advertising and operations teams with dynamic, actionable dashboards of ad performance data and insights. The new system would also have to be able to predict the performance of future advertising campaigns. This meant we had to solve quite a few complex, interesting problems. Now that we've tackled these challenges, with the new platform already in production and bringing value every day to our users, we'd like to share the approach and solutions we came up with to build this state-of-the-art BI tool.

## Context

Our client is a leading distributor of television channels to millions of households. Their customers can access the content through two different pipelines: 1- Via satellite: a parabolic antenna (“dish”) receives the live data and sends it to a device known as a set-top box (STB) which is connected to the television; 2- Via an app, which displays streaming data received over the internet (this is known as over-the-top, or OTT).

For both of these data pipelines, our client is also responsible for the dynamic insertion of targeted advertisements. Their systems perform the ad insertion in the available inventory of breaks for each channel. The choice and prioritization of ads shown to each customer are based on the just-in-time execution of specific business rules, which run in the firmware of each STB, in the case of satellite-based delivery.

The diagram below gives a high-level view of the systems involved in the satellite-based delivery of targeted ads to customers:

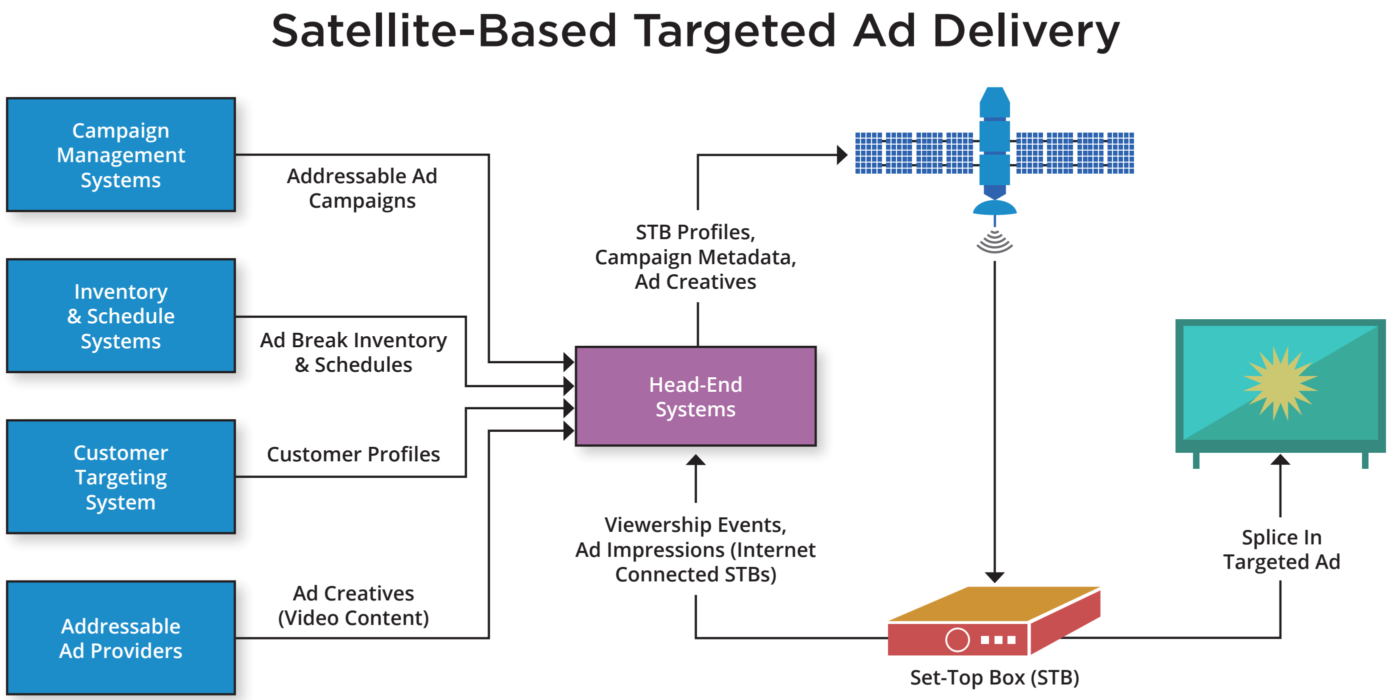
Satellite-Based Targeted Ad Delivery
Campaign Management Systems
Addressable Ad Campaigns
Inventory & Schedule Systems
Ad Break Inventory & Schedules
Customer Targeting System
Customer Profiles
Addressable Ad Providers
Ad Creatives (Video Content)
STB Profiles, Campaign Metadata, Ad Creatives
Head-End Systems
Viewership Events, Ad Impressions (Internet Connected STBs)
Set-Top Box (STB)
Splice In Targeted Ad

