Data Analytic Tools for Inconsistency Detection in Large Data Sets

Final Report

Team 27
Client - Kingland
Advisers - Cai Ying
Team Members/Roles -
Logan Heitz (Team Lead), Camden Voigt (Technical Lead),
CJ Konopka (Communication Lead), TJ Rogers (QA Lead)
Team Email - sdmay18-27@iastate.edu
Team Website - http://sdmay18-27.sd.ece.iastate.edu/
Revised: April 22, 2017 Version 1

Table of Contents

1 Introduction

1.1 Acknowledgement

This project would not be possible without the assistance of the faculty advisor Dr. Cai. Working with Dr. Cai on this project are two graduate students Guolei Yang and Zehua Li who have been invaluable in design and implementation of this project. Finally, this project relies on the support of Kingland Systems for testing data and any other needed materials for implementation of the project.

1.2 Problem and Project Statement

Kingland processes a large amount of data that it receives from its clients everyday. This data can relate to customers, companies, or agreements between entities. This data is compared to a central database in order to detect inconsistencies. An example of an inconsistency would be two customer records containing the same social security number, but different names. This is an issue, since a social security number should be unique. The database contains over 100 million records, and around 10% of these records are updated or inserted daily. Due to its size, this comparison takes about 24 hours to run using Kingland's current solution. This time stems from the fact the entire database cannot be loaded into main memory at one time and the use of SQL inner join statements to check for inconsistencies, which is inefficient. Kingland would like to process 500,000 records for inconsistencies against the central database faster than the current solution, preferably in under an hour.

1.3 Intended Users and Uses

This project will supply Kingland's analysts with information on inconsistencies within their records. The product will be on the backend of Kingland's system and its processes will be automated. As such, no users will directly interact with our system on a day to day basis as Kingland will display the results of our output using their own user interface.

2. Project Design

2.1 Specifications

2.1.1 Functional Requirements

- Solution must not use SQL inner-join statements
- Solution must utilize only relevant information
- Solution must compare records from an incoming report against the central database as well as other records in the same report
- Solution must update inconsistency database after analysis

2.1.2 Non-Functional Requirements

- Solution must perform inconsistency check faster than current solution
    - Preferably in less than 1 hour.
- Solution must be able to analyze 100 million or more records at a time
- Solution must find all inconsistencies from an incoming report

2.1.3 Constraints

- Solution must work with MySQL
- Solution must run on AWS
- Must handle XML input

2.2 Design Overview

Our solution to Kingland's problem is to remove the heavy dependency on SQL and introduce parallelization to improve performance. Currently Kingland's system is heavily reliant on SQL to make large and complex queries to the database. By adding in more preprocessing to the system we are able to simplify the SQL queries that we make and spend less time getting information from the database. We also utilize threading to make multiple SQL queries at the same time and get further performance gains.

The design also incorporates a lot of configurability, so that Kingland can make changes in the future. This is achieved through the use of two configuration files, the Data Configuration File and the Inconsistency Configuration File. The Data

Configuration File allows for changes to what fields should be kept for consistency checking. While the Inconsistency Configuration File allows for Kingland to change what Inconsistencies are checked and add new ones. This will allow for Kingland to setup different configurations of the program for different types of incoming reports.

Our design utilizes an event driven XML parser as opposed to a tree model XML parser for the incoming XML reports. This allows us to parse out one record at a time and conserve memory. As each item is parsed it will be immediately checked for inconsistencies, once it is no longer being checked it is no longer needed in memory and the next record can be loaded. When the records are being checked for inconsistencies each inconsistency will be checked in its own thread to speed up the check. An overview of the different modules used in the project can be found in *Figure 2.1*.

