Data Manipulation: Converting XER Files for Analysis

CS4991 Capstone Report, 2024

Vlad Tarashansky
Computer Science
The University of Virginia
School of Engineering and Applied Science
Charlottesville, Virginia USA
vladtarashansky@gmail.com

ABSTRACT
Kroll Construction Expert Services needed to analyze large-scale construction scheduling data in a faster way than navigating the scheduling software to settle legal disputes. To present this data in a simple spreadsheet, Kroll Government Solutions (KGS) created a pipeline of python and SQL scripts to parse XER files used by the common scheduling software, Primavera, which was extremely inefficient. My goal was to increase efficiency, reliability, and create new features. Utilizing my knowledge of memory-efficient data structures, I was able to speed up the process. I utilized increased error logging and debugging to aid in correcting wrongly formatted input files, and also created various new features in the final output table. The Expert Services team's efficiency increased, leading to more cases, clients, and profit. The system still has room for more improvements such as automating the reformatting of faulty inputs and the running of consecutive steps. Eventually, this system could be integrated into a standalone tool that converts and creates reports on any construction scheduling dataset, eliminating the need for manual running and troubleshooting each time.

1. INTRODUCTION

The question is not if, but for how long, a large construction project will get delayed. Then the problem is why and who should pay for it. Was the delay due to weather conditions? Poor planning? Mistakes in construction? Kroll Construction Expert Services helps legal teams settle such large construction disputes. A major part of their work is analyzing and reporting construction schedules to legal teams. Many construction projects utilize a scheduling software, Primavera P6. This software exports data in proprietary .XER file format. This format is not easily analyzable, so Expert Services would open and view the data inside Primavera, which proved to be tedious and inefficient.

KGS had a working pipeline to parse and reformat the .XER files. The system was slow and required manual checking of each step. My goal was to improve and streamline this process. A more efficient process would empower the legal team to focus on analysis over data reformatting and serve more cases. The task proved difficult because I had no knowledge of the .XER file format, Primavera software, structure of pipeline, and the creator of the pipeline was no longer with the company.

2. RELATED WORKS

A page in O'Reilly's Python Cookbook influenced the simplest and most significant improvement in the pipeline. Blissett discussed the difference between write() and write_lines() in the Python Cookbook [1]. This key difference influenced a change that made the pipeline run in minutes rather than hours.

Djouallah offered a different approach to viewing and analyzing .XER files, importing the files directly into Excel using a special macro [2]. This approach would work if the goal were to edit and reconstruct the .XER file. However, my work was to deconstruct, reformat, and display into one final spreadsheet. Using Excel directly would make joining the tables difficult. Also, Djouallah did not have a method for making the calendar table data readable. The calendar table contains essential information about the time and amount of work done on the projects.

A forum on Winter's planningplanet.com discussed calendar data formatting. The forum specifically discussed how to convert the numbers within the data to dates. The numbers were from Microsoft's date function that begins enumerating dates from 12/30/1899 [4]. The forum also supplied insight on the formatting of the calendar data. Eagle [3] was trying a similar process of reading the calendar data in SQL.

3. PROJECT DESIGN

The Expert Services team employed the KGS team to create a pipeline to extract data from the .XER files into a comma-separated values (csv) format. Expert Services would then visualize and analyze the file in a Power BI dashboard. The pipeline already existed when I began my work at KGS. It ran for multiple hours and required extensive manual checks to see that it successfully parsed all the files. I improved the efficiency of this pipeline.

3.1 XER File Format

Primavera's .XER files serve as the database that the software uses. The file is tab delimited and has multiple tables with their respective data. The first row contains metadata about the project and when the file was created. The rest of the file contains tables. The first column has four attributes that specify what that row has. These attributes are:

- %T: Table
- %F: Fields
- %R: Record
- %E: Last row in the XER file

The structure of each table is a table name row, %T, followed by a row with the fields of that table, %F, then records in that table, %R's. A table ends when the next row begins with the next table's name, %T. A %E row indicates the XER is over.

3.2 Existing Pipeline

Projects received by KGS had multiple XER files, each capturing a snapshot of the construction schedule at a certain time. The flow of the pipeline is as follows:

- Parse received XER files.
- Upload data to Oracle database
- Select, join, reformat, and create meaningful fields.
- Export the final view as a deliverable CSV file.

The pipeline did this in a 6-step process. Step 1 parses the .XER files. Step 2 combines and uploads all tables with the same name from different snapshots to an Oracle database. Step 3 creates a template calendar table. Step 4 populates the new calendar table with expanded data from the original calendar table. Step 5 cleans and standardizes the calendar table. Step 6 extracts and combines data from all the tables into one final view. Steps 1,2, and 4 use python. Steps 3,5, and 6 use SQL scripts.

3.2 Efficiency Problems

My first introduction to the process was a zip folder with the code, a dataset, and short instructions for running the process. After troubleshooting, I successfully ran the process. Step 1 took approximately 24 hours, and the rest of the steps took about 3 hours combined. The pipeline was a black box at this point, so I began to understand the process. After learning the information in section 3.1 and 3.2,

I was ready to begin finding inefficiencies within the process. Because step 1 was the slowest, I focused on it.