*Figure 1: Satellite-based targeted ad delivery*

In Figure 1, we see that the satellite pushes frequently updated sets of data to the customer’s set-top box:

- Ad campaign configurations
- Inventory of upcoming breaks per channel
- Customer profiles
- Ad video content

The set-top box is responsible for executing the business rules that combine this data to determine the most pertinent ads to display to each customer, for a given upcoming scheduled break, and dynamically inserts these ads on top of the underlying ads on the channel.

The vast majority of set-top boxes are connected to the internet and frequently send back events pertaining to:

- Viewership: channel, and start and end time of a viewership event (this includes live view events, as well as playback of recorded content)
- Ad impressions: an impression event is a single viewing of a given ad

## Objectives

Our project, to which we will refer as the AdsIntelligence project, has ambitious goals and deliverables:

- Build a centralized platform that will become the one-stop shop for all ad related data
- Provide users with dynamic configurable dashboards that allow them to slice-and-dice across very large volumes of historical data
- Compute aggregated metrics, for past, active, and future advertising campaigns
- Predict the performance of upcoming ad campaigns
- Alert users as to which active or future ad campaigns will under- or over-perform, so that they can adjust campaign configurations in real time

By leveraging these capabilities, the users can easily get answers to the following questions (and more):

- Can I understand the behavior of my campaigns by easily slicing-and-dicing across facets of all my historical data points?
- How many impressions can we sell across different sales categories?
- How many impressions do I expect a future campaign to deliver based on previous similar campaigns?
- Why is an ad campaign under-/over- delivering?
- Will an ad campaign reach its impression target?
- Are we maximizing the use of the available advertising inventory?

In order to offer these high-value features, our platform needs to leverage a set of robust ETL pipelines that will feed a centralized datalake with data from a wide set of sources:

- Customer viewership events
- Ad impressions
- Customer profiles, including persistent and campaign-related (ephemeral) attributes
- Ad campaigns
- Ad break inventory
- Channel inventory, including those that support targeted ad insertion (known as Addressable)
- Program schedules

## Volume of data

Here is a quick summary of the volumes of data that we store and analyze:

- Customer profiles: 15M
- Ad campaigns: 70K total, 200 active
- Viewership events: 150M per day, 2 GB per day
- Impression events: 200M per day, 10 GB per day
- Ad break inventory: 10K per week

## What we needed to solve

To provide our client with the powerful and ambitious platform they had hired us to build, we had to solve the following challenges:

- Optimize the storage of very large volumes of data
- Build robust ETL pipelines with resiliency
- Find a performant way to query very large volumes of semi-structured data
- Provide a user-friendly Business Intelligence UI, with fast slice-and-dice and aggregation capabilities
- Develop models and algorithms to accurately predict the performance of future advertisement campaigns

In the following paragraph, *Architecture*, we describe the architecture we created that solved these challenges in an elegant and maintainable way.