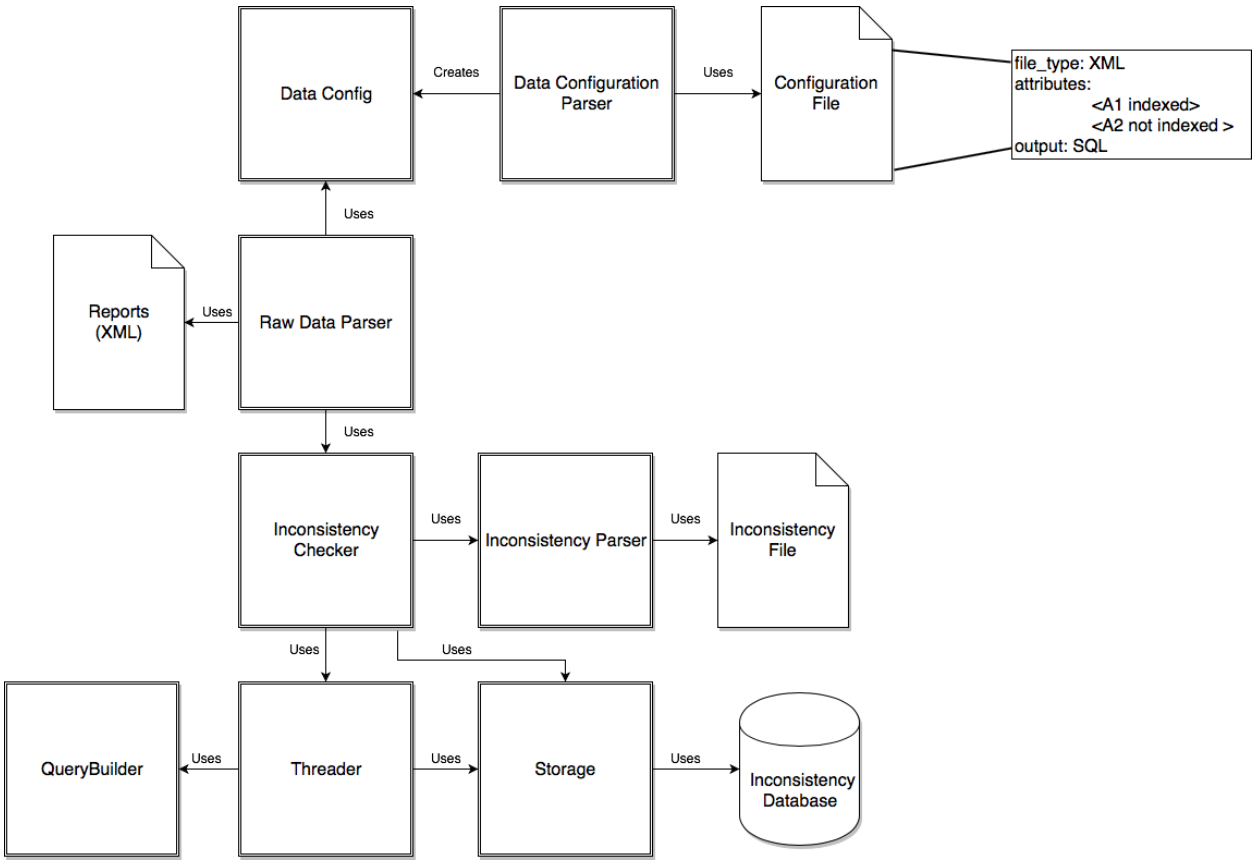

Figure 2.1: Block diagram of high-level system architecture

2.2.1 Module Design

This section will go over the basic design of the different modules in the project and what functionality they will provide for the project.

Data Configuration

The purpose of the Data Configuration module is to allow the user to give information needed for a run of the solution via a configuration file that the program parses at run time. This allows the user to specify different configurations each time the program is run. The options users may include in their configuration file are the input location of raw data files, database information, and key element specification.

Raw Data Parser

The Raw Data Parser allows the program to parse over raw data files containing records. The parsed records are saved to the system based on the configuration the user has specified. Each record is sent to the Inconsistency Checker to determine if it conflicts with any existing records.

Inconsistency Parser

This module allows the user to provide a configuration file that outlines the inconsistencies that need to be detected. The parser then parses the file and creates a list of inconsistencies for the Inconsistency Checker to use.

Inconsistency Checker

The Inconsistency Checker provides the ability to compare a record against those already in storage and detect inconsistencies. Checking of the inconsistencies is done in separate threads, one for each inconsistency provided by the user. These threads are given to the Threader to be executed. The results are then compiled and any detected inconsistencies are sent to the Storage module.

Threader

The Threader module accepts threads from the Inconsistency Checker and executes each of these threads in parallel. The threads use the Storage module to make SQL queries to detect inconsistencies.

Storage
The Storage module interacts with the storage medium specified by the user in configuration file. It provides the ability to make queries to detect an inconsistency. It also provides an interface that allows for saving of records and inconsistencies to the storage medium.

2.3 Design Analysis

2.3.1 Specification Fulfilment

The design laid out in section 2.2 fulfills the specifications laid out in section 2.1 through a variety of methods. The use of a configuration file allows us to handle various forms of input and to utilize only the relevant information for any given scan by allowing Kingland to specify those items before the program runs. By making smaller SQL queries we are able to reduce the time the queries take, which leads to improved performance. Parallelization provides further performance gains over the existing solution.

2.3.2 Strengths

Our solution has several strengths that make it an appropriate choice for Kingland's problem. The first strength is the ease of implementation. This solution is built on a few small parts that can be implemented with relative ease. Another strength of this solution is its modularity. The solution is separated out into several distinct parts and changes to one part will not require changes to the entire design. The design of the configuration file allows us to quickly add in new inconsistencies if needed and modify the settings of current ones to adapt to any changes we encounter. Finally, parallelization is a strength of our solution. The parallelization takes advantage of our smaller SQL queries to complete several queries at a time, providing improved performance.

2.3.3 Weaknesses

The biggest shortcoming of our solution is that a third party solution may be able to do this job almost as well. In this case it may be easier for Kingland to use this third party solution as it would have better support from a full development team, and could be used in some of Kingland's other solutions. The benefit of our solution over this is that it has been developed specifically for Kingland's needs and they will be able to modify it if needed in the future. In addition, it will likely cost Kingland less to use in the long run than a third party solutions.

