SANDIA REPORT

SAND2017-0406
Unlimited Release
Printed January 2017

The Bird Project

Using Big Data tools to support Search Analytics

John A Herzer, and Pengchu Zhang

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multi-mission laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@osti.gov
Online ordering: http://www.osti.gov/scitech

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5301 Shawnee Rd
Alexandria, VA 22312

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.gov
Online order: http://www.ntis.gov/search

SAND2017-0406
Unlimited Release
Printed January 2017

The Bird Project
Using Big Data tools to support Search Analytics

John A Herzer, and Pengchu Zhang
Knowledge Systems
Sandia National Laboratories
P.O. Box 5800
Albuquerque, New Mexico 87185-MS1325

Abstract

The Bird project explored the use of big data analytics tool to improve the findability of information within the Sandia internal network. We were able to perform query classification utilizing the supervised learning algorithms in the Apache Spark library. By relying on the distributed processing capabilities provided by the Apache Hadoop framework, we successfully processed the large query log files needed to train the models in this effort. The capabilities developed in this project are being used to enhance the effectiveness of the enterprise search engine.

ACKNOWLEDGMENTS

The authors thank Richard Hickey of the High Performance Computing group for his active support of the project as a system administrator.

CONTENTS

1. Introduction 7
1.1. The Need For Improved Search Metadata 7
1.2. Working With Search Log Data 7
1.3. Applying Advanced Analytics To The Search Domain 8

2. The Bird Development Environment 11
2.1 Hadoop / Spark Overview 11
2.2 Plato Hosting Environment 12
2.3 Software Development Environment 12

3. The Bird Machine Learning Pipeline 15
3.1. Preprocessing The Search Data 15
3.1.1. Loading The Search Log Data 15
3.1.2. Joining The Data 17
3.1.3. Cleaning The Data 17
3.1.4. Normalizing And Labeling The Data 17
3.1.5. Transforming The Data 18
3.2. Training The Model 18
3.3. Evaluating The Model 18

4. Conclusions 21

Distribution 23

FIGURES

Figure 1. The Bird Architecture 11
Figure 2. Sample Bash Script 12
Figure 3. The Bird Processing Pipeline 15
Figure 4. Sample Gobblin Pull File 16

TABLES

Table 1. Search Log Tables 16
Table 2. Query Types And Labels 17
Table 3. Confusion Matrix 19
Table 4. Model Accuracy 19

NOMENCLATURE

GB	GigaBytes
SNL	Sandia National Laboratories
SRN	Sandia Restricted Network
ATL	Applications That Listen

1. INTRODUCTION

The goal of the BIRD project is to demonstrate how big data infrastructure can support analytics for search data. We set out to show how query classification using supervised machine learning can be accomplished in a distributed, parallelized computing environment. By proving out this capability and gaining experience with big data tools and infrastructure, we open the door to a number of advanced analytic efforts aimed at improving search and information findability in general.

1.1. The Need For Improved Search Metadata

Enterprise search is inherently different from internet search. The extensive cross-linking between sites on the internet doesn't exist on intranet sites like the SRN. Consequently, the rich set of metadata that search companies like Google gather on the internet doesn't exist within enterprise networks. Not surprisingly, Google's PageRank algorithm, which boosts the ranking of sites that are linked to more frequently, isn't effective for enterprise search.

Modern search engines provide powerful capabilities for indexing content and retrieving documents that contain the query terms supplied by the customer. Without good metadata, however, they have difficulty determining which documents should be ranked at the top of the search results. Consequently, documents that contain the search term but that aren't particularly relevant to the query are often listed at the top of the results.

One approach to address the lack of metadata for enterprise content is to implement conceptual search. With conceptual search, we are searching for the concept or idea represented by the customer's search query rather than rely on exact term matching. To accomplish this, we need to understand the logical domain of the customer's query as well as metadata about our collections that identify the domain of individual documents. We can then match up queries with appropriate documents based on the similarity of the domains to which they belong

Query classification, then, is an important step on our path to achieving conceptual search. Search logs, consisting of query, results and click data can be of great value in classifying queries.

