UNIVERSITY OF CAMPINAS

INSTITUTE OF COMPUTING

Efficient Memory Management in Data Parallel Computing: A Chunk Optimization Technique

Student: Daniel De Lucca Fonseca

Advisor: Prof. Edson Borin

Abstract: Efficient job resource allocation in large-scale computing clusters is often hindered by the challenge of accurately predicting memory usage for certain complex algorithms. Seismic operators stand out as one such class of algorithms that can exhibit highly variable and hard-to-predict memory requirements. When processing these operators in computing clusters, substantial time is lost due to factors such as suboptimal resource allocation, inefficient data partitioning, and the need for additional data transfers between processing units.

To mitigate these issues, data is commonly divided into smaller chunks for parallel processing across multiple nodes. However, determining the optimal chunk size remains a challenging task, as it requires striking a delicate balance between memory usage, computational efficiency, and inter-node communication overhead. Gaining insight into an algorithm's memory footprint can greatly simplify this task, enabling better resource allocation, reduced execution time, and improved overall performance.

Existing research in this domain primarily focuses on scheduler-based approaches for resource usage prediction or highly specific use-cases that are not easily generalizable. Recognizing this gap in the literature, this study aims to develop a reinforcement learning model capable of predicting memory usage across a broader range of scenarios.

Moreover, this research seeks to provide a deeper understanding of the relationship between memory usage and parallel processing efficiency. This knowledge will not only benefit the processing of seismic operators but also extend to other algorithms with similar characteristics. Ultimately, our reinforcement learning-based approach aims to enhance the performance of large-scale computing clusters and contribute to more effective resource management in diverse computational settings.

Contents

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 can not store the whole dataset on the memory while processing it, hence, this kind of data is usually partitioned and processed in chunks.

For some algorithms, choosing the right data partitioning strategy is quite straightforward, since some frameworks, like Dask [1], provides automatic chunking. On the other hand, the optimal strategy is not so obvious for others, usually because they require a large amount of work memory. For such algorithms, the amount of used memory is not limited by the size of the input data, since they may require additional memory to store intermediate results. The latter is true for some seismic processing algorithms, therefore the data partitioning strategy is usually defined after a series of trials and errors.

In this research proposal, I suggest the creation of an efficient 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 usage.

The proposal is based on discovering the relationship of the input data shape and the memory footprint of the algorithm. By discovering this relationship, we can define the optimal chunk size for the amount of available memory on the worker. This tool can be used by frameworks like Dask [1] to enhance their automatic chunking strategy, leading to a more efficient data partitioning.

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 based on Dask [1] and created by the Discovery laboratory at *Universidade Estadual de Campinas* (UNICAMP) to simplify the development of seismic processing algorithms.

The following sections are organized as follows. I start by presenting the background on section 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. Finally, I present the research proposal on section 4, where I describe the research plan and the expected results.

2 Background

In this section I will present and discuss all relevant background knowledge that is needed to understand the rest of the proposal. Each subsection will present a specific topic, giving a general overview of it, and discussing its relevance to the project.

2.1 Seismic attributes

Seismic attributes are quantitative measures derived from seismic data, which describe various aspects of the subsurface geology. Over the past few decades, seismic attributes have become an indispensable tool in the exploration and production of hydrocarbons, mineral resources, and the management of geohazards. They have significantly contributed to the understanding of complex geological settings and have improved the accuracy of reservoir characterization and prediction.

Seismic data, which is obtained through the use of controlled seismic sources and an array of receivers, is used to create images of the subsurface geology. The acquired data is processed and interpreted to reveal geological features, such as faults, fractures, and rock properties. However, seismic data can be challenging to interpret due to the complex nature of the subsurface geology and the limitations of seismic imaging techniques.

The evolution of seismic attributes is explored in Barnes and Arthur [3], Chopra *et al.* [4], Fawad *et al.* [5], and Taner and Turhan [6]. This concept was first introduced in the late 1970s as a means to enhance the interpretation of seismic data. Since then, the field has experienced significant advancements, driven by the growth in computational power and the development of innovative seismic processing and interpretation techniques. Seismic attributes can be broadly classified into five categories:

- **Amplitude-based attributes**: These attributes are derived from the amplitude of the seismic waveforms and are used to analyze variations in rock properties, fluid content, and stratigraphy. Common examples of amplitude-based attributes include average amplitude, root-mean-square amplitude, and instantaneous amplitude;

