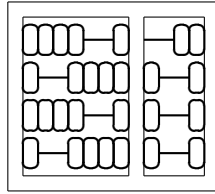


UNIVERSITY OF CAMPINAS
INSTITUTE OF COMPUTING



Dwsr: Predicting the memory usage of tensorial algorithms

Student: Daniel De Lucca Fonseca

Advisor: Prof. Edson Borin

Abstract: Lorem ipsum dolor

Contents

1	Introduction	1
2	Background	2
3	Related Work	2
3.1	Resource-aware scheduling	2
3.2	Resource-aware execution	3
4	Research proposal	3
4.1	Problem statement	3
4.2	Proposed solution	5
4.2.1	Predicting the memory-footprint of an algorithm	5
4.2.2	Automating the data partitioning process	6
4.3	Potential risks and limitations	6
4.3.1	Sensibility towards too many control statements	7
4.3.2	Unpredictable bottlenecks	7
4.3.3	Language-agnosticity	7
4.3.4	DASK's memory management	8
4.4	Problems to be addressed	8
4.4.1	How to measure memory usage	8
4.4.2	Historical data requirements	8
4.4.3	Python's garbage collection	9
4.4.4	Graph execution	9
4.5	Research questions	9
4.6	Research methodology	9
4.6.1	Feasibility	9
4.6.2	Accuracy	10
4.6.3	Applicability	10
4.6.4	Prototype development	11
4.6.5	Research extension	11

1 Introduction

Memory management is a crucial aspect of modern computer applications. Some problems, such as seismic processing, are particularly sensitive to the amount of available memory. Due to the large size of seismic data, even the most powerful supercomputers cannot execute a computing graph in a single node. To overcome this challenge, most seismic algorithms rely on data parallelism.

For some algorithms, choosing the right data partitioning strategy is quite straightforward. On the other hand, the optimal strategy isn't so obvious for others, usually because they require a large amount of work memory. The latter is true for some seismic processing algorithms, therefore the data partitioning strategy is usually defined after a series of trials and errors. Although this approach is effective, it is time consuming and requires a lot of human effort.

In this research proposal, I suggest exploring about a new data partitioning strategy for data parallelism. The proposed strategy is based on the *memory footprint* of the algorithm, which is the amount of memory required to execute it. This approach may be effective not only for seismic algorithms, but also for any other algorithm that has a predictable memory footprint. The process is divided into two main stages: (i) the *memory footprint estimation* stage, where the memory footprint of the algorithm is estimated; and (ii) the *data partitioning* stage, where the data is partitioned according to the estimated memory footprint.

For the memory footprint estimation stage, I propose to use a machine learning model to predict the memory footprint of the algorithm. Most of the existing research to predict the memory footprint of an algorithm focus on predicting resource consumption from a scheduler's perspective, while no research has been conducted to predict it for a single algorithm.

For the data partitioning stage, I propose to leverage Dask [1] automatic chunking to partition the data. Dask [1] is a Python library that provides a high-level interface to parallelize Python code. Considering that we can estimate the memory usage, figuring out the optimal chunk size is a matter of finding the relationship between the amount of available memory and the memory footprint itself.

During this research, I am planning to work with *DASF is an Accelerated and Scalable Framework* (DASF) [2] to implement the proposed strategy. DASF [2] is a library created by the Discovery laboratory at *Universidade Estadual de Campinas* (UNICAMP) to facilitate the development of seismic processing algorithms.

The following sections are organized as follows. I start by presenting the background on sec-

tion 2, explaining relevant concepts for this research. Then, I present the related work on section 3, where I discuss the existing research on memory usage estimation and resource-aware scheduling. Finally, I present the research proposal on section 4, where I describe the research plan and the expected results.

2 Background

Sit lorem dolor

3 Related Work

3.1 Resource-aware scheduling

Pupykina et al. [3] presets a nice overview of the memory management field until 2019. According to the authors, the main challenge on predicting memory usage is the lack of knowledge of the memory access patterns within an application. Most of current research focus on using machine learning techniques to bypass this problem. Although this approach has been successful, it is only capable of predicting the overall resource usage of a whole workload, and not the resource usage of a single task.