3 Implementation

3.1 Implementation Details

Our solution has been implemented and works as outlined within the design section. It is able to parse and detect inconsistencies from XML files by querying a MySQL database. Below are implementation details for the different modules within the project.

Data Configuration

The data configuration file will be an XML file with three major sections. The first of these is the input path which will take the path to either a file or a folder. *Figure 2.2* shows how the input path should be specified if it is a file or a folder.

```
<inputpath>/path/to/file.xml</inputpath>
<inputpath>/path/to/folder</inputpath>
```

Figure 3.1: Example XML input tags

If a file is specified it will be used as the report and if a folder is specified it will utilize all files within the folder. This allows multiple reports to be easily processed.

The second section is the storage. This section specifies information on the storage used by the application. Currently only MySQL databases are supported. An example of the storage section from the configuration file can be seen in *Figure 2.3*.

```
<storage-list>
	<storage type="database">
		<location>http://localhost:3306/database</location>
		<driver>mysql</driver>
		<username>myusername</username>
		<password>mypassword</password>
	</storage>
</storage-list>
```

Figure 3.2: Example XML storage tags

Finally, there is a key element section that contains information on all the attributes that we will need from the records contained in the report. This includes the key for the attribute, a list of alternative keys that may be seen, and finally whether the attribute should be used as an index. The indexing allows us to have quicker lookup on fields that are used for inconsistency lookup. *Figure 2.4* gives an example of how the key-elements are formatted in the configuration file.

```
<key-elements>
	<element key="ssn" index=’true’/>
	<element key="name" optional-keys="firstname,lastname" index='false'/>
</key-elements>
```

Figure 3.3: Example XML element tags

The parser will convert the XML configuration files into java objects. These configuration objects are then used in the rest of the program to determine user settings. This module uses the Java XML DOM parser as the XML file being parsed is small, so it is faster to load the whole tree into memory at once and then parse it.

Raw Data Parser

The Raw Data Parser has been implemented using the SAX parser for XML. This parser was chosen because it is event based instead of tree based. This means that the whole XML file is not read into memory at one time, which is better when working with large XML files. This also allows us to parse a single element at a time and have it sent to the Inconsistency Checker to be checked and then remove that element from memory before reading in the next element.

Inconsistency Parser

The inconsistency configuration file is used to specify what inconsistency the program should detect. This is an XML file that contains a list of inconsistencies that should be checked. The inconsistency status is the status code associated with that inconsistency. The index is the value that should be used to pull records from the database as the corresponding columns will be indexed in the table. The index value should correspond to keys in the data configuration file that have a true value for index. The compare attribute is the value that needs to be compared in checking.

```
<inconsistencies>
	<inconsistency type="same" status="1" index="TID" compare="NAME">
```

```
        <inconsistency type="different" status="1" index="SSN" compare="NAME">
</inconsistencies>
```

Figure 3.4: Example XML inconsistency tags

The parser will convert each inconsistency tag into a java object so they can be used by the Inconsistency Checker. The XML parsing also uses Java XML DOM parser as the inconsistency configuration file will be small.

Inconsistency Checker

The Inconsistency Checker compares incoming records against existing records within the inconsistency database to find inconsistencies. It starts by sending the record to the storage module to be added to the database. Then a separate thread is created for each inconsistency that is being checked. Each thread is setup to query the database for all records that have the same index value, but a different compare value. If a conflict is found, an Inconsistency Record is created for each conflict and sent to the Storage module to be added to the Inconsistencies Table in the database.

Threader

The Threader accepts threads that query storage to check inconsistencies. The Threader is implemented using a fixed thread pool and as each thread is received by the threader it is sent to the pool to be executed or to be added to the queue. The threader provides a blocking call that waits until all current jobs are finished.

Storage

Our design allows for flexibility in extending this to other storage mediums, such as other databases or a storage file, in the future. However, the current implementation only supports a MySQL database. This module uses JDBC to support connection with the database and uses the Apache Commons DBCP library to allow for connection pooling, which improves performance for our multi-threading.

Inconsistency Database

The inconsistency database stores two types of information, records and inconsistencies detected. There are two types of tables to do this. The first is a record table which holds the incoming records. The columns for this type of table are based on the key elements outlined in the configuration file. Each record also has a unique identifier. The record table is index using a B-Tree based on the

index values provided in the Data Configuration File. This will allow for quicker lookup times in inconsistency checking. The second kind of table is the inconsistency table, which holds the detected inconsistencies. Each inconsistency has five pieces of information: a unique id, the id of the reporter for the incoming report, the id of the record from the incoming report, the id of the reporter from the existing record, and the id of the existing record.

3.2 Class Diagram

The class diagram is provided below in figure 3.1. The diagram shows the relationships and major interfaces for the classes that are implemented as part of this project.

