Barry Yung, Nick Simmons

Professor Vesely

CS4500

7 Feburary 2020

Data Adapter Performance Testing

Introduction

To choose a proper data adapter for our company, performance testing must be done to decide on the ideal implementation. Performance, in our case, is determined by code readability, runtime, and adaptability. For this testing, we have chosen six different implementations in different languages (CwC and Python). All CwC implementations will be build with C++ 11 and all Python implementations will be run with Python3.

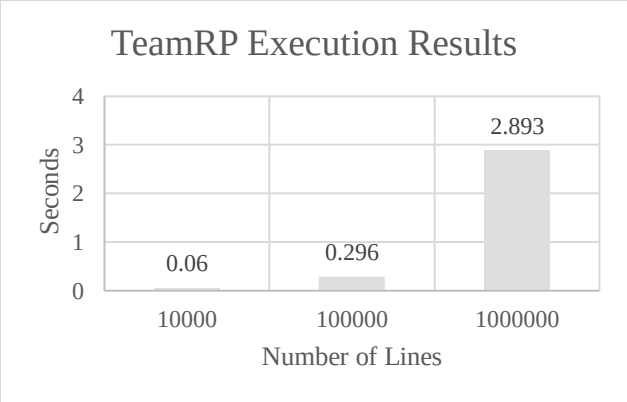
Description of the Analysis Performed

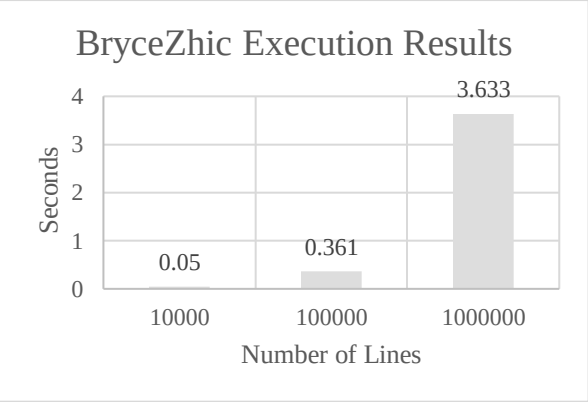
The benchmark that we will be using to determine the ideal data adapter is the execution time to parse large SoR files. We find this to be one of the most important statistics because we do not want bottleneck issues from a parser. To find the execution time, we will be using the Linux command “time” and running each implementation on a .sor file with 10000, 100000, and 1000000 lines of text. Each implementation will be built once and executed 10 times across all sets and the execution time for each run will be recorded. We will also be looking at the code quality by analyzing the reusability of code and the adaptability to our current project. All these benchmarks will be tested on a MacBook Pro 2018 running macOS Mojave. The system uses a 2.7 GHz i7 Processor with 6 cores and has 16 GB of 2400MHz DDR4 RAM.