The observation made by Pupykina et al. [3] is visible while evaluating recent work. Most of the research done so far focus only on the scheduler perspective, using memory usage history to predict the expected resource requirements of a given cluster.

E. R. Rodrigues et al. [4] presents a machine learning model that can easily be integrated into a scheduler to predict the resource requirements of a given task. Their work is mainly focused on the scheduler perspective, since it uses past executions by the user to predict the resource requirements for future jobs. While submitting a new job to the cluster, the user must provide a manual estimate. This estimate is used by E. R. Rodrigues et al. [4] alongside with past executions to predict the actual resource requirements of the job. Although effective, this approach is vulnerable to spurious correlations, since past executions from other jobs by the same user can influence the prediction.

T. Mehmood et al. [5] presents an ensemble machine learning model that can predict the expected resource usage of a cloud provider. Like E. R. Rodrigues et al. [4], T. Mehmood et al. [5] uses past executions to predict the resource requirements at a given time.

3.2 Resource-aware execution

Table 1: Related work comparison table

Work	Used features	Prerequisites	Purpose
[6]	Input data and hardware characteristics	None	Predict resource consumption
[7]	Memory usage history and executed query	Query logs	Query resource usage estimate
[4]	Memory usage history, user data, and job data	Past executions	Scheduler resource allocation
[5]	Memory usage history and job data	Past executions	Scheduler resource allocation
[8]	Memory usage history and input structure	Past executions	Scheduler resource allocation
[9]	Resources being used and user-provided information	Trial and error	Scheduler resource allocation
[10]	Resources being used and memory pressure	Resource competition	Scheduler resource allocation
[11]	Memory usage estimate	None	Resource-aware scheduling
[12]	Memory usage history, input structure, and run command flags	Past executions and code instrumentation	Recurrent task scheduling
This work	Input structure	None	Parallelism optimization

4 Research proposal

In this research proposal, I aim to propose a new data partitioning strategy that is aware of the algorithm memory footprint. As seen in section 3, while there have been numerous studies on historical analysis of memory usage to predict resource requirements from the scheduler perspective, there is a significant gap in research when it comes to predicting memory usage of a single algorithm.

4.1 Problem statement

The Discovery ¹ laboratory, located at UNICAMP ², is working on a seismic analysis project with Petrobras ³. This project relies heavily on machine learning and seismic operators for its computing graphs. However, the input of these graphs is a massive seismic dataset that can contain terabytes