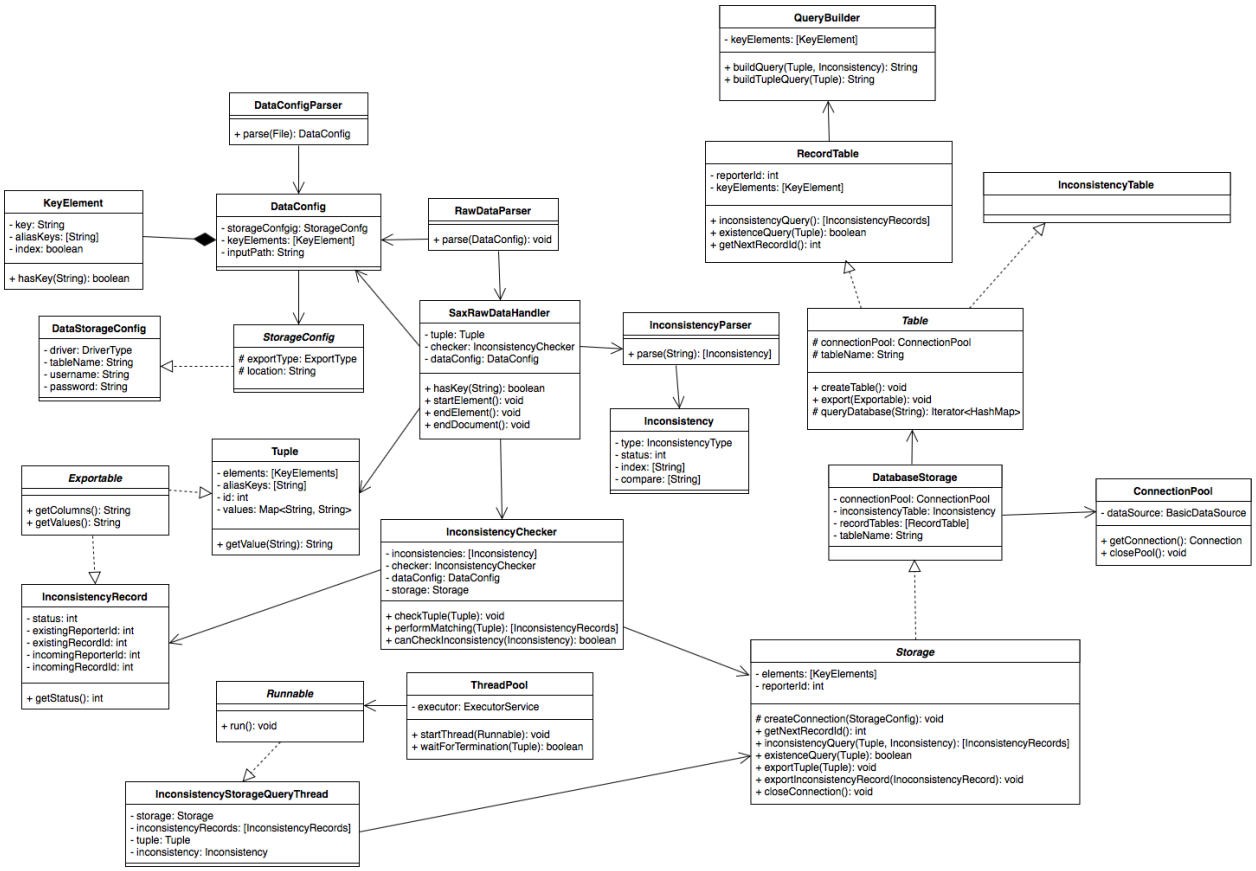

Figure 3.5: *Class Diagram*

3.3 Dependencies

- **Apache Commons CLI**: This is used to parse command line options when the program is started.

- **Apache Commons DBCP**: This is used to provide connection pooling to the storage module.
- **Apache Log4j**: This provides logging functionality to the project.
- **SAX Data Parser**: This is the event driven XML parser used to parse the raw data files.

4 Testing Process and Results

4.1 Process

Testing is a large component of our development cycle and requires the use of several different testing strategies. These strategies help ensure that the solution we develop meets the needs of Kingland and that we maintain the functionality throughout development of the program. The testing process we followed involved Test Driven Development, Unit Testing, Integration Testing and Performance Testing.

4.1.1 Test Driven Development

The first part of our testing process is to follow the three basic Test Driven Development (TDD) guidelines. These guidelines are:

1. Create a test and make it fail.
2. Make the test pass by any means necessary
3. Change the code to remove duplication in your project and to improve the design while ensuring that all tests still pass.

We felt this was important to establish at the beginning of the project because it would encourage the development of a well rounded test suite. By following the TDD process, we felt that we could ensure a high percentage of code coverage, or percentage of lines of code tested, in our unit tests.

4.1.2 Unit Testing

The next part of our testing process was the unit tests, which tie heavily into Test Driven Development. The theory behind unit tests is that a developer uses a unit test to assert that in a given scenario gives a correct output. This is usually testing a specific section of code or function. We chose to use the JUnit and Mockito testing libraries to make our tests more comprehensible and verbose.

JUnit
JUnit is a unit testing framework designed for the Java programming language. It provides developers the ability to determine the correctness of functions through various assert functions. These unit tests can also be integrated with gitlab to help developers ensure that changes made to the code base do not alter functionality that has already been developed and implemented.

