Parallel Workflow Support for StarVZ using Drake

Guilherme Rezende Alles, Lucas Mello Schnorr
Graduate Program in Computer Science (PPGC/UFRGS), Porto Alegre, Brazil

***Abstract*—This paper presents an approach to programming data analysis workflows using Directed Acyclic Graphs (DAGs) as the mean of separating computation stages. In this work, we analyze the programming structure of StarVZ – a framework written in the R programming language for data visualization - and rearrange it in order to form a graph that represents the data manipulation involved in the processing stage of the data analysis pipeline. The creation of the DAG is done with the help of Drake, an R package for managing workflows and usually applied in data science contexts. Our objective with this work is to increase the amount of data that can be analyzed with StarVZ, and we approach this problem by leveraging Drake features that can potentially speed-up a data analysis workflow, such as the parallel execution of independent tasks and the caching of results of previous computationally intensive steps (also known as memoizing). Even though we were able to explore these features, the results obtained are not satisfactory because of the way Drake implements communication between parallel processes.**

I. Introduction

Data science workflows are often referred to as a series of steps that can be represented as a pipeline. The most common steps in this pipeline are reading, manipulating (tidying), visualizing, and analyzing data, and data manipulation frameworks such as the Tidyverse [1], for the R programming language, enforce this pattern with its APIs. During a typical data science workflow, this pipeline is executed many times with incremental changes to the process (especially in the manipulation step), and this iterative development is important because it allows for the scientist to improve the quality of its work based on previous observations.

Despite its relevance, the process of constantly iterating over the same set of steps can add up to a considerable productivity overhead. The data analysis pipeline can take anywhere from a few seconds to several minutes (or even hours) to complete, depending on the complexity and the amount of data that it involves. Additionally, because these pipeline operations often change in small increments, most of the computation time is wasted executing redundant code that was exactly the same as the previous iteration.

As an alternative for data science pipelines, this paper's objective is to present the advantages of using a directed acyclic graph (DAG) to represent a data science workflow. We use a workflow manager, Drake [2], to create such graphs for a data analysis framework called StarVZ [3]. We also present how the visualization of the graph and features like parallelism support and memoization have the potential to both increase the understanding of the data flow in a data analysis process and also speed up the execution of these workflows. Additionally, by leveraging such features, we intend to tackle the problem of execution time on StarVZ's data analysis process [3], which would eventually allow for larger data sets to be studied in a reasonable time.

II. Background

A. Drake

Drake is a library written for the R programming language that strives for managing workflows in the form of directed acyclic graphs of tasks. Despite being used in the context of data science, drake can be used as a general purpose workflow manager.

Drake allows its users to describe a set of tasks and its dependencies, and it uses this information to generate the DAG that represents the workflow. The three key features offered by Drake are:

- a DAG visualization and profiling tool, which allows the user to graphically visualize the tasks that are executed and their execution time;
- parallelism support, which, in conjunction with the explicit tasks dependencies, allows for parallel execution of multiple R sessions that compute independent tasks;
- memoization, which caches the result of previously executed tasks to avoid unnecessary computations and speed-up future executions.

B. StarVZ

StarVZ is a data analysis framework written in R for the visualization of traces in high-performance computing applications. It works in a two-phase process, in which the first phase is responsible for filtering, manipulating and tidying the data collected from the traces and the second phase is responsible for processing the visualization of the data.

In this work, we foscused on working with the first phase of the StarVZ processing, which involves a considerable number of operations on multiple data frames. The original source code implements a batch of operations on relatively independent data sets, some of which are merged together at given points of the execution pipeline. This pattern of operations can be suitably ported to a DAG representation.

III. Implementation

In order to port StarVZ to a new version that supports a workflow manager, some changes to the source code were needed. These changes reflect the effort to make task dependencies explicit and to postpone operations that do involve merging data from multiple data frames.

The first effort consists of separating the step of reading data from that of data manipulation. This separation is necessary if we want to maximize the number of in-memory operations

in further manipulation steps, which can be slowed down considerably if data needs to be read from the disk when demanded.

The next effort was to separate and take note of all the transformations that took place in the data frames during the data processing step. This step is relatively complex when the source code has not been originally written with task separation in mind, which was the case for StarVZ. Ideally, a data science workflow can be written from the ground up with clear, distinct operations for each data set that is involved in the process, such that extracting a DAG that represents separate tasks and identifying each task's dependencies is trivial.