## Architecture

### AdsIntelligence Architecture

The diagram below illustrates the architecture of the AdsIntelligence platform:

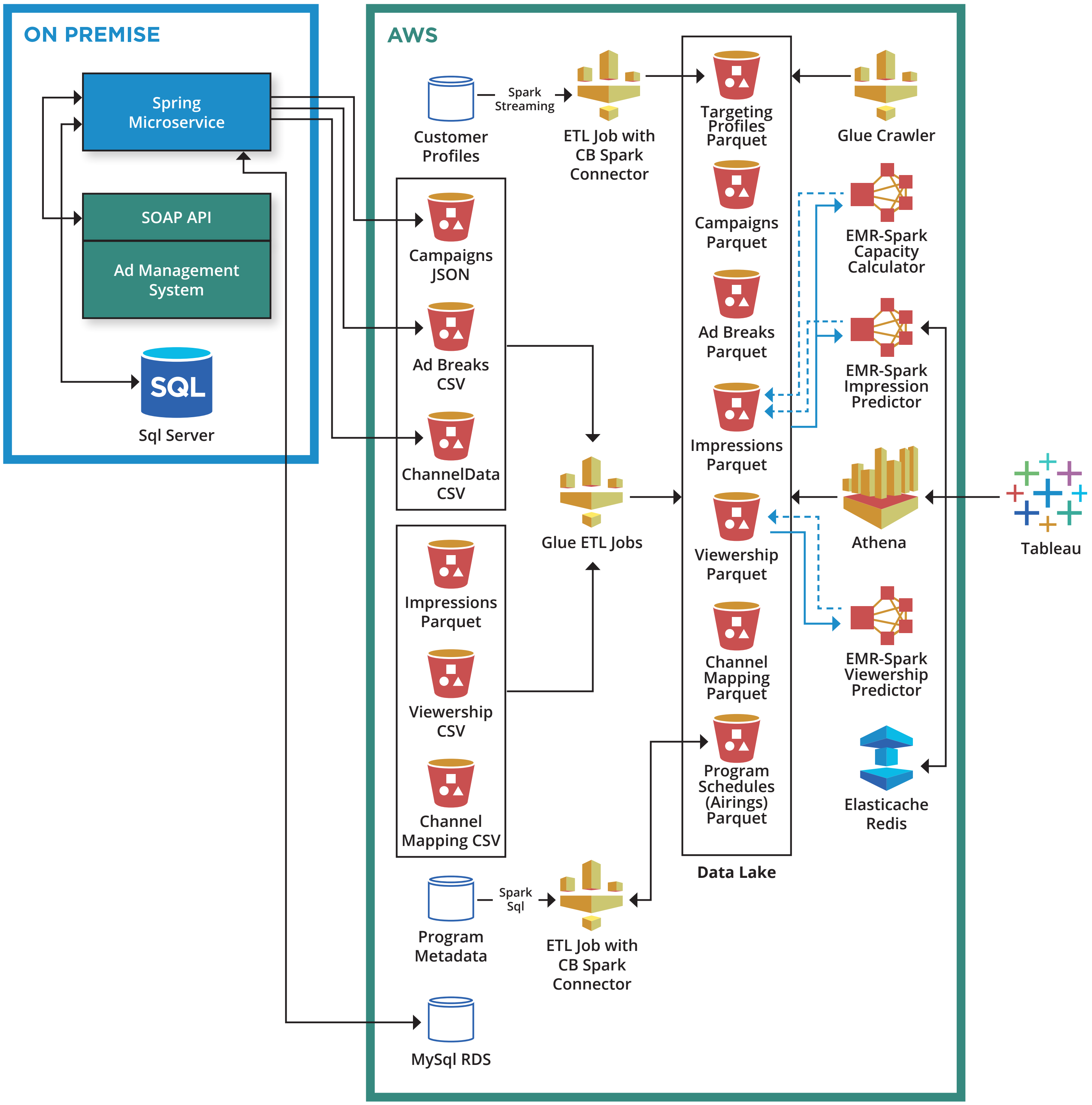

*Figure 2: AdsIntelligence Architecture*

Let's walk through the various components of the architecture illustrated in Figure 2.

## Campaign data extraction

On the left hand-side, we see components that are running in our client's on-premise datacenter. We deployed a cluster of small Spring Boot based microservices, whose responsibility is to regularly extract data from the advertisement campaign management system, via its legacy SOAP API, as well as perform some direct queries to the database for data that is not exposed via the API. The microservices then push the extracted data to a landing area in AWS S3 buckets.

