Software Requirements Specification

for

DAKA

Version: 1.3
February 4, 2014

Prepared by:
Sergey Matskevich, Ernesto Cejas Padilla, Christopher Harvey,
ByeongGil Jeon and Jirakit Songprasit

Stakeholder: iPipeline
Instructor: Msc. Bill Mongan, Dr. Jeffrey Popyack

Drexel University

Revision History

Name	Date	Comments	Version
Sergey Matskevich	January 7, 2014		1.0
Sergey Matskevich	January 20, 2014	Added business-oriented background and scope details. Added MapReduce and HDFS sections. Made functional requirements more concrete, added interface requirements. Updated hardware requirements.	1.1
Chris Harvey	February 3, 2014	Added product description, business cases and citations.	1.2
Sergey Matskevich	February 4, 2014	Modified description, add frequent pattern and classification descriptions, modified requirements to be more specific. Moved some requirements from functional to expected category. Updated scope of the project.	1.3

Table of Contents

1 Introduction

1.1 Purpose

iPipeline is in need of a flexible tool that will process large amounts of data on a daily basis. As new data is coming in it must be analyzed and processed in a timely manner. This will allow the business to use new information to be more competitive and make better strategic decisions on the market.

DAKA will utilize horizontally scalable distributed system Hadoop and implement algorithms for it in order to achieve required performance. For this DAKA will be deployed on a cluster where number of nodes can grow as needed in order to accommodate data growth and processing requirements as needed. New data will be analyzed in order to make predictions about possible customer base expansion and improving customer relations with existing customers.

1.2 Scope

DAKA will process all data coming from iPipeline product Data Rail, which collects and standardize all information from all sources in the company. DAKA will find common patterns in the data and store them for future use in predictive analysis. Data will be exported, sanitized, cleaned and stored in HDFS by iPipeline in CSV format. The tool will read files from specified location in HDFS and pass it to MapReduce framework for processing and analysis. After processing is done, produced results will be saved in specified location in HDFS. DAKA will accept any normalized data set and generate frequent patterns based on the data, which later can be used as a training data set for classification algorithm. A separate training data set will be provided by iPipeline in order to run classification without finding frequent patterns. Terminal interface will be used in order to invoke DAKA and set its run time. DAKA can also be scheduled to run by automated system, such as Cron.

1.3 Background

1.3.1 iPipeline

iPipeline leads its industry in providing the next-generation suite of sales distribution software to the insurance and financial services markets through its on-demand service.

iPipeline's channel solutions for carriers, distributors, producers and financial professionals automate activities for CRM, forms distribution and processing, quotes and illustrations, in Good Order e-Applications, agency management, data services, policy delivery and related services, enabling the insurance industry to market, sell, and process faster. (Internet Pipeline, Inc. n.d.)

1.3.2 Hadoop

The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures. Hadoop mainly relies on HDFS for reliability, which provides data redundancy and fault tolerance. (The Apache Software Foundation n.d.)

1.3.3 The Hadoop Distributed File System

The Hadoop Distributed File System (HDFS) is a distributed file system designed to run on commodity hardware. It has many similarities with existing distributed file systems. However, the differences from other distributed file systems are significant. HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware. HDFS provides high throughput access to application data and is suitable for applications that have large data sets.

HDFS spreads files across multiple nodes and create duplicate pieces of the files on different nodes. This provides fault tolerance in case hardware failure – the computation process will just be moved to a different node that has missing data. It is much more efficient to execute application near the data it operates on. This minimizes network congestion and increases the overall throughput of the system. HDFS provides interfaces for applications to move themselves closer to where the data is located. (Borthakur n.d.)

1.3.4 MapReduce

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key.

Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system. (Dean and Ghemawatt 2004)

1.3.5 Frequent Patterns

Frequent patterns (itemsets, subsequences, or substructures) are also called association rules. Frequent pattern mining is a method for discovering interesting relations between variables in large databases. Patterns could be a set of attributes that appear in a transaction together or a sequence of events (purchases) that happens often. In the case of iPipeline, a pattern could be a male in his 40s, with two children, living in South Philadelphia with an income of $70,000 would imply that he is likely to prefer Pacific PremierCare Advantage insurance plan.