1.2. Working With Search Log Data

SearchPoint, Sandia's corporate search engine, records all activity related to customer search behavior in database logs. These logs include queries submitted by customers, the set of search result links returned by the engine, as well as the links that were clicked by the customer. In May 2013, the database administration department notified us that the size of the SearchPoint database logs were causing performance problems on GPPR1, the production database instance

where they resided. They worked with us to migrate nearly three years of data from the larger tables to an archive instance in the data warehouse. This archival eliminated the performance issues that they were seeing. Since then, though, the database has continued to grow at a rate of several GB per month and is now approaching 200 GB in total size. It appears that another data archival may need to be performed in the near future.

While the archival of older data eliminated the related performance issues, it limits what can be achieved with machine learning. The effectiveness of machine learning models increases with the amount of data that is provided to them. Our ability to identify patterns in search logs would be significantly enhanced if we could analyze all available search log data from a single online repository. Like many other companies, we are finding that modern enterprise computing infrastructure, with its reliance on relational databases, cannot handle the volume of data needed for large machine learning projects. Hadoop and Spark, on the other hand, were specifically designed to handle these large data sets. These tools have become a standard part of the big data environment because they address this problem by running parallel processes across a distributed set of nodes.

1.3. Applying Advanced Analytics To The Search Domain

One of the biggest challenges faced in the implementation of any search engine is relevancy ranking. For a search engine to be useful, the results must be ranked by relevancy with the most relevant documents listed first. There are two aspects to optimizing relevancy; local optimization and global optimization. Local optimization addresses the problem of ranking documents obtained from a single search index. There are a number of algorithms designed to address this problem, the most popular of which is Term Frequency – Inverse Document Frequency (TF-IDF). This algorithm boosts the ranking of a document based on how frequently a search term occurs in the document. It adjusts the boost factor based on how common each term in the query is across the entire corpus.

Global optimization addresses the need to rank results coming from federated sources. In a federated system, queries are made to remote systems in real time to obtain their relevant results. The results coming from multiple indexes or applications must be integrated and ranked appropriately. Because term frequencies and similar corpus statistics are typically not available from remote repositories, algorithms like TF-IDF cannot be utilized. One approach to global optimization is to classify customer queries by domain. If we know the probability that a query pertains to a specific domain, we can adjust ranking results from responding systems based on the domains they represent. Essentially, we are matching the domain of the query with the domain of responding systems.

The ATL framework is a type of federated system in which registered applications provide formatted answers for the customer's query. It is primarily responsible for intercepting requests from the SearchPointNext client and applying various degrees of artificial intelligence to construct a set of rich results that are returned to the client as a response. This platform interacts with a diverse set of Sandia Restricted Network (SRN) applications, services, databases, and Solr search indices to extract relevant information and construct these rich results. Since 2012, the ATL framework has evolved to host a variety of services that are consumed by the SearchPointNext application in order to create a richer, more personalized experience for the user.

The ATL framework's justification is largely based on the assumption that user search queries, and their associated metadata, often offer clues that can allow an intelligent application to construct a more useful, detailed, and personalized response for the user. As more applications are added to the framework, the potential for multiple applications constructing responses to individual queries becomes increasingly likely. To address the potential information overload associated with an increasing number of responses, query classification can help us assign confidence scores to application responses and determine the most appropriate ranking.

2. THE BIRD DEVELOPMENT ENVIRONMENT

2.1. Hadoop / Spark Overview

The BIRD project utilizes the Apache Hadoop product in order to efficiently analyze the large volume of search log data produced by the SearchPoint application. Hadoop is an open source framework for distributing data storage and processing across a large cluster of nodes. The Hadoop model relies on commodity hardware for the nodes and has a fault tolerant capability that anticipates hardware failures and provides automatic fail-over. The Hadoop Distributed File System (HDFS) is responsible for breaking up large data sets into equal-sized blocks and distributing them across the nodes in the cluster. It works hand in hand with MapReduce, the algorithm in Hadoop that is responsible for breaking up a job into multiple tasks and running them in parallel across the nodes of the cluster. The MapReduce process consists of two phases, the map phase and the reduce phase. In the map phase, the job is broken up into many small tasks, each of which can be performed on a single block of data. The processing code and the data on which it work are sent to available nodes, taking advantage of data locality by ensuring that each nodes only works with its local data. The output from each nodes processing is then integrated in the reduce phase using summary operations.

To improve efficiency, the BIRD project uses Apache Spark running on top of Hadoop to provide in-memory operations. By performing operations in memory, it is able to avoid the pervasive file IO activity in pure Hadoop installations, reducing latency by several orders of magnitude. With this toolset, it is possible to run iterative algorithms on the data, something that is not practical in the Hadoop environment. Spark also provides an SQL component, bringing support for structured data typically handled by relational databases. Figure 1 shows the elements of the big data environment used by the Bird project.