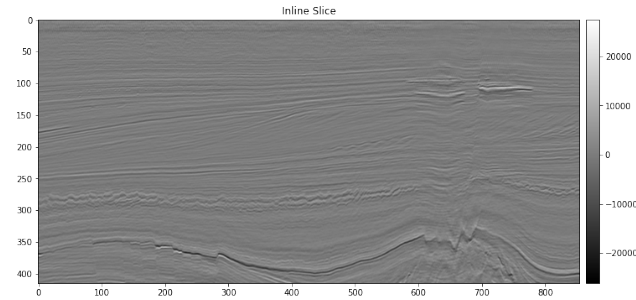
- **Frequency-based attributes:** These attributes focus on the frequency content of the seismic data and can reveal information about the thickness and composition of subsurface layers. Examples of frequency-based attributes include spectral decomposition, dominant frequency, and instantaneous frequency;

- **Time and spatial-based attributes:** These attributes describe the geometrical and temporal properties of seismic events, such as the continuity, dip, and azimuth of reflectors. Some examples of time and spatial-based attributes are coherence, curvature, and semblance;
- **Mitigation attributes:** These attributes are designed to reduce the impact of noise and other artifacts in the seismic data, improving the overall quality and interpretability of the data. Examples of mitigation attributes include random noise attenuation, multiple suppression, and ground roll removal;
- **Texture attributes:** These attributes characterize the texture or patterns within the seismic data, providing insights into geological features such as fractures, channels, and stratigraphic boundaries. Texture attributes can be computed using techniques such as gray-level co-occurrence matrices, wavelet decomposition, and local binary patterns.

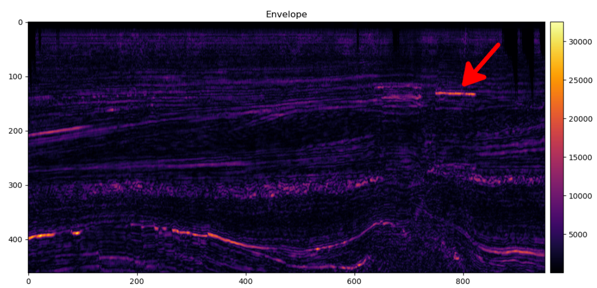
To illustrate how seismic attributes are used, figure 1 shows a sample of two complex seismic attributes derived from the amplitude of a sample seismic data. The envelope of the amplitude, which is ilustrated by figure 1b, highlights the presence of a possible hydrocarbons reservoir. Figure 1c shows the instantaneous phase of the seismic data, which defines the phase of the seismic wavelet at each sample.

As explained in the previous example, the integration of seismic attributes into the interpretation workflow allows geoscientists to extract valuable information from seismic data more efficiently. Seismic attributes can be combined with other data, such as well logs and geological models, to provide a comprehensive understanding of the subsurface geology. Furthermore, advanced machine learning and data analytics techniques have enabled the development of multi-attribute analysis, which involves the simultaneous examination of multiple attributes to identify patterns and relationships that may not be apparent when considering individual attributes.

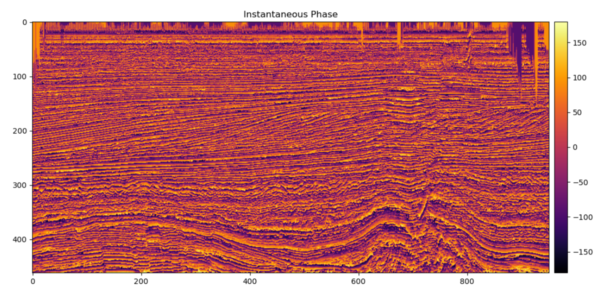
In summary, seismic attributes play a crucial role in the analysis and interpretation of seismic data by providing quantitative measures of subsurface geology. They enhance the understanding of complex geological settings, improve reservoir characterization, and aid in the prediction of subsurface resources. As the field continues to evolve, the integration of seismic attributes with other data sources and advanced computational techniques will further advance the state of knowledge in the field and enable more accurate and efficient exploration and production efforts.



(a) Amplitude (raw data)



(b) Amplitude envelope (the red arrow highlights a possible hydrocarbons reservoir)



(c) Instantaneous phase

Figure 1: Sample of two complex seismic attributes derived from the amplitude to the inline

2.2 Memory footprint

The memory footprint of an algorithm refers to the amount of memory space required for its execution, considering both static and dynamic memory allocations. It is a crucial performance metric, as it directly affects the overall system resources and impacts the efficiency and scalability of an algorithm. In essence, the memory footprint is an essential aspect of an algorithm's resource consumption, alongside other factors such as time complexity and processing power.