Mockito
Mockito allows programmers to create and test double objects (mock objects) in automated unit tests for the purpose of Test-driven Development (TDD) or Behaviour Driven Development (BDD). Mock objects allow developers to create objects that will function as desired without actually needing an implementation. The use of Mock objects in unit tests is very important because unit tests should be created around a single function of the code regardless of the actual implementation of the code, so unit tests of code that have many dependencies can be performed modularly.

4.1.3 Integration Testing

While Unit Testing is a large part of the development cycle, some dependencies are most effectively tested by actually running various data through the system. These types of tests are called Integration tests because they provide feedback on how well the system components integrate with each other. Because our product uses a database as a storage source, we perform integration tests that revolve around querying and updating a database.

4.2 Testing Criteria

Kingland requires that our product meets several functional and non-functional criteria. These requirements are primarily concerned with the product's performance, configurability, security and accuracy. We sought to write tests that would allow us to test these criteria and measure how successful we were during our development. Listed below are all of the functional and non-functional criteria. Tables 4.1 and 4.2 below specify how each criteria is validated

4.2.1 Functional Criteria

Criteria	Validation/Acceptance test
Solution must not use SQL inner-join statements	A Style Checker is used to ensure that SQL inner-join statements do not appear in the production code.
Solution must utilize only relevant information	The size of the Inconsistency database created by this solution shall be compared to Kingland's central database to determine if this requirement is satisfied by our solution.
Solution must compare records from an incoming report against the central database as well as other records in the same report.	Unit tests are used to validate that records are properly checked against those existing in the database prior to the run as well as other records contained in the incoming report.
Solution must update inconsistency database after analysis	Unit tests are used to validate that inconsistencies are correctly sent to the database following the completion of the check.

Table 4.1: Validation tests for the Functional requirements

4.2.2 Non-Functional Criteria

Criteria	Validation/Acceptance test
Solution must perform inconsistency check faster than current solution. Preferably in less than 1 hour	This is validated by comparing performance log files gathered in Log4J to determine the success of this.
Solution must be able to analyze 100 million or more records at a time	This is validated using tests on large data sets in order to determine if the solution can properly function with a central database of more than 100 million records.
Solution must find all inconsistencies from an incoming report	Unit tests are used to validate that all inconsistencies are found within by the program.

Table 4.2: Validation tests for Non-Functional requirements

4.3 Results

We simulated the operating environment using test data files that have been provided by Kingland and an AWS instance provided by our advisor. Testing has been conducted over two different configurations of the solution. One configuration was single-threading and the other used multi-threading. Figure 4.1 below displays the test data.

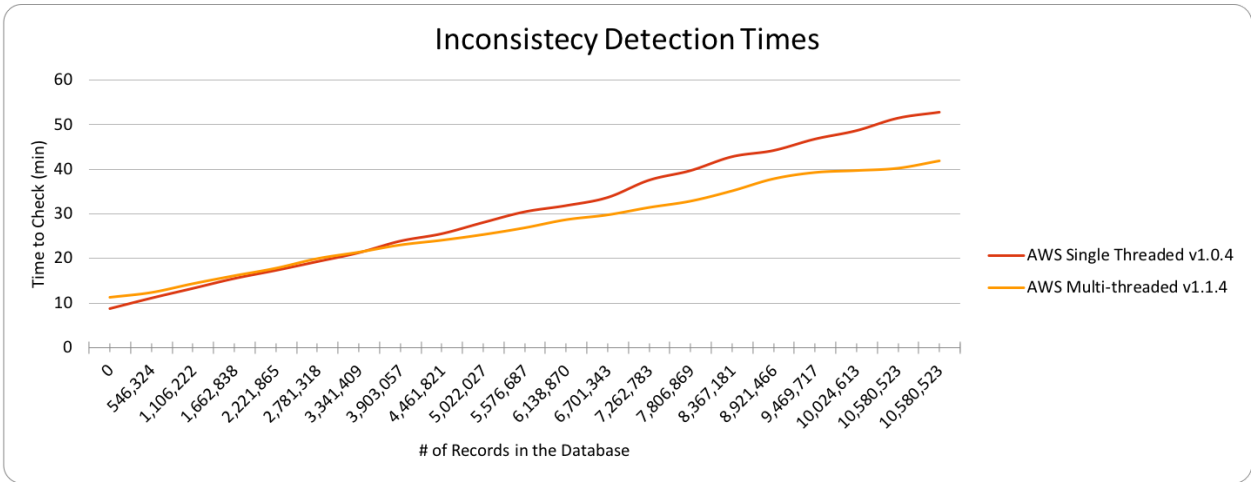

Figure 4.1: Performance data

As expected, the threaded version of the project scales better than the single threaded version. Extrapolating the data, we estimate that the project will run in under 5 hours for a central database of 100 million records. This provides a significant increase over Kingland's current performance. While it does not meet Kingland's desired performance, we anticipate further performance gains when deployed to Kingsland's AWS instance. Their instance will have more memory and more processing power that will help improve performance.