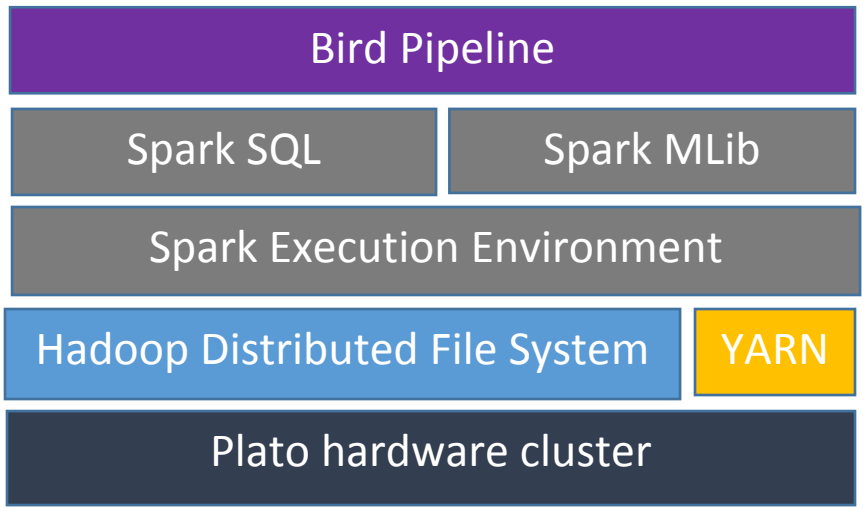

Figure 1: The Bird Architecture

2.2. Plato Hosting Environment

Plato is the Hadoop cluster implemented as part of the High Performance Computing initiative at Sandia. It is provided in the Cloudera Enterprise suite along with Hadoop cluster configuration management and monitoring tools. Hadoop version 2.6.0 and Spark version 1.6.0 are installed on this platform along with a number of common tools for the Hadoop ecosystem, including Avro, HBase, Hive, Mahout, Yarn and others. Currently, the Plato environment consists of 3 management nodes and 51 worker nodes running on Red Hat Enterprise Linux boxes. The cluster provides 1.5 petabytes of raw storage.

The Plato cluster is accessed via designated edge nodes. Customers can log into edge nodes using a Plato account that can be obtained from the WebCars application .

2.3. Software Development Environment

The Spark API supports three programming languages at this time: Scala, Java and Python. Spark itself is written in Scala and this language is very popular in big data projects. Scala was chosen as the language for this project because of its powerful functional programming capabilities and conciseness. While Spark provides an interactive shell for trying out the framework's functionality, the complexity of the Bird project's data transformations required submitting batch jobs via the spark-submit command. For scheduling and controlling the Spark jobs, YARN (Yet Another Resource Negotiator) was utilized. A sample Bash script used to invoke a Spark job is shown in Figure 2.

```
/usr/bin/spark-submit \
      --class SPNLoader \
      --master yarn\
      --deploy-mode cluster \
     --jars /home/jherzer/spark-csv_2.10-1.4.0.jar,/home/jherzer/commons-csv-1.4.jar \
     --executor-memory 10g \
     --driver-memory 10g  \
     target/scala-2.10/maraca_2.10-1.0.jar
```

Figure 2: Sample Bash script

3. THE BIRD MACHINE LEARNING PIPELINE

The BIRD project is architected as a collection of data processing applications that, when combined, expose a distributed machine learning pipeline for the classification of SearchPointNext search queries. This pipeline is essentially a workflow consisting of a set of ordered steps. By executing these steps in the prescribed order, the workflow will preprocess and store raw query data, implicitly generate record labels, train and store a series of one-vs-rest logistic regression models for query classification, and finally, evaluate the performance of the generated models using a held-out set of test data.

3.1. Preprocessing The Search Data

A number of preprocessing steps were performed in order to prepare the data for training of the logistic regression models. Figure 3 shows the components that comprise the project pipeline. Spark, itself, provides a pipeline mechanism for transforming data and fitting the data to a model. The Bird pipeline is built on top of the underlying Spark pipeline.