¹<https://discovery.ic.unicamp.br/>

²<https://ic.unicamp.br/>

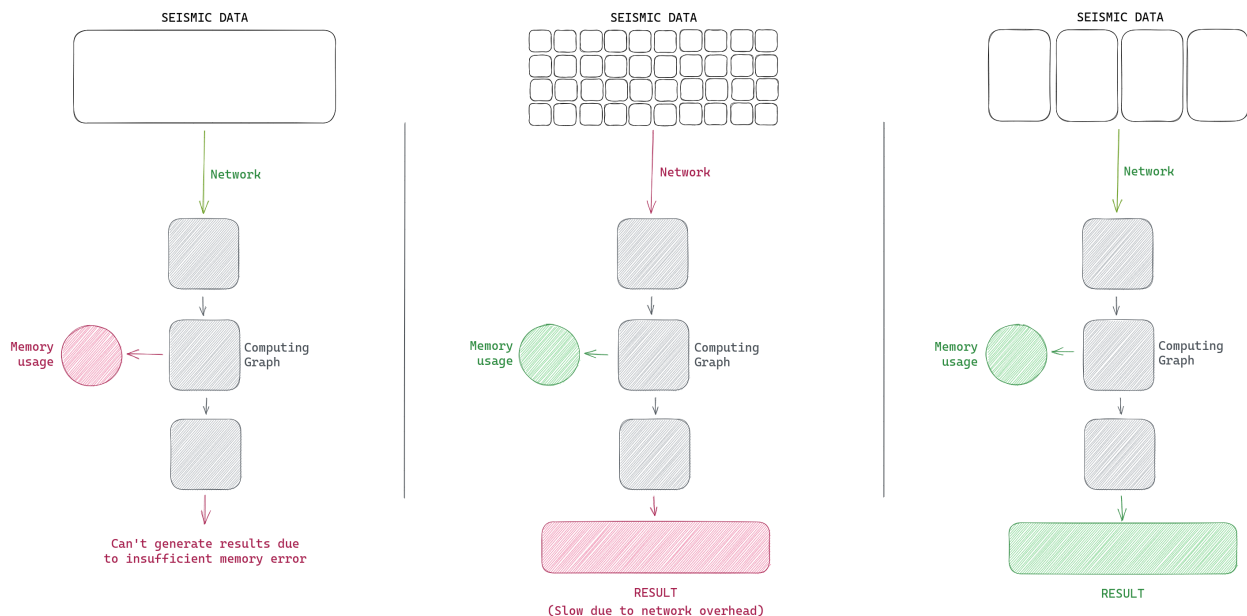
³<https://petrobras.com.br/>

of data. Even supercomputers do not have enough memory to handle the computation on a single node. Therefore, usually the execution is distributed by using data parallelism.

To facilitate this process, our laboratory implemented a framework called DASF [2] that simplifies the development of data-parallel computing graphs using Dask [1] under the hood. DASF [2] has an input parameter called **block_size** that defines the size of each seismic block used during data parallelism. However, setting this parameter can be challenging because it required finding the optimal relationship between it and the network overhead caused by it.

To illustrate this challenge, I present image 1 which contains three computing graphs receiving input data from a seismic dataset. The first graph shows the input data as the entire dataset, which requires a significant amount of memory to execute. The second graph divides the data into thousands of small parts, reducing the memory requirement but adding network overhead. The third graph divides the data into a smaller number of parts, minimizing both network and memory requirements.

Figure 1: Block size impact on memory and network usage



While executing the graph, the developer must manually set the `block_size` parameter. Setting a large number may lead to memory issues and it causes a significant delay due the trial-and-error nature of the execution flow. Since Petrobras uses supercomputers to execute those graphs, this delay is even larger considering the time it takes to submit a job due to the queue waiting time.

On the other hand, setting a small number may increase the execution time due to network

overhead. Since Petrobras have a large number of graphs to execute, and each graph usually takes a long time to execute, I need to find a way to optimize the `block_size` parameter.

Dask [1] provides an automatic chunking feature, but it relies on the `chunk_size` parameter, which is a static parameter define prior to execution. Since the developer do need to define this parameter prior to the execution, they can't rely on Dask [1] auto chunking feature to automatically split the data, but they can rely on the automatic chunking if they figure out the ideal `chunk_size` prior to the execution.

Figuring out the `chunk_size` for algorithms that doesn't require a large working memory is easy, since the developer can set that to a percentagem of the available memory. But, some of seismic operators used by Petrobras genereates a large working memory during the graph execution, which makes it difficult to determine the ideal `chunk_size`.

Based on this assumption, if someone predicts the memory usage of the graph that person can use the Dask [1] auto chunking feature to automatically split the data into the ideal number of chunks. Since DASF [2] uses Dask [1] under the hood, the `block_size` parameter on DASF [2] is equivalent to the `chunk_size` parameter on Dask [1]. Therefore, this research aims to develop an automated data partitioning strategy that is memory-footprint aware. In other words, I aim to create a DASF [2] plugin that can automatically set the optimal `block_size` parameter during execution based on a machine learning model that can predict the memory-footprint of the algorithm. This will help Petrobras to optimize resource utilization and minimize waiting and execution time. The model will provide a comprehensive understanding of memory usage patterns for different block sizes and contribute to the development of a more efficient data partitioning strategy to execute a graph in large-scale clusters.

4.2 Proposed solution

The proposed solution is divided into two stages: (i) predicting the memory-footprint of an algorithm, and (ii) automating the data partitioning process. Each stage is presented as a subsection of this section.