After identifying independent tasks in the workflow, we used the Drake package to create a plan. A Drake plan is a recipe containing a description of the tasks in the workflow, and each task is represented by an R function to be called. When creating a plan, Drake inspects all the function definitions (i.e. tasks) identifying their inputs, which are interpreted as task dependencies. The outputs of these functions are called targets. Because Drake (and R, in general), enforce the functional programming paradigm, global state is not supported as task dependencies. Figure 1 shows the DAG created by drake with the tasks and dependencies that were described in the Drake plan. In the DAG, nodes in purple are the initial resources needed by each target (inputs and manipulation functions) and nodes in black are the targets to be computed during the execution of the workflow.

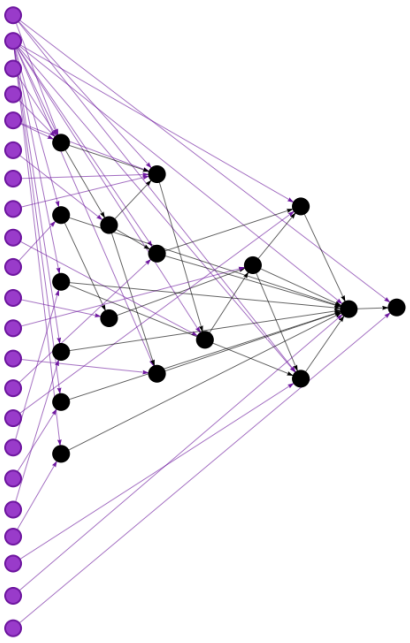

Fig. 1. Dependencies graph generated by Drake to execute the data manipulation of the first phase of StarVZ

As mentioned previously, Drake does not support the usage of global state as inputs for targets, meaning that functions should be pure (with no side effects) and every dependency should be explicitly defined. This allows for Drake to assume that, given a fixed set of inputs to a task, its output will always be the same. Consequently, Drake can cache the results of previous stages of the workflow to speed-up future executions by only calculating targets whose inputs have changed. Additionally, Figure 1 makes it explicit that every task in each column is independent from one another, meaning that they can be executed in parallel without concurrency drawbacks such as race conditions.

IV. Results

Even though Drake provides features that can potentially speedup code execution on data science pipelines, we could not observe the benefits of such features when porting the StarVZ code to a workflow.

The caching feature, for instance, saves the result of previous computations to disk in order to avoid recalculating the same steps repeatedly. However, because StarVZ's data frames are considerably large, writing them to disk in every stage of computation imposes a fairly heavy overhead in execution time.

With respect to parallel execution, Drake is limited by R's single-threaded implementation. In order to overcome this implementation decision, Drake starts multiple R sessions to compute parallel stages of the workflow. However, because these R sessions are separate processes, they are not able to share the same address space, and thus the data exchange between workers is done through the same caching system that writes the targets outputs to disk, once again imposing a fairly heavy overhead.

V. Conclusion and Future Work

In this study we explored the usage of Drake, a workflow manager for data science applications in R, to describe the data manipulation that take place in StarVZ. We conclude that using a DAG for workflow representation is helpful because it allows for the scientist to better understand the data flow of its application. Additionally, this representation also makes data dependencies explicit, and such information can be used to leverage parallel execution more easily. Caching of previous results is also possible, as long as the functions provided to the workflow manager are pure and there is no global state.

Unfortunately, StarVZ's data manipulation stage could not leverage the features of Drake that can positively impact performance. We believe that a combination of factors, such as large data frames being written to disk and the load imbalance between tasks that generate such large data frames are responsible for affecting the performance in this specific case.

As a solution to the previously mentioned drawbacks and as suggested future work, we can leverage the insights drawn by the DAG visualization of the workflow with respect to tasks dependencies to port StarVZ to a more robust and scalable data science framework, such as Apache Spark. In this case, Apache Spark can be used as a backend for the computations involved in StarVZ, and R frontends such as sparkR and sparklyr can be used as an interface.

REFERENCES

[1] H. Wickham, *tidyverse: Easily Install and Load the 'Tidyverse'*, 2017, r package version 1.2.1. [Online]. Available: https://CRAN.R-project.org/package=tidyverse

[2] W. Landau, "The drake r package: a pipeline toolkit for reproducibility and high-performance computing," vol. 3, p. 550, 01 2018.

[3] V. Garcia Pinto, L. M. Schnorr, L. Stanisic, A. Legrand, S. Thibault, and V. Danjean, "A Visual Performance Analysis Framework for Task-based Parallel Applications running on Hybrid Clusters," *Concurrency and Computation: Practice and Experience*, Apr. 2018. [Online]. Available: https://hal.inria.fr/hal-01616632