Static memory allocation is the memory allocated during compile-time, which includes the memory needed for storing executable code, global variables, and static local variables. This memory remains fixed throughout the program's execution and is typically allocated in the text, data, and *Block Started by Symbol* (BSS) segments.

Dynamic memory allocation, on the other hand, refers to the memory allocated during run-time, including the memory needed for storing dynamically allocated variables, function call stacks, and memory required by recursive functions. This type of memory allocation occurs in the heap and stack segments.

A deeper understanding of the memory footprint of an algorithm can be achieved by analyzing the following components:

- **Space Complexity:** Measures the total amount of memory used by an algorithm as a function of its input size. It is generally expressed using big-O notation, such as (n) or (n^2), where n is the input size. Space complexity can be further categorized into two subtypes: (i) auxiliary space (temporary memory used during execution) and (ii) input space (memory needed for storing input data);
- **Data Structures:** The choice of data structures employed by an algorithm significantly influences its memory footprint. Different data structures have varying memory overheads and trade-offs, and selecting the most appropriate data structure can lead to substantial improvements in memory usage;
- **Memory Management Techniques:** The way an algorithm manages memory can have a substantial impact on its memory footprint. This includes memory allocation and deallocation strategies, garbage collection, and other techniques that optimize memory usage during the algorithm's execution.

The memory footprint of an algorithm is a critical aspect of its performance, as it directly influences the overall system resources and efficiency. By comprehending and optimizing the memory footprint of an algorithm, developers can create more efficient and scalable solutions, ultimately contributing to improved computational capabilities in various applications and industries.

2.3 Dask

Lorem ipsum dolor

2.4 Machine Learning

Lorem ipsum dolor

2.5 Reinforcement Learning

Lorem ipsum dolor

3 Related Work

In this section I will discuss the existing research related to this proposal. Most of the existing work aiming to predict the memory consumption are focused on resource-aware scheduling. The most common use-case for this type of research is to predict memory consumption in order to efficiently allocate resources in a cluster. There are a few works that focus on resource-aware execution, most of those focused entirely on a extremely specific use-case.

During this section I will discuss both approaches, comparing and contrasting the current state-of-the-art of each. I will also discuss the main differences between the proposed work and the existing research. Finally, I present a comparison table that summarizes my main findings.

3.1 Resource-aware scheduling

Pupykina *et al.* [7] presets a broad 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.* [7] 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.* [8] presents a machine learning model that can easily be integrated into a scheduler to predict the resource requirements of a given task when executing on a cluster. 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 submiting a new job to the scheduler, the user must provide a manual estimate. This estimate is used by E. R. Rodrigues *et al.* [8] 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.* [9] presents an ensemble machine learning model that can predict the expected resource usage of a cloud provider at a given period of time. That estimate is calculated based on the resource usage of recent tasks submitted to that specific provider. Like E. R. Rodrigues *et al.* [8], T. Mehmood *et al.* [9] uses past executions to predict the resource requirements at a given time. As the input data to train the machine learning model, T. Mehmood *et al.* [9] uses a dataset provided by Google containing the trace data of a large number of jobs executed on Google Cloud Platform.

Similarly to both T. Mehmood *et al.* [9] and E. R. Rodrigues *et al.* [8], Phung *et al.* [10] proposes an approach to use past executions to achieve a smaller upper-bound on resource allocation. To do so, Phung *et al.* [10] uses a trial and error method where the scheduler tries to increase the available resources for a given job until it reaches the point where the job stops failling.

On the other hand, Fang *et al.* [11] takes a different approach. Instead of using historical data to predict the exact amount of resources required, Fang *et al.* [11] uses a machine learning model to predict the current memory pressure of a given cluster. Their research is coupled with the Hadoop [12] framework, and analyzes the status of all active jobs before submitting a new one on the same cluster. This approach can not be so effective to predict the resource requirements of a single job, but it can be used to predict the overall performance of the cluster itself.

All the research discussed so far focus on handling proper resource allocation within the scheduler. Although effective, all of them require either a large amount of historical data or a large number of incoming jobs to train the machine learning model. Both the required amount of historical data, and the focus on resource allocation based on a Heterogeneous pool of incoming jobs, limits the usage of those approaches to large-scale clusters.

Considering this limitation, Ferreira da Silva et al. [13] proposes a machine learning model that uses a clustering approach to evaluate the input data and predict the resource requirements of a given job. Their approach has a high accuracy, but the results varies a lot based on the size and density of the input itself.