## ETL pipelines and datalake

The larger right-hand side of the diagram contains a collection of components that are all running in a Production VPC on AWS. The bulk of these components are ETL pipelines that are based on AWS Glue workflows. The Glue jobs are responsible for loading data from the various data source buckets on the left, applying transformations and filters to the data, and writing the transformed data to a group of S3 buckets that make up our datalake.  All the data in the datalake is written using a format called Apache Parquet, which is a binary column-oriented format, and compressed using Google's Snappy compression algorithm. Where it makes sense, we partition the data using Hive-style partition folders, either at the hour or day level.

We'll discuss this in more detail in a dedicated paragraph, but it's important to point out that we leverage many of Glue's awesome built-in features, such as triggers, workflows, bookmarks, and crawlers.

We can see on the diagram that two of our ETL pipelines use Glue jobs to read data from Couchbase buckets. Couchbase is a hybrid document and key-value NoSql store, and we read documents from Couchbase DBs owned by other systems, one for customer profiles, the other for program schedules and metadata. Both jobs leverage the Couchbase Spark Connector.

## Heavy lifting: computational analytics and predictive algorithms

The complex analytics and predictive algorithms are performed by three different distributed computing jobs, using Apache Spark. They run on AWS's EMR service (Elastic Map Reduce), which offers the power and flexibility of running Spark jobs on a managed platform, where we can easily tune our jobs as far as number of CPUs, memory, number of executor instances, executor instance types, etc. (more on this below).

We leverage a first Spark job to run algorithms that predict future viewership events. Our model analyzes historical viewership behavior. The result, combined with the inventory of future ad breaks, is used to produce a set of expected viewership events over a sliding window of future time. We're proud to say we've achieved excellent accuracy in our viewership predictions (which also confirms that most people are creatures of habit and have semi-repetitive viewing behaviors).

A second Spark job is responsible for computing the ad impression capacity, which is the theoretical maximum number of ad impressions that customers could see if every single ad break was utilized in an optimal manner by the active ad campaigns. We compute a historical capacity, based on actual viewership events, as well as a predicted capacity, which utilizes our predicted viewership data (itself produced by the previously mentioned Spark job).

The piece de resistance is the third Spark job. Using a complex algorithm, it combines the predicted viewership data with in-flight and future campaign information, and creates a prediction of expected number of impressions, over a sliding window of future time. In addition to leveraging Spark on EMR, this job also makes use of Redis, the very fast key-value datastore, which we configured using the Elasticache managed service from AWS.

### What the users see: BI dashboards

It would be pointless to produce all this data in our datalake if we didn't provide the users with powerful visualization tools. We chose Tableau Server as our BI tool of choice, with which we built an array of user-friendly dashboards that allow users to slice-and-dice across our vast volumes of historical, current and predicted datasets.

The queries against our datalake are performed by an AWS service called Athena, which is built on top of Presto, a query engine that supports SQL-like querying across vast volumes of structured, semi-structured and unstructured data. In our case, Athena queries our Parquet files on S3.

## Choice of tools, stacks, and corresponding best practices

Here we'll briefly go through our choices for the tools and stacks that make up our architecture, as well as a few of the best practices we've learned.

### Datalake

The obvious choice for the storage of our data was **S3** (Simple Storage Service), which provides unbounded, resilient storage at a low cost. For the file format, we chose **Apache Parquet**, which is a binary, column-oriented format and is well suited to large, semi-structured, non-relational datasets. It stores the schema inside its metadata and has built-in optimizations for things like nullity and repetition of values. For compression, we chose Google's **Snappy**, an algorithm which offers a decent compression ratio, but more importantly is optimized for high compression/decompression speeds.

We used the **Glue Data Catalog** to store the schema and metadata of the tables built on top of the various datasets.

For time-based data, using Hive style (folder name based) **partitioning** offered substantial query performance optimizations.

With our choice of query engine, **Athena**, the recommended optimal file size is 128 MB, and we strove to keep our file sizes as close as possible to that recommendation.

The use of TTL retention policies at the S3 bucket or folder level helped us keep the size of our data bounded.