5 Appendix

5.1 Operation Manual

5.1.1 Obtaining Source Code

The easiest way to obtain the source code for the project is to download the project using git. To do this navigate to where you want to place the source code

for the project and use the git clone command. To perform this command you will need to have access to the gitlab repository.

```
$ git clone https://git.ece.iastate.edu/sd/sdmay18-27.git
```

5.1.2 Project Configuration

Before running the project two configuration files will need to be setup. The first configuration file is the Data Configuration file and will tell the project where to get the data to be tested and how to store that data. The inconsistency configuration file will tell the program which inconsistencies to check the data for.

5.1.3 Data Configuration File

The Data Configuration file needs to be saved in XML format. An example configuration file can be seen in Figure 5.1 below.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <!-- Input file location. If this is a folder, use all the files in the folder -->
    <inputPath>/Users/kingland/Desktop/0_CAIS_Test_Data.xml</inputPath>

    <!-- A list of output methods -->
    <storage-list>
        <!-- Define different ways we want to store our data -->
        <storage type="database">
            <location>//127.0.0.1:3306/Kingland</location>
            <driver>mysql</driver>
            <tableName>CentralDB</tableName>
            <username>myusername</username>
            <password>mypassword</password>
        </storage>
    </storage-list>

    <!-- A list of key keyElements we want to parse out -->
    <key-elements>
        <key-element key="NameValue" index="true"/>
        <key-element key="LEI" index="true"/>
        <key-element key="TAX_ID" index="true"/>
        <key-element key="Address" index="true"/>
        <key-element key="City" index="true"/>
        <key-element key="DateOfBirth" index="true"/>
        <key-element key="SOCIAL_SECURITY_NUMBER" index="true"/>
    </key-elements>
</configuration>
```

Figure 5.1: Example Data Configuration File

<inputpath> - The <inputpath> takes a path to a file or folder as value. If the path is a folder, we have to sort the files in the folder by last modified date.

```
Example:
        <inputpath>/path/to/file.xml</inputpath>
        <inputpath>/path/to/folder</inputpath>
```

<storage-list> - The <storage> defines a list of ways we want to store the data. The <storage-list> contains one or more <storage> elements.

- <storage>
  The <storage> defines the way to store our output data. A database type will required to have <location>, <driver>, <username> and <password> elements. While a file type will only require <location> element.

  Attributes:
  type=[database|file] required

  Internal Tags:
  - <location>
    The <location> can take a url or local path as value.
    Example:
    ```
    <location>http://localhost:3306/database</location>
    ```
  - <driver>
    This <driver> is required if the storage type is database. It should correspond to the driver type for the database being used.
    Example:
    ```
    <driver>mysql</driver>
    ```
  - <username>
    The <username> for the database account
    Example:
    ```
    <username>myusername</username>
    ```
  - <password>
    The <password> for the database account
    Example:
    ```
    <password>mypassword123</password>
    ```

<key-elements> - This defines a list of key elements we want to parse out from the row xml. The <key-elements> contains one or more <element> tags.

- <element>
  The <element> defines what elements we want to parse out from the input file. It will contain two attributes key and alias-keys. The alias-keys take in a string that contains one or more alternative key names. They are comma delimited.

  Attributes:
  key=[ssn|name|address|...] (required)
  alias-keys="key1,key2,..." (optional)

Example:

```
<element key="ssn" optional-name="IdentifierValue" index='true'>
```

5.1.4 Inconsistency Configuration File

The inconsistency configuration needs to be saved in an XML format. An example Inconsistency Configuration file can be seen in Figure 5.2 below.

```
<inconsistencies>
    <!--1XX: Record has inconsistencies with from different Reporters.-->
    <inconsistency type="same" status='0'  index='SSN|ITIN'     compare='Name'/>
    <inconsistency type="same" status='1'  index='TAX_ID'       compare='Name'/>
    <inconsistency type="same" status='2'  index='TAX_ID'       compare='LEI'/>
    <inconsistency type="same" status='3'  index='LEI'          compare='TAX_ID'/>
    <inconsistency type="same" status='4'  index='Name'         compare='SSN|ITIN'/>
    <inconsistency type="same" status='5'  index='Name'         compare='TAX_ID'/>
    <inconsistency type="same" status='6'  index='SSN|ITIN'     compare='DateOfBirth'/>
    <inconsistency type="same" status='7'  index='Name+Address' compare='SSN+ITIN'/>
    <inconsistency type="same" status='8'  index='SSN|ITIN'     compare='Name+Address'/>
    <inconsistency type="same" status='9'  index='Name+Address' compare='TAX_ID'/>
    <inconsistency type="same" status='10' index='Name+Address' compare='LEI'/>
    <inconsistency type="same" status='11' index='TAX_ID'       compare='City'/>
    <inconsistency type="same" status='12' index='LEI'          compare='City'/>