4.2.1 Predicting the memory-footprint of an algorithm

Most seismic operators are tensorial algorithms. Due to this fact, we can assume that the memory footprint of an algorithm is proportional to the shape of the input data. Based on this, to predict the memory-footprint of an algorithm I plan to create a machine learning model that would predict

the memory usage of an algorithm based on the input data.

The proposed solution can be executed in two different ways:

1. **Algorithm-specific model:** train a model for each algorithm, considering the seismic data's shape and size as the primary features for the prediction.
2. **Generic algorithm model:** train a model that is algorithm-agnostic, considering the source code as also a feature for the prediction.

Although creating a generic algorithm model is more flexible, I plan to start by coding an algorithm-specific model to understand the problem better. While coding the model, I expect to find input features that are relevant to the prediction. My initial hypothesis is that the input data's shape and size are the most relevant features, but I will experiment with other features to improve the prediction too.

Depending on the results of the initial experiments to create an algorithm-specific model, I may pursue the generic algorithm model. This secondary goal is to develop a model that is algorithm-agnostic. I understand that the source code of the algorithm contains relevant information, such as how the code author deals with memory management. However, I do not have a clear picture of the features that I could extract from the source code, but I believe that this is a possible experiment for this phase of the research.

4.2.2 Automating the data partitioning process

The second stage of the proposed solution is to automate the data partitioning process. The goal is to create a DASF [2] plugin that can automatically set the optimal `block_size` parameter during execution based on a machine learning model that can predict the memory-footprint of the algorithm. The plugin would be executed before the algorithm starts, and it would set the `block_size` parameter based on the prediction of the model.

4.3 Potential risks and limitations

In this section, I will discuss the limitations of the proposed solution for both predicting memory consumption, as well as automatically partitioning the data.

4.3.1 Sensibility towards too many control statements

A possible limitation for this solution is that it may not work if the algorithm has too many control statements. Depending on the structure of the algorithm, the amount of control statements may change the memory usage.

This limitation can be ignored if the memory allocation of the algorithm happens before the control statements. On this scenario, even with control statements that change drastically the execution of the algorithm the memory usage will not change.

Either way, all the seismic operators and machine learning models being used are tensorial algorithms, which means that they do not have too many control statements. Therefore, this possible limitation will not be a problem during the research.

4.3.2 Unpredictable bottlenecks

There are two possible memory bottlenecks in the proposed solution: *Central Processing Unit* (CPU) and *Graphics Processing Unit* (GPU) memory. Seismic operators are likely to be GPU-memory bounded, but I need to conduct experiments to verify this.

If the algorithms can be both GPU memory and CPU memory bounded, then the proposed solution must be able to handle both scenarios.

4.3.3 Language-agnosticity

The proposed solution is currently focused on using Python since it is the language used for the seismic operators and Dask [1] itself.

Although Python is a popular language in the scientific community, it may not be the language of choice for all researchers. This solution may not be language-agnostic at the moment. However, in the future, I can improve the solution to accept algorithms from any language.

In the case of the algorithm-specific model, I can improve the training structure to accept algorithms from any language. This can be done by allowing decoupling both the feature extraction, as well as the execution of the algorithm, from the training process.

In the case of the generic algorithm model, it is possible improve the feature-extraction part to allow more languages as well. Since the only difference between the two strategies is allowing to extract features from the source code, a specialized feature extractor per-language should be enough.

4.3.4 DASK’s memory management

Since DASF [2] uses Dask [1] under the hood, I need to understand how Dask [1] manages memory. As far as my initial research went, Dask [1] has an automatic chunking feature that deals with executing the proper data partitioning as long as the developer can define the maximum chunk size in bytes.

This feature seems promising, and it may solve most of the challenges related to automatic data partitioning. But, I may need to conduct proper experiments to understand exactly how this feature works and what are its limitations. Depending on the results of such experiments, I may need to adapt the proposed solution to work with Dask’s [1] memory management.

4.4 Problems to be addressed

In this section, I will outline the potential problems that I may encounter during our research and how I plan to address them. Each problem is going to be presented as a subsection, with a brief description of the problem and how I plan to address it.