1.3.6 Classification

Data classification is a two-step process, consisting of a learning step where a classification model is constructed and a classification step where the model is used to predict class labels for given data. During the learning step the algorithm consumes observation data with known class labels. After the training data is consumed, class labels will be assigned on new data based on the transactions from learning step.

1.4 Requirements Apportioning

The Priority levels for the requirements are:

Priority	Description
P1	All requirements of this level must be fully satisfied and verified in order for the software system to be released.
P2	Requirements of this priority are not expected to be fully satisfied in the current release. These requirements are expected to be fully satisfied and verified at the next release of the system.
P3	These requirements are not within the current scope of the system design. These requirements are included only to indicate where the software is expected to change in future development.

1.5 Glossary

1.5.1 Apache ZooKeeper - A centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services.

1.5.2 Closed Itemset – an itemset with no proper superset that has the same support count

1.5.3 Confidence – the conditional probability of a set of attributes to appear in the data if this set has a given support.

1.5.4 Cron – a time-based job scheduler in Unix-like computer operating systems

1.5.5 Data Node – a machine in HDFS which stores file data

1.5.6 Frequent Itemset – an itemset that occurs in the data with specified minimum support

1.5.7 Frequent Pattern – a set of individual attributes that is shared among many different entities of data

1.5.8 Hadoop Distributed File System (HDFS) – a special file system that Hadoop uses for file storage.

1.5.9 Horizontal Scalability – a feature of software to increase performance linearly with the addition of new nodes in a cluster

1.5.10 Itemset – a set of items in a transaction

1.5.11 Maximal Itemset – an itemset that is not a subset of any other itemset

1.5.12 Name Node – a machine in HDFS which keeps the directory tree of all files in the file system, and tracks where across the cluster the file data is kept

1.5.13 Transaction – a row of input to DAKA or a tuple in a database

1.5.14 Support – the probability of a given set of attributes to be in a random entry in database

2 Description

2.1 DAKA

2.1.1 Overview

DAKA will be a tool for statistical data analysis. It will help iPipeline to find association rules and interesting trends in their data and to identify possible products that can be offered to a particular customer.

2.1.2 Data Handling

In order for DAKA to classify existing or potential customers, it must be flexible enough to handle different sources of data that can vary in size and type. For this reason, DAKA will be built on top of HDFS. Because HDFS is designed to scale, DAKA will be able to handle large data sets. Since HDFS is a general file system it is able to store data in any desired format.

2.1.3 Algorithmic Capabilities

DAKA will include implementations of classification and frequent pattern mining algorithms. The frequent pattern algorithm will make two passes on the data and identify interesting dependencies among transactions. In order to do that DAKA will build maximal closed frequent itemsets and then generate association rules based on those itemsets. The output of this algorithm will contain itemset, association rule and support, confidence and lift as statistical measures for this rule.

The classification algorithm will split the data by attributes according to maximum information gain in each step, until a result can be reached. After the algorithm runs each tuple in the database will have a class label associated with it. The output will contain all of the fields from database and a class label so the agent will be able to identify reasoning behind assigned class and verify its correctness.

These algorithms have to scale linearly with hardware power increase, so they will be implemented using the MapReduce paradigm, which will make them automatically parallelizable and scalable almost linearly.

2.2 Business Cases

2.2.1 Overview

Since DAKA will be built for iPipeline, solutions will be developed for two business cases. These are customer classification and demographic breakdowns. Customer classification is the process of identifying what product a customer is likely to buy based on their demographic information. Demographic breakdowns will provide detailed analysis about customers who purchase different products.

2.2.2 Customer Classification

Customer classification will identify what product a customer is likely to purchase. This classification will happen by running a classification algorithm. This classification algorithm will use fields describing a customer, such as demographic information, to help identify which product matches that customer. It will learn these associations by training with a body of data that includes the products customers have purchased. Because DAKA will include implementations of classification algorithms, this business case will take advantage of these offerings.

2.2.3 Demographic Breakdown

Demographic breakdown will provide informative data about the characteristics of customers of different products. Looking at the characteristics of customers of specific products will be informative. Things such as age distribution, gender division, and geographic trends will educate iPipeline on associations with individual products. This business case will use the frequent pattern algorithms that will be included in DAKA.

