Data Analytic Tools for Inconsistency Detection in Large Data Sets

Sdmay18-27

http://sdmay18-27.sd.ece.iastate.edu/

Advisor - Dr. Ying Cai

Client - Kingland Systems

Team Members

- Logan Heitz
  - Project Lead
  - Senior in Software Engineering
- Christopher Konopka
  - Communication Lead
  - Senior in Software Engineering
- TJ Rogers
  - Quality Lead
  - Senior in Software Engineering
- Camden Voigt
  - Technical Lead
  - Senior in Software Engineering

Project Plan

Problem Statement

- Kingland Systems performs inconsistency detection on large data sets
    - An inconsistency arises when two records should match, but don’t
- Current solution takes a lot of time and resources
- Approach

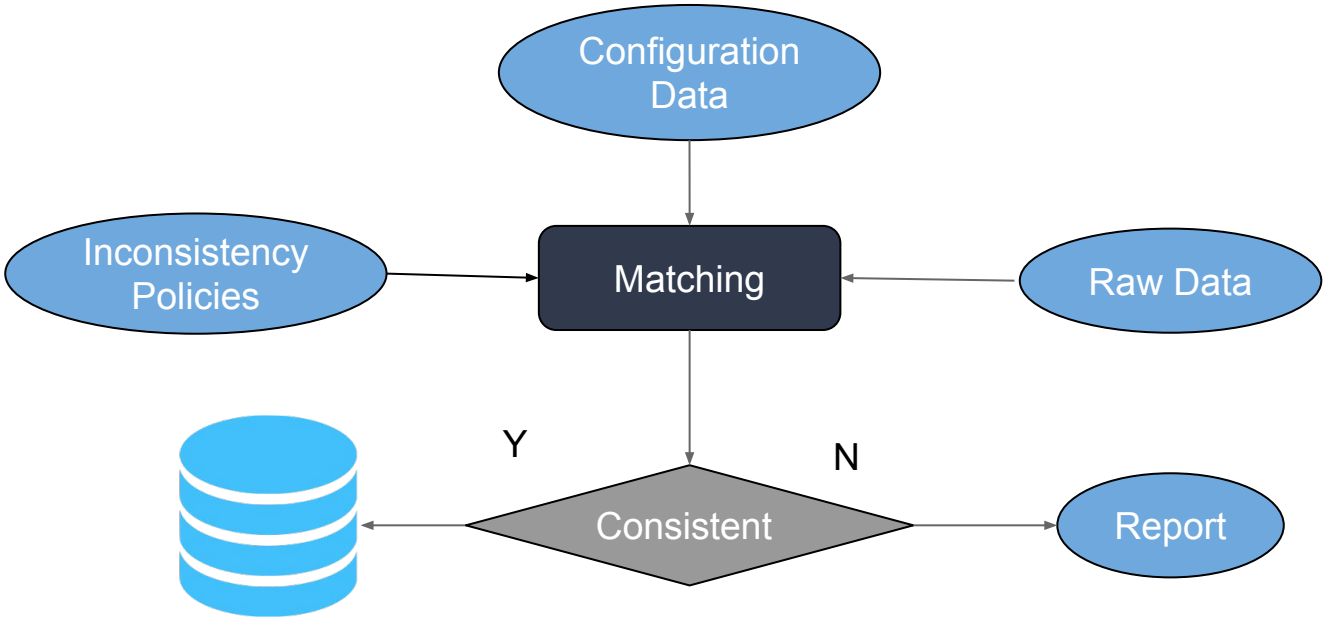
Functional Requirements

- Doesn’t use SQL inner-join statements
- Utilize only relevant information
- Compare records to previous records and current records
- Validate that all necessary fields are present in the data
- Handle various forms of input
- Update database after analysis

Non-functional Requirements

- Solution must perform inconsistency check in less than 1 hour for daily reports
- Solution must be able to analyze 100 million or more records at a time
- Solution must run on Kingland’s system
- Solution reports less than 5% false positives