4.4.1 How to measure memory usage

One of the primary challenges in our research is accurately measuring memory usage. This is particularly true for GPU memory, which may differ from CPU memory. I aim to explore different options to measure memory usage accurately.

I plan to start by investigating if there is an API to measure GPU memory, and if not, I will explore common libraries that allow gathering memory consumption based on a specific process. I understand that the optimal approach would be to use a specific API for this, since this would allow more flexibility for our tool, but I will explore alternative options if necessary.

4.4.2 Historical data requirements

Many of the current approaches require a significant amount of historical data to train the models. This is not feasible for our research, as it would be time-consuming to generate such data for algorithm-specific models. I aim to overcome this limitation by creating a proper discovery approach by merging reinforcement learning with Bayesian analysis.

I plan to gather the minimal amount of data, even by executing the algorithm locally with a small amount of synthetic data, and then cover the exploration space for the input data for shapes

and data with features the model have never seen before. With this approach, I think the model will be able to learn while it is being used, and it will be able to generalize to new data.

4.4.3 Python’s garbage collection

Python’s garbage collector can pose a problem for us. Python uses reference-counting as its garbage collection strategy, and it is lazy, so it usually waits for the memory to be needed to clean. If our operators are CPU memory bounded, I may need to figure out how to deal with Python’s garbage collector to gather the real memory usage. However, I will only need to address this issue depending on the results of our experiments.

4.4.4 Graph execution

Another problem I may encounter is how to figure out the entire graph’s memory requirements. While our proposed solution can help us find the amount of memory required for a specific algorithm, integrating multiple algorithms into a graph poses a challenge. I plan to address this issue by predicting not only the memory usage, but also the output shape of the algorithm and its features. By having prior knowledge of the algorithms in the graph, I can compose a graph with the models using the output of the first model as the input data of the second one.

4.5 Research questions

In this section, I present the research questions I aim to answer in this study. The research questions are divided into different categories, including: (i) feasibility, (ii) accuracy, (iii) and applicability. Table 2 summarizes the research questions and their respective categories.

4.6 Research methodology

In this section, I will describe the methodology I will follow to answer the proposed research. Each research question is going to be answered by a set of experiments. To organize the execution of the experiments, I will use the categories presented on section 4.5.

4.6.1 Feasibility

This experiment category aims to answer **RQ1**, **RQ2**, **RQ3**, and **RQ4**. The goal is to understand the feasibility of our seismic operators and their main characteristics. I plan to start with an

Table 2: Research questions

#	Question	Category
RQ1	How DASF [2] deals with automatic chunking?	Feasibility
RQ2	What is the optimal way of gathering memory-usage data?	Feasibility
RQ3	Are seismic tensorial algorithms memory bounded by the CPU or GPU?	Feasibility
RQ4	Which features do we need to extract from the input data?	Feasibility
RQ5	What is the memory-usage behavior of our algorithms?	accuracy
RQ6	Under extreme circumstances, how is the memory usage of our algorithms?	accuracy
RQ7	How to integrate a reinforcement learning model with Bayesian analysis?	Applicability

experiment to explore how DASF [2] deals with automatic chunking (RQ1). Then, I will try to figure out the proper way to gather memory usage metrics from both the CPU and GPU (RQ2). After this, I aim to have an experiment that will explore if our seismic operators are GPU-memory bounded or CPU memory bounded (RQ3). Lastly, I will explore multiple executions of our seismic operators, trying to understand possible features we can extract from the inputs, not only the shape, but also other possible features that we can extract from the data (RQ4).

4.6.2 Accuracy

This experiment category aims to answer **RQ5** and **RQ6**. The goal is to understand how reliable the results can be. I will first explore the behavior of the seismic operators, exploring how they use memory in a set of different synthetic executions (RQ5). Finally, I will try pushing our algorithms to the limit to check if, under extreme circumstances (like uncommon inputs), the algorithm can break or display an unpredictable memory-usage pattern (RQ6).

4.6.3 Applicability

The third, and last, experiment category is applicability, which aims to answer RQ7. The goal is to act as a pre-prototype experiment. I aim to explore reinforcement learning models and Bayesian analysis to understand how I can apply the results from the previous experiments to predict memory usage.