3 User Requirements

3.1 Functional Requirements

3.1.1 The tool must be able to determine frequent patterns from the data set. **P1**

3.1.2 The tool must be able to assign class labels based on frequent patterns generated by the tool and using normalized training data set. **P1**

3.1.3 The tool must be able to classify customers based on data training provided by iPipeline. **P1**

3.1.4 The tool must have web interface with dashboard that lets users to view visual representations of frequent patterns and correlations. **P3**

3.1.5 The tool will export interesting associations into data cube. **P3**

3.2 Input

3.2.1 The tool must accept normalized data with all attributes from iPipeline's database. **P1**

3.2.2 The input file format must be CSV. **P1**

3.2.3 The tool must be able to work on a normalized data set of any size. **P1**

3.2.4 The tool must read input from specified location in HDFS. **P1**

3.3 Output

3.3.1 General output requirements

3.3.1.1 The tool will produce output to specified location in HDFS. **P1**

3.3.2 Frequent pattern output

3.3.2.1 The output format of frequent patterns will have the following format:

Key: 0: Value: ([0],80), ([141, 0],42), ([132, 0],42) **P1**

3.3.2.2 Key field will represent an attribute from the data set. **P1**

3.3.2.3 Value field will represent the list of all interesting associations in the dataset. **P1**

3.3.2.4 Items in square brackets, for example ([141, 0]), will be associated itemsets with a particular key. **P1**

3.3.2.5 The number after an itemset will be the support count for that itemset (frequency of occurrence for an itemset in the database). **P1**

3.3.3 Classification output

3.3.3.1 The output will be in the following format:

Attribute1, Attribute2, Attribute3 => class label **P1**

3.3.3.2 Attributes represent data fields in the database. **P1**

3.3.3.3 Class label will be learned from training data and will be a product type sold by iPipeline customer. **P1**

3.4 Expected Requirements

3.4.1 Frequent Patterns must have high enough support and confidence for making solid prediction based on this pattern. **P1**

3.4.2 Predictions made by the tool must have high probability of correctness. **P1**

3.4.3 The tool must scale linearly with addition of new nodes to the cluster. **P1**

3.3 Interface Requirements

3.3.1 The tool must be invoked from the command line. **P1**
3.3.2 Ability to set input folder:
-i [PATH] must specify data input folder in HDFS. **P1**
3.3.3 Ability to specify output folder:
-o [PATH] must specify data output folder in HDFS. **P1**
3.3.4 The tool must be able to run frequent patterns and classification depending on user's choice:
-t FPGrowth must run frequent patterns algorithm.
-t Classify must run classification algorithm. **P1**
3.3.4 User must be able to specify minimum support:
-s [VALUE] flag must specify minimum support value. **P1**
3.3.5 User must be able to specify required confidence:
-c [VALUE] flag must specify required confidence value. **P1**
3.3.6 The tool must accept a path to training data:
–l [VALUE] flag must specify the location of training dataset. **P1**

4 System Requirements

DAKA runs on top of Hadoop, the following are recommended Hadoop requirements. The details and explanations are in the Hadoop hardware guide. (Hortonworks, Inc. n.d.)

3.1 Software

Linux based operating system, Java 6u15 or later, Hadoop 1.2.1 or later.

3.2 Hardware

See Hadoop recommended cluster building guide. (O'Dell n.d.)

4 Appendices

4.1 References

Borthakur, Dhruba. n.d. "HDFS Architecture Guide." *Apache Hadoop.* https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html.

n.d. *DAKA Project.* http://charvey.github.io/DAKA.

Dean, Jeffery, and Sanjay Ghemawatt. 2004. "MapReduce: Simplified Data Processing on Large Clusters ." *Sixth Symposium on Operating System Design and Implementation,.*

Hortonworks, Inc. n.d. "Hardware Recommendations for Hadoop." *Hortonworks Data Platform Documentation.* http://docs.hortonworks.com/HDPDOCS/HDPv1.0.1.14/index.htm#About_Hortonworks_Data_Platform/Typical_Hadoop_Cluster.htm.

Internet Pipeline, Inc. n.d. "About iPipeline." *iPipeline.* http://ipipeline.com/company/about-ipipeline.php.

O'Dell, Kevin. n.d. "Hadoop Cluster Hardware Guide." *Cloudera Developer Blog.* http://blog.cloudera.com/blog/2013/08/how-to-select-the-right-hardware-for-your-new-hadoop-cluster/.

The Apache Software Foundation. n.d. *Hadoop Home Page.* http://hadoop.apache.org/.