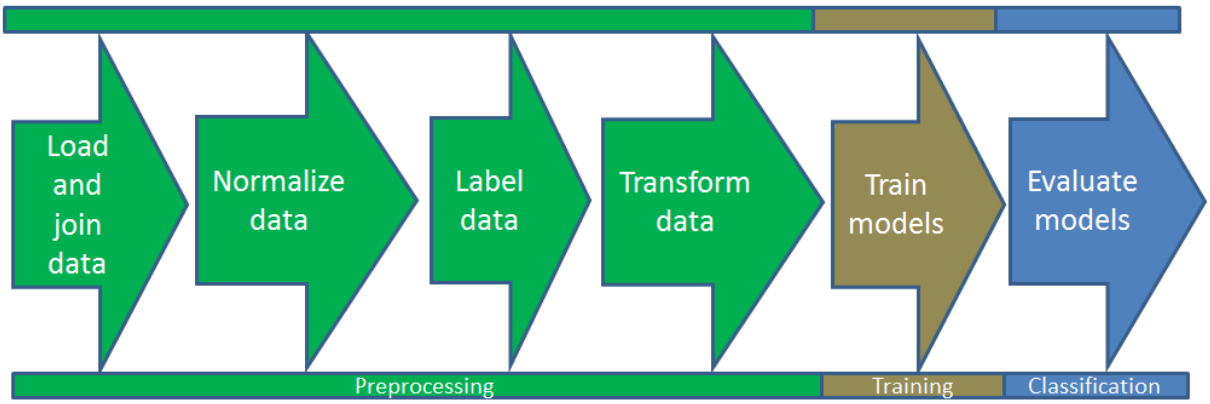

Figure 3: The Bird Processing Pipeline

3.1.1. Loading The Search Log Data

The search logs are currently stored as relational tables in the Oracle database. The table below summarizes the content of the log tables that are being used by the BIRD project.

Table 1: Search Log Tables

table	content	size
search_query	search terms submitted by customers	6.4M rows
search_result	search result links returned for individual queries	144.8M rows
search_result_set	metadata about a set of search results	6.4M rows
search_result_click	history of search result links clicked by customers	3.4M rows

To pull Search log data from the Oracle database, the BIRD project used Gobblin, a universal data ingestion framework for Hadoop. Gobblin is an open source software tool developed at LinkedIn. It was designed to support the extraction, transformation and loading of large volumes of data from a number of different data sources. Gobblin jobs have been created to pull the current contents of the search logs to HDFS storage and provide updates on a recurring basis for newly added content. Pull files were created to define the parameters for these jobs. The following pull file, for example, defines a job which will move content from the search_query table to HDFS storage.

```
# Job properties
job.name=SearchQueryGobbler
job.group=KnowSys
job.description=batch ingestion of searchpoint records
job.schedule=0 0 0 0/1 * ?

# Extract properties
extract.namespace=spt
extract.table.type=append_only
extract.delta.fields=received_date
extract.primary.key.fields=search_id

# Source properties
source.querybased.schema=spt
source.entity=search_query
source.querybased.extract.type=append_hourly
source.timezone=America/Denver
```

Figure 4: Sample Gobblin Pull File

3.1.2. Joining The Data

A number of joins were performed in order to create a single DataFrame (i.e. Spark table) that could be used for training the models. First, an inner join between the search_result and search_result_click dataframes was completed. Next, an outer join of the newly created join table and the search_result_set dataframe was performed. Finally, an outer join of the previous DataFrame and the search_query dataframe was carried out, resulting in a DataFrame where the tuples represented all user searches with corresponding clicks.

3.1.3. Cleaning The Data

Before the data could be normalized, it needed to be cleaned up. Filters were used to remove records with null query fields. In order to remove special characters and extra whitespace, SparkSQL User Defined Functions (UDF) were created. UDFs are column-based functions for transforming DataFrames that are defined using Scala functions.

3.1.4. Normalizing And Labeling The Data

Classification tasks, such as the one undertaken in the Bird project, are a form of supervised learning. As such, the data used to train the model is expected to contain labels. These labels are the output or response that we want the model to be able to predict. In our case the label for each search query would be the type of the query. Because the search query data does not contain labels itself, we needed to find a way to generate artificial labels for the data.

We identified 7 distinct query types that could be identified from the data. A combination of lookup tables and regular expressions were utilized to identify queries. Those queries that could not be put in one of these categories were identified as "other". This data normalization activity resulted in a set of consistent query term identifiers from which labels could be derived. A docNormalizer UDF was created that read in the query term values from the DataFrame and created a new column, "clean_normalized_query", which contained one of the assigned labels.