4.6.4 Prototype development

After we execute all the experiments, I will implement a prototype. The idea for that prototype is to act as a plugin for DASF [2] and use it as a contribution to the active Petrobras seismic project in our laboratory. The plugin can act not only to tune the input `block_size` for every operator, but also as a decision heuristic for scheduling.

4.6.5 Research extension

Based on the results of all past experiments, I can focus on the second proposed solution. This part aims to explore the possibility of generalizing the developed machine learning model to be algorithm-agnostic. To implement this, I need to figure out how and which features to extract from the original source code. If this is possible, I can develop a generic model that can be used to predict memory usage for any seismic operator.

References

- [1] M. Rocklin *et al.*, *Dask*, <https://github.com/dask/dask>, 2023.
- [2] J. Faracco, O. Napoli, and E. Borin, *Dasf is an accelerated and scalable framework*, <https://github.com/discovery-unicamp/dasf-core>, 2023.
- [3] A. Pupykina and G. Agosta, “Survey of memory management techniques for hpc and cloud computing,” *IEEE Access*, vol. 7, pp. 167 351–167 373, 2019. DOI: 10.1109/ACCESS.2019.2954169.
- [4] E. R. Rodrigues, R. L. F. Cunha, M. A. S. Netto, and M. Spriggs, “Helping hpc users specify job memory requirements via machine learning,” in *2016 Third International Workshop on HPC User Support Tools (HUST)*, 2016, pp. 6–13. DOI: 10.1109/HUST.2016.006.
- [5] T. Mehmood, S. Latif, and S. Malik, “Prediction of cloud computing resource utilization,” in *2018 15th International Conference on Smart Cities: Improving Quality of Life Using ICT & IoT (HONET-ICT)*, 2018, pp. 38–42. DOI: 10.1109/HONET.2018.8551339.
- [6] D. Duplyakin, J. Brown, and D. Calhoun, “Evaluating active learning with cost and memory awareness,” in *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2018, pp. 214–223. DOI: 10.1109/IPDPS.2018.00031.

- [7] C. Tang, B. Wang, Z. Luo, *et al.*, “Forecasting sql query cost at twitter,” in *2021 IEEE International Conference on Cloud Engineering (IC2E)*, 2021, pp. 154–160. DOI: 10.1109/IC2E52221.2021.00030.
- [8] R. Ferreira da Silva, G. Juve, E. Deelman, *et al.*, “Toward fine-grained online task characteristics estimation in scientific workflows,” Nov. 2013, pp. 58–67, ISBN: 9781450325028. DOI: 10.1145/2534248.2534254.
- [9] T. S. Phung, L. Ward, K. Chard, and D. Thain, “Not all tasks are created equal: Adaptive resource allocation for heterogeneous tasks in dynamic workflows,” in *2021 IEEE Workshop on Workflows in Support of Large-Scale Science (WORKS)*, 2021, pp. 17–24. DOI: 10.1109/WORKS54523.2021.00008.
- [10] M. Fang Juanand Wang and H. Sun, “Research of task scheduling mechanism based on prediction of memory utilization,” in *Mobile Ad-hoc and Sensor Networks*, L. Zhu and S. Zhong, Eds., Singapore: Springer Singapore, 2018, pp. 227–236, ISBN: 978-981-10-8890-2.
- [11] A. V. Goponenko, R. Izadpanah, J. M. Brandt, and D. Dechev, “Towards workload-adaptive scheduling for hpc clusters,” in *2020 IEEE International Conference on Cluster Computing (CLUSTER)*, 2020, pp. 449–453. DOI: 10.1109/CLUSTER49012.2020.00064.
- [12] B. T. Shealy, F. A. Feltus, and M. C. Smith, “Intelligent resource provisioning for scientific workflows and hpc,” in *2021 IEEE Workshop on Workflows in Support of Large-Scale Science (WORKS)*, 2021, pp. 9–16. DOI: 10.1109/WORKS54523.2021.00007.