## ETL

ETL pipelines are clearly the backbone of our platform's architecture. To build robust and maintainable data flows, we picked **AWS Glue**, which is an ETL as a service. It is built on top of Spark and EMR (Elastic Map Reduce), and therefore offers great horizontal scalability and computing power. Glue supports two programming languages, Scala and Python. We chose **Scala**, which is one of the main languages supported by Spark, and therefore the de facto 1st class citizen. This also allowed us to mix in some Java libraries as needed.

We were able to leverage a slew of built-in Glue features:

- **Workflows:** the ability to assemble a sequence of Glue jobs and crawlers as a runnable unit.
- **Triggers:** fire off jobs based on schedules or conditions (such as the success of a preceding job).
- **Worker types and cluster size:** as mentioned above, Glue is an abstraction layer on top of Spark/EMR. We can configure two types of worker nodes (G1X and G2X), as well as define how many nodes a job uses.
- **Bookmarks:** this has got to be the greatest of all Glue features. Instead of having to code this yourself in Spark, Glue gives you the power of bookmarks. Each job keeps track of which source data it's already processed, so that you're always guaranteed to process only new data, even if failures occur.
- **Crawlers:** Glue crawlers traverse new data files and automatically detect new partitions. Crawlers can also automatically infer schemas, although we found it is a better practice to explicitly define the schemas in the Glue Data Catalog.
- **Network connections:** it's possible to configure Glue network connections to allow a Glue job to connect to various external systems. We use them to connect to Couchbase instances.
- **Console:** As with all AWS services, Glue provides a graphical UI.

## Query Engine

The choice of tool to query the data in our datalake was also obvious: **Athena**. Amazon's Athena is a service that is built on top of Presto, a distributed query engine originally built at Facebook. Athena is serverless, and it integrates seamlessly with the Glue Data Catalog. It supports ANSI SQL queries that can traverse our large datasets.

We use Athena queries extensively, both for live queries and scheduled queries that populate data extracts used by our BI tool (Tableau).

## BI Dashboards

As mentioned previously, having an awesome datalake would be pointless without a powerful BI tool on top of it, to offer users a collection of fast, dynamic, customizable and user-friendly dashboards. The dashboards should give users the possibility to visualize aggregated metrics, graphs of key variables over time, as well the ability to drill down and slice-and-dice across various facets of the data in an efficient manner.

The choice of BI tool was somewhat bumpier than for the other building blocks of our platform. We initially opted for Amazon's Quicksight, which is yet another SaaS service in AWS. Quicksight seemed like a suitable candidate for its ease of integration with other AWS services. However, we quickly ran into major limitations of the product, which we summarize here:

**Quicksight limitations & shortcomings:**

- SPICE data cache is limited to 250M rows
- Direct SQL queries have a timeout of 2 minutes
- A given widget can only use one dataset
- Parameterized queries are not supported
- Incremental refreshes of datasets are not supported
- Cannot run analyses within the widgets (such as a rolling sum)
- Cannot connect to more than 23 S3 buckets

Having run into the Quicksight limitations listed above, we pivoted and chose one of the major players in the BI space: **Tableau** (as seen on the right-hand side of the architecture diagram in Figure 2).

We run Tableau Server on AWS, and leverage many of its features, including:

- Easy integration with Athena (using AWS IAM credentials)
- Live queries
- Scheduled extracts: scheduled extracts run Athena queries to populate datasets stored on the Tableau servers; we use both full and incremental extracts
- Ability to combine datasets in a workbook
- Workbook level aggregations (e.g. rolling sums…)
- Parameterized queries
- Hover tool tips

## Distributed computing

In *Heavy lifting: computational analytics and predictive algorithms*, we described how we built models and algorithms that leverage our datasets to provide major value to the users of the platform. By performing massively parallel calculations, we compute three types of time-based indicators:

1. Impression capacity, both historical and predicted
2. Predicted viewership (over time, by channel and by customer)
3. Predicted campaign performance (predicted impressions over time, by ad campaign)

These recurring calculations are performed over datasets so large that a conventional, single-runtime (vertically scaled) approach would be fruitless. Therefore, our heavy lifting tool of choice is Apache **Spark**.

