Data Analytic Tools for Inconsistency Detection in Large Data Sets

Project Plan

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: September 21, 2017 Version 1

Table of Contents

1. Introduction

1.1 Project Statement

Kingland processes a large amount of data that is gathered from its clients everyday. This data can be relating to consumers, agreements between consumers, or products offered. This data is compared to Kingland's central database in order to detect inconsistencies and then added to said database. The database contains over 100 million records, and around 10% of these records are updated daily. Due to its size, this comparison takes several hours to run every day. This time stems from the fact the entire database cannot be loaded into memory at one time and an inefficient use of SQL join statements to check for inconsistencies. Kingland would like to process 100 million records for inconsistencies in an hour or less. Additionally this detection must begin with the latest version of the central database after the reports come in.

1.2 Purpose

The purpose of our project is to provide Kingland Systems with a significantly faster solution of detecting and flagging data inconsistencies between daily reports and their central database. This will be executed before May of 2018.

1.3 Goals

- Develop a proprietary data inconsistency detection solution which enables Kingland to process over 100 million records in one hour
- Have our solution be able to process upwards of 80 different types of inconsistencies of varying severities.
- Always be able to detect inconsistencies of specified types when they are present
- Notify technicians of present inconsistencies in a clear and productive manner.
- Evaluate at least three third party solutions to data inconsistency detection and compare their speeds with a proprietary solution
- Have our final solution be planned to be integrated into Kingsland's day to day operations after May 2018.

2. Deliverables

- System architecture and algorithm design of proprietary solution
- System implementation of proprietary solution
- Analysis of proprietary solution against industry standard solutions
- Test cases of solutions

- User manual

3. Design

3.1 Previous Work/Literature

Article on Inconsistency detection in overlapping reports.

Lee, Pei-Ju, et al. "Inconsistency Detection and Data Fusion in USAR Task." Engineering Computations, vol. 34, no. 1, Jan. 2017, pp. 18-32. EBSCOhost, doi:10.1108/EC-11-2015-0339.

Article regarding hashing when applied to large data sets

Petroni, Nick L., Jr., Timothy Fraser, Aaron Walters, and William A. Arbuagh. "An Architecture for Specification-Based Detection of Semantic Integrity Violations in Kernel Dynamic Data." *USENIX-SS'06 Proceedings of the 15th Conference on USENIX Security Symposium* 15 (2006): 289-304. Web.

An article outlining an experiment which successfully analysed a large data set

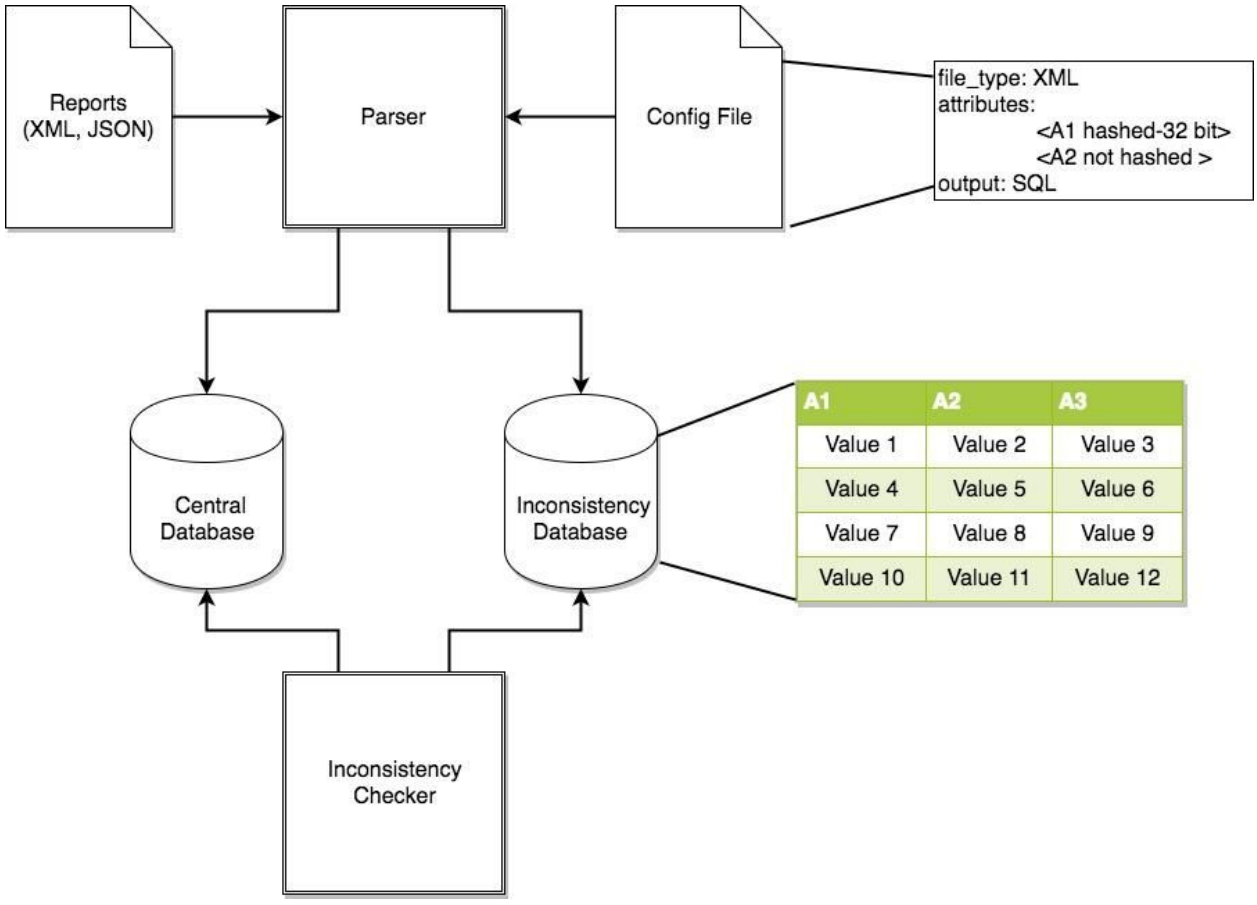
Sug, Hyontai. "An Efficient Method of Data Inconsistency Check for Very Large Relations." S International Journal of Computer Science and Network Security 7.10 (2007): 166-69. Web. 22 Sept. 2017. <http://paper.ijcsns.org/07_book/200710/20071022.pdf>.