    <!--2XX: Record has inconsistencies with from different Reporters.-->
    <inconsistency type="different" status='0'  index='TAX_ID'       compare='LEI'/>
    <inconsistency type="different" status='1'  index='LEI'          compare='TAX_ID'/>
    <inconsistency type="different" status='2'  index='SSN|ITIN'     compare='Name'/>
    <inconsistency type="different" status='3'  index='TAX_ID'       compare='Name'/>
    <inconsistency type="different" status='4'  index='Name'         compare='SSN|ITIN'/>
    <inconsistency type="different" status='5'  index='Name'         compare='TAX_ID'/>
    <inconsistency type="different" status='6'  index='SSN|ITIN'     compare='DateOfBirth'/>
    <inconsistency type="different" status='7'  index='SSN|ITIN'     compare='AccountType'/>
    <inconsistency type="different" status='8'  index='Name+Address' compare='LEI'/>
    <inconsistency type="different" status='9'  index='TAX_ID'       compare='Name+Address'/>
    <inconsistency type="different" status='10' index='LEI'          compare='Name+Address'/>
    <inconsistency type="different" status='11' index='Name+Address' compare='TAX_ID'/>
    <inconsistency type="different" status='12' index='TAX_ID'       compare='AccountType'/>
    <inconsistency type="different" status='13' index='TAX_ID'       compare='City'/>
    <inconsistency type="different" status='14' index='LEI'          compare='City'/>
</inconsistencies>
```

Figure 5.2: Example Inconsistency Configuration File

<inconsistencies> - This defines a list of key elements we want to match between with the daily data and previously received records. The <inconsistencies> contains one or more <inconsistency> tags.

- <inconsistency>
  The <inconsistency> defines the inconsistency we want to check. The status defines the status code of the <inconsistency>. Two records will be inconsistent when they have the same index key(s) and different compare key(s).

Attribute:

type=[“same”|”different”] required

Is this inconsistency for records from the “same” reporter or “different reporters

status=int required

Status number of inconsistency usually 2 digits

index=”key” or "key1|key2" or “key1+key2” required

Fields to use for look up. ‘|’ symbol indicates either key 1 or key 2 match and ‘+ indicated key 1 and key 2 match

compare=”key” or "key1|key2" or “key1+key2” required

Fields to use for comparison. ‘|’ symbol indicates either key 1 or key 2 match and ‘+ indicated key 1 and key 2 match

For example:

```
<inconsistency type="different" status="2" index="SSN|ITIN",
compare="Name">
```

5.1.5 Building the project

The project can be built using Maven. To do this navigate to the the source code directory and run the following command.

```
$ mvn clean install -Dmaven.test.skip=true
```

This will compile all the java files and libraries into a .jar file called sdmay18-27-1.0-SNAPSHOT.jar saved in the targets directory inside project directory.

5.1.6 Running the project

To run the project move the jar file to the desired location and run the following command.