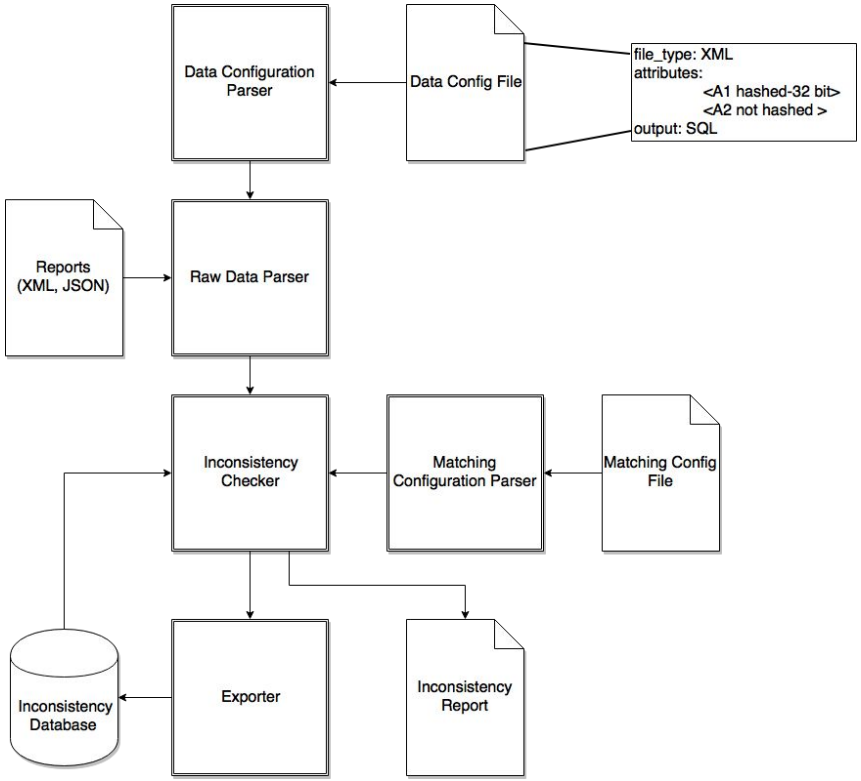
Conceptual Sketch



System Description

- Reduce the data size
  - Parse the key information out from XML file to reduce the data size.
  - Use hashing function to reduce size of some data.
- Check the consistency of each record in the report.
- Modular design to easily support comparison of different DBMS.

System Block Diagram



Operating Environment

- Amazon Web Services
    - Integrate into Kingland's existing infrastructure
    - Integrate with any additional third party solutions
- Operate with minimal user input

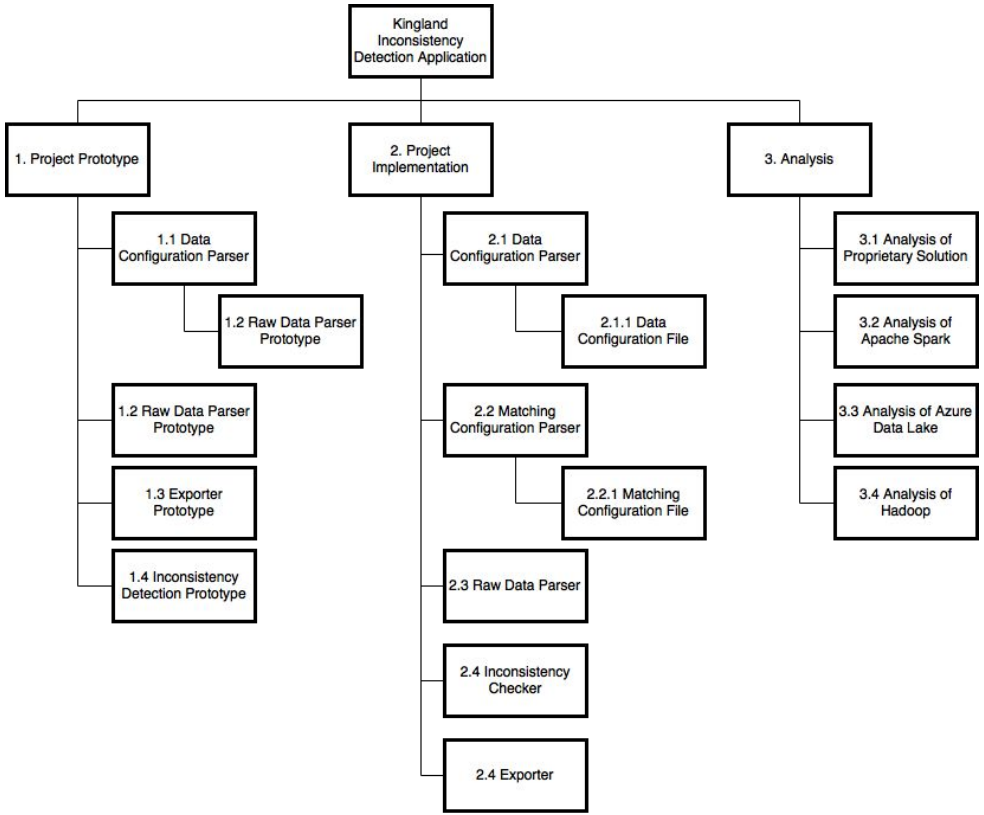
Market Survey

- Two papers on the subject
  - “An Efficient Method of Data Inconsistency Check for Very Large Relations.”
    - Functional dependencies
    - Works well with smaller number of rules and very specific types
  - “Inconsistencies in big data”
    - Learning how inconsistencies are caused

Deliverables

1. System architecture of proprietary solution
   - Delivery Date: 01/20/2018
2. System implementation of proprietary solution
   - Delivery Date: 02/20/2018
3. Analysis of industry solutions
   - Delivery Date: 03/20/2018
4. Test cases of solutions
   - Delivery Date: 04/02/2018
5. User manual and final product
   - Delivery Date: 04/02/2018