This proposed solution uses confidence rules and functional dependencies to detect inconsistencies very quickly. Our project would not be an ideal application of their solution for two reasons. First, the data we are working with has many more types than the solution proposed. Second, it would not solve the problem of having to swap parts of the table out of memory.

3.2 Proposed Solution

Our proposed solution to this problem is to use either hashing or a binary file format to make the comparison checks faster than before. Hashing would work because we only need to do equality checks between the different values. This means we can use hashing to turn all values into unique integers which make lookup faster and easier in a SQL table, and also makes our equality comparisons faster. A binary file format could significantly reduce the amount of space that the data is taking. It could also mean faster look ups as we could optimize our search based on the format.

In terms of implementation both hashing and the binary file solution will almost the same. First, we will create a parser which is a program that accept new data sent to Kingland in either XML, JSON, or some other format. The parser will run through this data and convert it into either a database with hashed values or a binary file. It will only convert the fields that are specified in the configuration file. The parser may also update the central database which holds all records in their full state. Next, we will run an inconsistency checker which goes through either the inconsistency database or the binary file and recognizes conflicts. Once a conflict has been found the inconsistency checker will go to the central database and update the correct records with an error code.



3.3 Assessment of Proposed Methods

The proposed solution laid out above solves the main problem of finding and marking inconsistencies in the data faster than could previously be done. It comes with a few tradeoffs including having to duplicate data into another database and the possibility of not being able to detect every type of inter-record inconsistency. Both of these shortcomings are small issues. While usually duplicating data isn't a great solution, in

this case the duplicated data will actually take up less space than the original and only needs to updated as often as the original data. Also, not being able to solve every inconsistency isn't a huge issue as even if we can only solve a large amount of these issues it would still reduce the time needed to run a inconsistency scan significantly.

The biggest shortcoming of our proposed solution would be that a third party solution may be able to do this job almost as well. In this case it would almost 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.

3.4 Validation and Acceptance

Requirement	Validation/Acceptance test
Doesn't use SQL inner-join statements	Style Checker
Utilize only relevant information	Analysis database is smaller than central database
Compare against previous records as well as other current records.	Use an Oracle
Validate all fields are present in data	Use an Oracle
Handle various forms of input	Configuration tests
Update central database after analysis	Check central database after a given set of input
Perform faster than current system (3-5 hours)	Run side-by-side with existing system
Analyze 10 million or more records at a time	Load/Stress test

4. Project Requirements/Specifications

4.1 Functional

- Doesn't use SQL inner-join statements.
- Utilize only relevant information.
- Compare current records to previous records as well as other current records
- Validate all fields are present in data

- Handles various forms of input
- Update central database after analysis

4.2 Non-functional

- Perform faster than current system (3-5 hours).
- Analyze 10 million or more records at a time

5. Challenges

5.1 Feasibility

This project is very feasible because every member of our team has been exposed to the various components of our solution, such as hashing and SQL. The amount of work we foresee ourselves doing is very manageable, if not less than we would desire.

5.2 Cost Estimate

Kingland has given us a budget of $500.00 for this project. We don't expect to have any expenditures for this project as it will be entirely software based, we're not receiving payment for our work and each member has the necessary equipment at their disposal to complete this project. We anticipate working on this project approximately six to ten hours each week for the entirety of at least one semester.

6. Timeline

6.1 First Semester

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
Config File Prototype	Prototype of configuration file for report parser	09/27/2017	10/06/2017

Design Document V1	Initial version of the design document	10/06/2017	10/13/2017
Parser Prototype	Prototype of parser to transfer records from reports to database	10/06/2017	10/23/2017
Inconsistency Detection Prototype	Prototype of inconsistency detection	10/06/2017	10/23/2017
Project Plan V2	Revised project plan	09/25/2017	10/27/2017
Design Document V2	Revised Design Document	10/07/2017	12/08/2017
Final Project Plan	Final version of the project plan	10/28/2017	12/01/2017

6.2 Second Semester

Deliverable	Description	Start Date	Due Date
Implementation	Implementation of proprietary solution	01/08/2018	02/20/2017
Analysis	Analysis of proprietary solution against industry standard solutions	02/20/2017	03/20/2017
User Manual	User manual for proprietary solution	03/20/2017	04/02/2017
Final Report	Final report of project outcomes and analysis	04/02/2017	04/20/2017