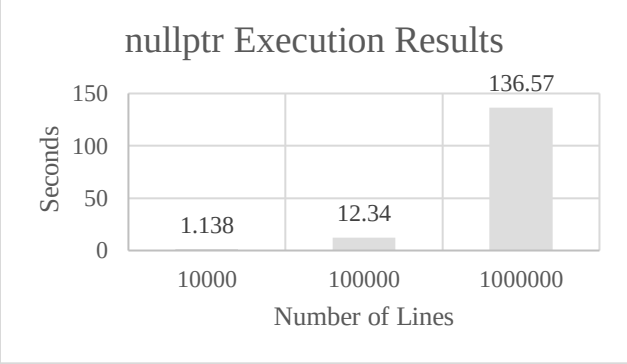
Comparison of the analysis performed

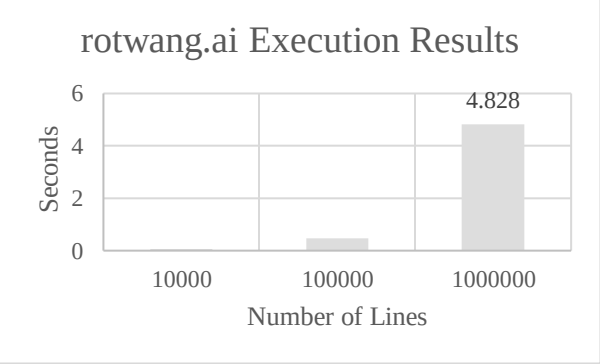
The implementations of data adapters and the results for each implementation can be found in the tables below. Each implementation was ran 10 times across each of the different data set sizes. The result shown is the average execution from the 10 executions. Based on only execution time, the best performer was the implementation from "TheSegFaultInOurStars." In order of performance – 1: TheSegFaultInOurStars, 2: Team RP, 3: BryceZhic, 4: rotwang.ai, 5: Team Baobab, 6: nullptr.

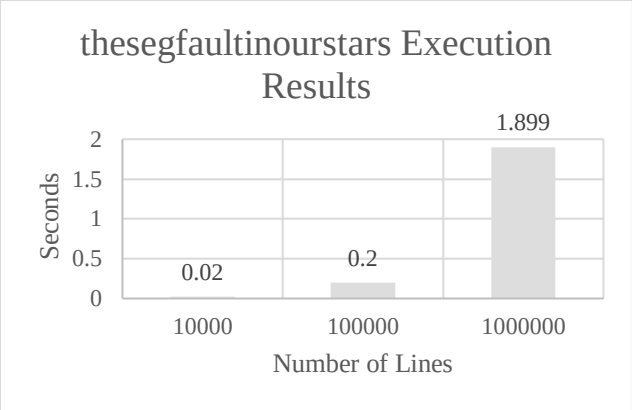
Team Name	Github Link
TeamRP	https://github.com/canonsawrey/A1P1-TeamRP
BryceZhic	https://github.com/ZhichaoC/A3Prize/tree/master/submission/part1
nullptr	https://github.com/csstransky/cs4500-assignment-1
the segfault in our stars	https://github.ccs.neu.edu/xiaju/sorer.git
Team Baobab	https://github.com/oriasm/swdev-assignment1
rotwang.ai	https://github.ccs.neu.edu/seannolly/swd_assignment_3.git

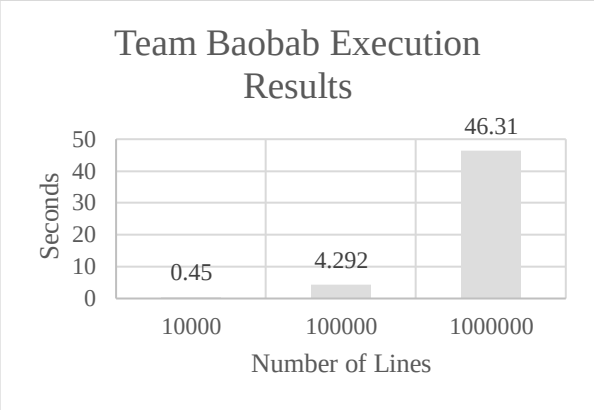












The nondeterministic factor can be attributed to the language that the data adapter was written in. Team Baobab wrote their adapter in python which has a large impact on the performance when parsing. We believe this can be attributed to the global interpreter lock that python has. Also python is interpreted while C++ is complied. Nullptr performed poorly in these tests because of the method they used to parse through the actual file. We believe that the number of objects that are created when they create their "sor" data frame hinder the runtime. These results show that C++ is the preferable language for efficiency, in this case.

Threats to validity

These results may differ user to user depending on the system that each implementation is run on. This difference can come from a different CPU with more or less cores and more or less memory in each system. Furthermore, other users may have different results depending on the test file they used. Our test file was 6 columns by 1 million rows.

Recommendation to management

After analyzing each implementation, we have decided to recommend TeamRP's implementation. The first factor in our decision is that it is written in CwC which will adapt

seamlessly with the application we will be building. Furthermore, the code is well commented and does not have much repetition. We chose this implementation over the best performing one because their sor class is much more built out and has more functionality. Furthermore, the parsing with TeamRP's is much more efficient.

The order of preference for all the implementations is TeamRP, TheSegFaultInOurStars, BryceZhic, rotwang.ai, TeamBaobab, and nullptr.