Step 1 is a multithreaded process that receives a folder with XER files and outputs a folder of text files. Groups of input files run concurrently on separate threads. The code reads each .XER file line by line to extract tables into tab delimited text files. It creates a table name from the %T row, then appends the %F row and all subsequent %R rows without their first column to a variable. When it reads a new %T row the variable gets written to a new text file in the output folder.

The code appeared fine. Why would this take a full day to run fifty input files? At first, I tried changing the number of threads, but this made no difference in run time. After careful consideration of the underlying mechanism of string variables, the appending of rows to a single string was clearly the issue. String concatenation requires consecutive memory. When a table has over a million rows, the process was looking for consecutive memory to store all those rows in a single string. This was the root cause of the slowdown.

The write_lines() function was my attempted solution to the problem. The string variable that represented the rows in each table was changed to an array where each element was a row. Then I used the write_lines() function to write the array to a new text file. After these changes were made, the process was rerun on the initial dataset. Step 1 ran in approximately 30 seconds.

3.3 Data Issues

When the next construction dispute came in, I oversaw the running of the process. I ran it with my new and improved code, however, step one did not successfully parse each input file. The issue was that the files held carriage returns. The affected files would have a row without a %T, %R, %F, or %E which would cause that file to prematurely terminate parsing. There was no error logging system. To fix these issues, a person would need to read hundreds of output files to see which input stopped early, then search that XER file for carriage returns and manually fix it. The person running the process would have to extensively manually check each run to ensure it parsed all the files.

My fix was to implement an error logging feature to say which line in which file caused the code to stop early. I stored these logs in the Oracle database. I created two tables: a job log and a job description log. The job log table had the following columns:

- JOB_ID
- PROJECT_NAME
- NUM_FILES
- NUM_FILES_COMPLETED
- TIME_STAMP
- USERNAME (Database login username)
- STATUS (Completed or failed)
- RUNTIME (Seconds)

The job description table had information on each XER file and contained the columns:

- JOB_ID
- JOB_TYPE
- FILE_NAME
- COMPLETE (Y or N)
- NUM_TABLES_COMPLETE
- RUNTIME (Seconds)
- EXCEP (Error message)

The error logging feature collects and uploads data into oracle at the end of parsing each file and the end of the entire job. The error message being logged would say which table and which line contained a carriage return. This removed manual checking but not manual fixing of the input files. It also created a backlog to see when, who, and how long previous jobs took.

4. RESULTS

The construction .XER pipeline is still far from fully polished, but the results of this work were substantial. The initial runs of the pipeline took over 24 hours just to run the first step. The first step then needed tedious manual checking for correctness and completeness. The rest of the process only took about 3 hours. The improvements made to step one drastically reduced the total time to run the pipeline. Step one now runs on the same data in approximately 30 seconds. This is over a 280,000% increase in efficiency. On top of this enhancement, the manual checking of a complete parse at step one was now automatic. Error logging allowed the user to immediately see if a parse job was complete or where it encountered issues.

5. CONCLUSION

The construction parsing pipeline is a tool used by the Kroll Government Solutions team to do work for the Kroll Construction Expert Services Construction Disputes team. As a result of the enhancements to the pipeline, the turnaround time for parsing projects was reduced from several days to several hours. With this improved efficiency, the Expert Services team had a higher throughput of cases. Their clients would receive construction schedule reports much quicker and have more time to analyze them.

The enhancements also provided value for the Kroll Government Solutions team running the pipeline. With improved error logging, less manual quality checks were required. Also, there was less downtime between checking the output and rerunning the parser to get the next output because of the increased efficiency in step 1. That means time that would be spent waiting for code to run could be actively spent quality checking, extracting new features, and further improving the pipeline.

6. FUTURE WORK

The XER parsing pipeline still requires manual correction of faulty input data. Future work might include automating the correction of inputs and handling of errors within the pipeline. The XER parsing project is one step in the analysis and representation of construction scheduling data. The pipeline could be built into a greater application that takes in .XER files and automatically creates reports, charts, graphs, and draws conclusions about delays in the schedules. We could sell this app/service to the Construction Expert Services team, legal teams, or directly to construction companies to find delays prior to having disputes with their clients. Such an application could also take advantage of collecting data from its users. It could learn common threads of a given company's delays and give recommendations for improvements.

REFERENCES

[1] Blissett, L. (n.d) Writing to a File. O'Reilly Media. https://www.oreilly.com/library/view/python-cookbook/0596001673/ch04s03.html#:~:text=Calling%20writelines%20is%20much%20faster,write%20repeatedly%20in%20a%20loop

[2] Djouallah, M. (2021, Oct.) Understanding Primavera XER Files. Plan Academy Inc. https://www.planacademy.com/understanding-primavera-xer-files/#:~:text=The%20XER%20file%20is%20one,that%20read%20a%20CVS%20file

[3] Eagle, D. (2014, July*) Hi, I have just figured it out too, it took me days. I am working in SQL [*Online forum post] Planning Planet. http://www.planningplanet.com/forums/primavera-version-pm5-pm6/413006/p3ec-export-xer-file-calendar-data-issue

[4] Winter, R. (2005, June) *Those 5 digit numbers are day numbers, beginning with 12/30/1899…* [Online forum post] Planning

Planet.
http://www.planningplanet.com/forums/primavera-version-pm5-pm6/413006/p3ec-export-xer-file-calendar-data-issue