```
$ java -jar <jar-location>/sdmay18-27-1.0-SNAPSHOT.jar -config <data configuration file
location> -inconfile <inconsistency file location>
```

There are 4 command line options available. These options are outlined in Table 5.1 below.

Option	Purpose
-config <path-to-configuration-file>	Specify the path to a custom configuration file
-inconfile <path-to-inconsistency-file>	Specify the path to a custom inconsistency file
-inputfile <path-to-input-file>	Specify the path to an input file (should be xml format)
-loglevel [ALL | DEBUG | ERROR | INFO | OFF]	Specify the level in which the program should log

Table 5.1: Command Line Options

5.2 Alternative Designs

Initially we planned on leveraging hashing in our design. The program would hash each important datapoint and then store the hashed values in our central table. The reasoning behind this was that hashing still preserved equality checking. We eventually ended up dropping hashing from our design. We found that the hash length often ended up being more data than many of initial values. Thus, it did not save nearly as much space as we intended. Furthermore, we would be required to maintain two large databases, one to hold the hashed values and one to hold the original data. Finally, hashing would limit the ability to do inconsistency detection checks that are not based on equality, if Kingland requires these in the future.

Our initial implementation was developed without the use of threading. This design was implemented first to ensure the project was able to function correctly and detect inconsistencies properly. It was only after this version had been tested and validated that multithreading was added to the project to improve the performance.

When we first started multithreading, we created new threads per inconsistency type as well as per reporter id and records. The reporter id threads were chosen to limit multi table checks. This is because our central database stores the info of

each reporter on its own table. With this implementation our maximum number of threads was 1000. Given our AWS configuration, this ended up slowing down our program more than helping it. Threading on records introduced the problem of having to load multiple records into memory at once. This introduced a concern of having too many records in memory at once and further slowing down the program. After testing this version we opted to scrape threading on reporter id and records since there could could a very large amount of reporters and only multi-thread on inconsistency type. This decision has kept our thread count to a reasonable and beneficial level.

5.3 Related Works

One consideration for our project is previous work done in detection of inconsistencies in large data sets. In order for our solution to provide value for our client it will need to suit their needs better than other existing solutions. We have looked at a few different systems that are similar to ours. The first of which is proposed in the paper “An Efficient Method of Data Inconsistency Check for Very Large Relations.” The solution proposes the utilization of functional dependencies and applying an association finding algorithm on the data set. This solution works well with smaller number of rules and when looking for very specific types of inconsistencies. However, for our project we will have a large number of rules and we will be checking for many different types of inconsistencies both intra-record and inter-record. This would lead to many types of associations in the data set. Our proposed solution will be better than the one laid out in the paper at handling a variety of inconsistency types.

We also looked at the paper “Inconsistencies in big data” by Zhang. In this paper he discusses four types of inconsistencies. These types cover one type of inconsistency we have with missing data, however it fails to highlight inconsistency between two data sets. The paper proposes the use of a machine learning system for detecting inconsistencies. While this system is good for learning how inconsistencies are caused and working to avoid them this is not an issue Kingland needs solved. Since all of Kingland’s data is sent to it by its clients it cannot avoid inconsistencies, so strategies for this are not relevant to their problem. Another reason this solution might not be practical is that Kingland will require their data analysts to check on inconsistencies to determine the best course of action. This would further limit the abilities of any machine learning system deployed. As such our solution is more practical and better suited to Kingland’s needs for quick data detection and reporting.

Another consideration of our project is parsing large XML files very quickly. Since the daily reports of records received by Kingland are around 2 GB in size, we need to have an XML parser that can handle this. For this we looked at the article, “Conveniently Processing Large XML Files with Java”, by Haufler on parsing XML files in Java. This article highlighted the benefits of using SAX. Specifically, the consideration of memory management with an XML parser. Using a parser that loads the entire DOM into memory at once would be costly since it can often take about three times the storage of the XML file itself. Thus, SAX seems to be a better choice for processing the large XML files we will receive. According to testing done by Staveley, there is also a performance benefit in terms of time when using SAX on large files compared to other Java XML parsers. This provides further justification for the use of the SAX parser in our project.

5.4 Figures and Tables

5.4.1 List of Figures

Figure Number	Figure Description
Figure 2.1	Block diagram of high-level system architecture.
Figure 3.1	Example XML input tags
Figure 3.2	Example XML storage tags
Figure 3.3	Example XML element tag
Figure 3.4	Example XML inconsistency tags
Figure 3.5	Class Diagram
Figure 4.1	Performance data
Figure 5.1	Example Data Configuration File
Figure 5.2	Example Inconsistency Configuration File

5.4.2 List of Tables

5.5 References

Haufler, Andreas. “Conveniently Processing Large XML Files with Java.” *Dzone.com*, 10 Jan. 2012, dzone.com/articles/conveniently-processing-large.

This article explains how using SAX as an alternative to a DOM parser reduces the amount of memory needed to handle large files because SAX invokes callbacks to detect XML tokens instead of loading the entire file into memory. It also shows example uses of the SAX parser.

Murnane, Tafline. “ISO/IEC/IEEE 29119 Software Testing.” *ISO/IEC/IEEE 29119 Software Testing Standard*, softwaretestingstandard.org, 24 Oct. 2013, www.softwaretestingstandard.org/part4.php.

ISO/IEC/IEEE standards for specification-based, structure-based and experience-based testing techniques. Along with outlining a variety of testing technique this document outlines some standard methods used to derive test cases that have been approved and adopted as international standards for software testing.

Smrcka, Ales I., Ph.D. "TEST PLAN OUTLINE (IEEE 829 Format)." *IEEE 829 - Standard for Test Documentation Overview*. Brno University of Technology, n.d. Web.

An outline for a test plan document that pinpoints important areas to address when creating a test plan. The template also provides a

description of each area so that the reader knows exactly what purpose that area of the test plan addresses.

Staveley, Alex. “JAXB, SAX, DOM Performance.” *Dzone.com*, 31 Dec. 2011, dzone.com/articles/jaxb-sax-dom-performance.

This article analyzes a performance evaluation of a JAXB, SAX and DOM XML parser. It provides the source code the tester used to perform the tests as well as the various times required for each xml parser with various sizes of data. For the largest data set, the SAX parser performed the parsing operations in the least amount of time, and has a much smaller memory usage than that of a DOM parser. The drawback to the SAX parser was the amount of developer attention needed to calibrate the parser to perform the desired operations.

Sug, Hyontai. "An Efficient Method of Data Inconsistency Check for Very Large Relations." S International Journal of Computer Science and Network Security 7.10 (2007): 166-69. Web. 22 Sept. 2017.

This defines a strategy to help detect inconsistencies in databases based on functional dependencies between attributes in a relation and apply an association rule algorithm based on the attribute sets. It is also assumed that the database under observation is not designed with much consideration about normalization.

Zhang, Du. (2013). Inconsistencies in big data. Proceedings of the 12th IEEE International Conference on Cognitive Informatics and Cognitive Computing, ICCI*CC 2013. 61-67. 10.1109/ICCI-CC.2013.6622226.

This article examines four types of inconsistencies in big data - temporal, spatial, text and functional inconsistencies - and how categorizing the inconsistencies in data can help to improve the quality of big data analysis.