Table 2: Query Types and Labels

Query Type	Label
Building Number	6
Organization Number	5
Corporate Policy Number	4
SAND Report	3
Corporate Form	2
Top 100 Query	1
Other	0

3.1.5. Transforming The Data

The Spark ML library provides a pipeline API as the primary mechanism for coordinating machine learning steps. Pipelines (or workflows) are based on DataFrames and replace the Resilient Distributed Dataset (RDD) model utilized in older versions of the product. The pipeline API simplifies the assembly and configuration of distributed machine learning workflows. Pipelines consist of Transformers, which provide transformations on the data, and Estimators, which are used to fit the model to the input data. Since all input data is accessed via DataFrames, operations on the data are implemented as UDF applications. Transformations such as feature extractions are typically represented as new columns in the DataFrame, making debugging much easier.

Machine learning models rely on features, or independent variables, to evaluate the data and provide predictions. The feature chosen to train the query classification model was a TF-IDF score calculated on the normalized query field. A Spark pipeline containing the components needed to calculate TF-IDF and fit the model to the data was set up. This pipeline included a Tokenizer to split the query terms by white space, a StopWordsRemover to remove stop words from the queries, a HashingTF component to create the term frequency scores and an IDF component for the inverse document frequency calculation.

3.2. Training The Model

The SPNTrainingPipeline application is responsible for training the model with the query data. The preprocessed DataFrame created by the last application in the Bird pipeline is read in from HDFS storage and utilized for this purpose. 40% of the data was set aside for testing the model, the remaining data was used for training.

Logistic Regression was chosen as the type of model used in this project for classifying queries. Unlike linear regression, which provides continuous outcome values, logistic regression predicts a category for each observation. That is, the dependent variable is categorical. Since there are more than two possible categories, we need to use multinomial rather than binomial logistic regression. The Spark implementation of logistic regression doesn't currently support multinomial outcomes, so we utilized Spark's One-vs-Rest classifier. This approach relies on construction of binary classifiers for each category. The classifier that is most confident that an outcome belongs to its category will be selected for each observation.

3.3. Evaluating The Model

The final application in the Bird pipeline, SPNQueryClassifier, was responsible for evaluating the model against the test data held back from the previous training stage.

Table 3: Confusion Matrix

	Other	Top100	Forms	SAND	Policy	Orgn	Bldg
Other	1523760.0	0.0	21.0	2.0	5.0	8.0	10.0
Top100	0.0	966913.0	0.0	0.0	0.0	0.0	0.0
Form	292.0	0.0	16501.0	2.0	12.0	12.0	6.0
SAND	47.0	0.0	0.0	4093.0	0.0	0.0	3.0
Policy	55.0	0.0	6.0	0.0	17700.0	16.0	1.0
Orgn	75.0	0.0	40.0	5.0	12.0	37417.0	8.0
Bldg	84.0	0.0	43.0	0.0	12.0	65.0	25218.0

A confusion matrix was generated in order to describe the performance of the classification model. It consists of a table where the columns represent the instances of a predicted class and the rows represent the actual class membership of the data. This way of depicting the results makes it easier to identify false positives and negatives. The confusion matrix for the query classification exercise is shown in Table 3.

Utilizing the MultiClassMetrics class provided by Spark, we were able to obtain the accuracy for each of the models, as shown in Table 4. The extremely high accuracy of the models is likely due to the fact that labels were artificially generated for the search query data. Because the labels and the TF-IDF scores used as the model feature were both derived from the same dataset column, there is a high level of correlation between them.

Table 4: Model Accuracy

Category	Accuracy
Other	0.9999
Top100	1.0
Forms	0.9807
SAND	0.9879
Policy	0.9956
Orgn	0.9963
Bldg	9.9920

4. CONCLUSIONS

The Bird project was able to demonstrate the classification of search queries using supervised machine learning. We utilized big data infrastructure based on Hadoop and Spark to perform predictive analytics on a large set of search query, results and click data. It would have been difficult, if not impossible, to carry out this type of analysis in a traditional enterprise computing environment using resources such as relational databases.

DISTRIBUTION

1	MS1325	John Mareda	09537
1	MS1325	Pengchu Zhang	09537
1	MS0899	Technical Library	9536 (electronic copy)


Sandia National Laboratories