Work Breakdown Structure



Resource Requirements

- Test data
- Test Machine
- Deployment Server

First Semester Schedule

Deliverable	Description	Start Date	Due Date
Project Plan V1	Initial draft of the project plan	09/15/2017	09/24/2017
Team Website V1	Initial version of the team website	09/15/2017	09/24/2017
Project Prototype	Prototype version of the application	09/27/2017	12/01/2017
Design Document V1	Initial version of the design document	10/06/2017	10/13/2017
Project Plan V2	Revised project plan	10/20/2017	10/27/2017
Final Project Plan	Final version of the project plan	11/27/2017	12/01/2017
Design Document V2	Revised Design Document	11/27/2017	12/04/2017

Second Semester Schedule

Deliverable	Description	Start Date	Due Date
Implementation	Implementation of proprietary solution	01/08/2018	02/22/2018
Analysis	Analysis of proprietary solution against industry standard solutions	02/23/2018	03/23/2018
User Manual	User manual for proprietary solution	03/23/2018	04/06/2018
Final Report	Final report of project outcomes and analysis	03/23/2018	04/20/2018

Risks

- Miscommunication with Kingland
- Lack of Big Data Knowledge
- Shortage of Time

System Design

System Requirements

- Solution must run on 100 million records in less than hour
- Solution must reduce the memory usage by 75%
- Solution must find all inconsistencies
- Less than 5% of inconsistencies found should be false positives

Functional Decomposition

- Data Configuration Parser
- Raw Data Parser
- Matching Configuration Parser
- Exporter
- Inconsistency Checker

Detailed Design - Configuration Parsers

- Data Configuration Parser
  - Parses the data configuration file
  - Configures
    - Raw Data Location
    - Output Location
    - Raw Data Format
- Matching Configuration Parser
  - Parses the matching configuration file
  - Configures which fields will be used for matching

Detailed Design - Raw Data Parser

- Reads raw input file and extracts desired elements
    - Parses a given file, or all files in a given directory

```
<cat:Account>
    <cat:FirmDesignatedID>120458269</cat:FirmDesignatedID>
    <cat:AccountType>CORPORATE</cat:AccountType>
    <cat:AccountStatus>ACTIVE</cat:AccountStatus>
    <cat:AccountOpened>1998-07-22</cat:AccountOpened>
    <cat:AccountEffective>1998-07-22</cat:AccountEffective>
    <cat:Identifier>
        <cat:IdentifierType>PRIME_BROKER_ID</cat:IdentifierType>
        <cat:IdentifierValue>3201</cat:IdentifierValue>
    </cat:Identifier>
    <cat:LegalEntity>
        <cat:FirmDesignatedCustomerID>45711549963251028541</cat:FirmDesignatedCustomerID>
        <cat:RoleOnAccount>HOLDER</cat:RoleOnAccount>
        <cat:BranchLocationIndicator></cat:BranchLocationIndicator>
        <cat:Name>
            <cat:NameType>LEGAL</cat:NameType>
            <cat:NameValue>Limb 2</cat:NameValue>
        </cat:Name>
```

Detailed Design - Exporter

- Configurable via the Data Configuration File
- Exports records into database
  - Exports consistent data immediately
  - Exports inconsistent records once resolved

Detailed Design - Inconsistency Checker

- Called each time a line is parsed from the raw data
- Performs queries to check if a tuple is consistent with a central database
  - Based on hashed value equality checks
- Returns 0 if consistent, otherwise the inconsistency number
- If consistent, newline is added to database
  - Otherwise, export to a report

System Analysis

- Strengths
  - Solution is modular
  - Easy configuration
- Weaknesses
  - Duplication of data
  - False positives

Test Plan

- Test Driven Development
  - Mockito
  - JUnit test suite
- Defect Reports as Issues on GitLab
- Continuous Integration

Prototype Implementation/Basic Building Blocks

- Configuration file accepts several fields
- Raw Data Parser collects fields as key-value pairs
  - TAX_ID, SOCIAL_SECURITY_NUMBER, LEI, NAME
- Inconsistency Check only does one type of inconsistency
- Exporter

Prototype Performance

With 2GB XML data

- Parse and detection takes 3 minutes
  - Parsing on its own is under 40 seconds
  - SQL upload is time intensive
- Size of the data parsed out is around 100MB (more than 90% size reduced)

Project Tools

- Current tools
  - Java
  - SQL
  - SAX (Simple API for XML)
- Future tools
  - Amazon Web Services
  - Apache Spark
  - Azure Data Lake
  - Apache Hadoop

Conclusion

Current Status

- Project prototype implementation finished
- Final project plan completed
    - Second semester schedule set
- Design document version 2 completed

Next Steps

- Amazon Web Services
- Implementation of proprietary solution
- Analysis of Solution
- Final Recommendations for Kingland

Questions?