Spark is a distributed computing and analytics framework. Originally developed at UC Berkeley, it can be described as the grandchild of Hadoop. Whereas Hadoop, which performs map-reduce operations, is very I/O intensive as it stores intermediate results on disk, Spark uses the notion of RDD (resilient distributed datasets) which are partitioned across nodes in a computing cluster, and can therefore be thought of as distributed shared memory. Spark supports several cluster managers; we use **YARN**.

On top of the legacy RDD API, **Spark Sql** introduces the notion of DataFrames (a collection of immutable DataSets), on which we can perform transformations and actions using SQL-like APIs and query syntax. At the launch of a Spark job, an internal engine computes an optimized execution plan which is modeled as a directed acyclic graph (DAG).

We run Spark as an application in Amazon's **EMR** (Elastic Map Reduce) service, which makes it easy to configure and submit Spark jobs. EMR also provides user-friendly UIs.

From our ongoing experience with Spark, here are some takeaways and recommendations:

**Spark recommendations:**

- Leverage the Spark Sql API as much as possible
- Analyze the DAG in the Spark history server UI to understand performance bottlenecks and various issues such as out of memory problems
- Explicitly configure the number of CPU cores and amount of memory used by the executors (and the driver)
- Leverage caching (local persistence of intermediate results) of datasets that are used multiple times, to limit the amount of data shuffling
- Explicitly configure the number of shuffle partitions; the best practice is to use 2 times the total number of executor cores
- Eliminate data skews by using salted join columns
- Avoid expensive operations on the driver (such as count, reduce…)
- Repartition data across the cluster using a smart choice of key, to avoid subsequent data shuffling
- Explicitly set the garbage collector algorithm to G1 (Spark on EMR uses JDK 8, for which G1 is not the default GC algorithm)
- Use broadcast variables for immutable data sets that need to be shared across all executors
- Leverage broadcast joins when one side of the join involves a smaller dataset that can be broadcast across the cluster
- Test your code locally in the Spark shell
- Launch the Ganglia application on EMR to get a UI that monitors CPU, memory and other metrics across the Spark cluster

## Conclusion

In this article, we've described an interesting and challenging endeavor that PDG Consulting was hired to solve and implement: to provide our client with a powerful, state-of-the-art, custom Business Intelligence (BI) platform, that has become the unified, central one-stop shop for all their advertisement data. We've covered how we assembled the right SaaS and managed services to create robust data pipelines and populate an ever-growing datalake. By adding the proper query engine and BI tools on top of the datalake, we provide the users with the dynamic configurable dashboards that allow them to monitor and optimize their advertising campaigns.

Using innovative algorithms and the Spark distributed computing framework, we're also able to help our client predict the performance of future campaigns, and tune them accordingly, even before they become active in production.

We've shared some of the problems we encountered, as well as our takeaways and best practices. We are currently working on adding many more features to this platform, so stay tuned for the inevitable follow-up articles!

## About the Author

Fabien Coppens, Director of Engineering at PDG Consulting, is a solutions architect and technology leader with 25 years of experience in building distributed, scalable business platforms. He has architected and engineered critical, complex systems for a variety of industries including entertainment, e-commerce, banks, heavy industry, medical imaging, and telecoms. Fabien has worked on large-scale enterprise ecosystems, as well as several early stage startups where he built products from the ground up. He is a hands-on tech leader who has been consistently leading and mentoring teams of engineers. Fabien holds a PhD in Fluid Dynamics from the University of Toulouse, France, as well as an MS in Physics from the University of Paris, and a French "Grande Ecole" Engineer diploma in aerospace engineering from ISAE-SUPAERO.

## ABOUT PDG CONSULTING

PDG is a team of innovators and technologists delivering large-scale software application development services, end-to-end big data solutions, and SAAS tools for Fortune 500 and mid-size clients. We specialize in enterprise software development, business intelligence, and digital transformation—with expertise in translating complex business requirements into application design. Customers like WarnerMedia, Sony Pictures, Apple, DIRECTV, CBS, 21st Century Fox, and Amazon trust PDG to increase productivity, drive business growth, and maximize opportunities.

PDG

PDG Consulting | sales@pdgc.com | +1 (323) 347-4640

714 S. Hill Street, M1 Los Angeles, CA 90014