During their research, Ferreira da Silva *et al.* [13] discovered that smaller datasets has a higher correlation rate between the parameters that the clustering algorithm extracts with the resource consumption itself. This observation is important, since it shows that it is possible to extract features from the input data that can be used to predict the resource requirements of a given job. As the final goal of their research, Ferreira da Silva *et al.* [13] created an online estimator tool, designed explicitly to be used by schedulers in order to improve their scheduling process.

Similar to Ferreira da Silva *et al.* [13], B. T. Shealy *et al.* [14] tries to extract features from the components of the execution in order to predict the resource consumption. Instead of gathering features only from the input data, B. T. Shealy *et al.* [14] also uses a set of user-defined run parameters as possible features to train the model. This approach aims to create an algorithm-specific model that would be able to predict the resource consumption of job. The main limitation of this approach is the requirement of building a training dataset for each algorithm that the user wants to predict the resource consumption. Due to this fact, this approach is only suitable for recurrent algorithms that are executed multiple times, changing only the input data and a few run parameters.

Finally, A. V. Goponenko *et al.* [15] focus on a different perspective. Their work proves that resource-aware scheduling can be way more effective than traditional scheduling algorithms. They have integrated their tool to Slurm taking into account resource requirements while scheduling new jobs. During the research, A. V. Goponenko *et al.* [15] discovered that their test workload executed 9.4% faster than the original scheduling, with a more efficient usage of the cluster resources.

3.2 Resource-aware execution

D. Duplyakin *et al.* [16] takes a different approach to evaluate the resource usage of a given algorithm. Instead of using the execution history in order to predict the expected requirements to efficiently handle resource allocation, D. Duplyakin *et al.* [16] tries to incrementally model the memory usage of the algorithm using an active learning approach. Their discovery is important, since they successfully demonstrated that it is possible to integrate active learning with gaussian process regression in order to explore the resource usage of a given algorithm. However, their approach is tighly coupled to their use-case, but the same concept can be explored on a more general approach.

Following a similar perspective, C. Tang *et al.* [17] proposes a method to predict the resource usage of a given query. Their approach was developed inside Twitter [1], and aims to simplify the calculation of the resource usage of SQL queries. During their research, C. Tang *et al.* [17] were able to extract keywords and features directly from the query itself and use those features to increase the prediction accuracy. With that method, they were able to achieve an average accuracy of 97%. Considering this, their research demonstrated that it is possible to use the source code of a given algorithm to predict its resource usage. Although their context is limited to SQL queries, the same concept can be generalized to other programming languages.

As demonstrated by D. Duplyakin *et al.* [16] and C. Tang *et al.* [17], it is possible to predict the resource usage of a single algorithm both by exploring key aspects of the application, as well as by dynamically exploring unknown executions. However, both approaches are limited to a specific

[1] https://twitter.com/

use-case, and they do not provide a general solution to predict the resource usage of any given algorithm.

Table 1: Related work comparison table

Work	Used features	Prerequisites	Purpose	Perspective
[8]	Memory usage history, user data, and job data	Past executions	Scheduler resource allocation	Scheduler
[9]	Memory usage history and job data	Past executions	Scheduler resource allocation	Scheduler
[10]	Resources being used and user-provided information	Trial and error	Scheduler resource allocation	Scheduler
[11]	Resources being used and memory pressure	Resource competition	Scheduler resource allocation	Scheduler
[13]	Input data parameters	Extensive application profiling	Efficient task scheduling	Scheduler
[14]	Memory usage history, input structure, and eun command flags	Past executions and code instrumentation	Recurrent task scheduling	Scheduler
[15]	Memory usage estimate	None	Efficient task scheduling	Scheduler
[16]	Input data and hardware characteristics	None	Predict resource consumption	Algorithm
[17]	Memory usage history and executed query	Query logs	Query resource usage estimate	Algorithm
This work	Input data shape	None	Parallelism optimization	Algorithm

4 Research proposal

In this research proposal, I aim to explore how to predict the memory footprint of a single algorithm. As seen in section 3, while there have been numerous studies on historical analysis of memory usage to predict resource requirements, they are most from the scheduler perspective. There is a significant gap in research when it comes to predicting memory usage of a single algorithm, specially focusing on data parallelism optimization.

4.1 Problem statement

The Discovery [2] laboratory, located at UNICAMP [3], is working on a seismic analysis project with Petrobras [4]. This project aims at developing a framework called DASF [2], which facilitates the execution of machine learning algorithms and seismic attribute operators on computing clusters. However, the input of the graphs created for seismic analysis are massive datasets that can contain

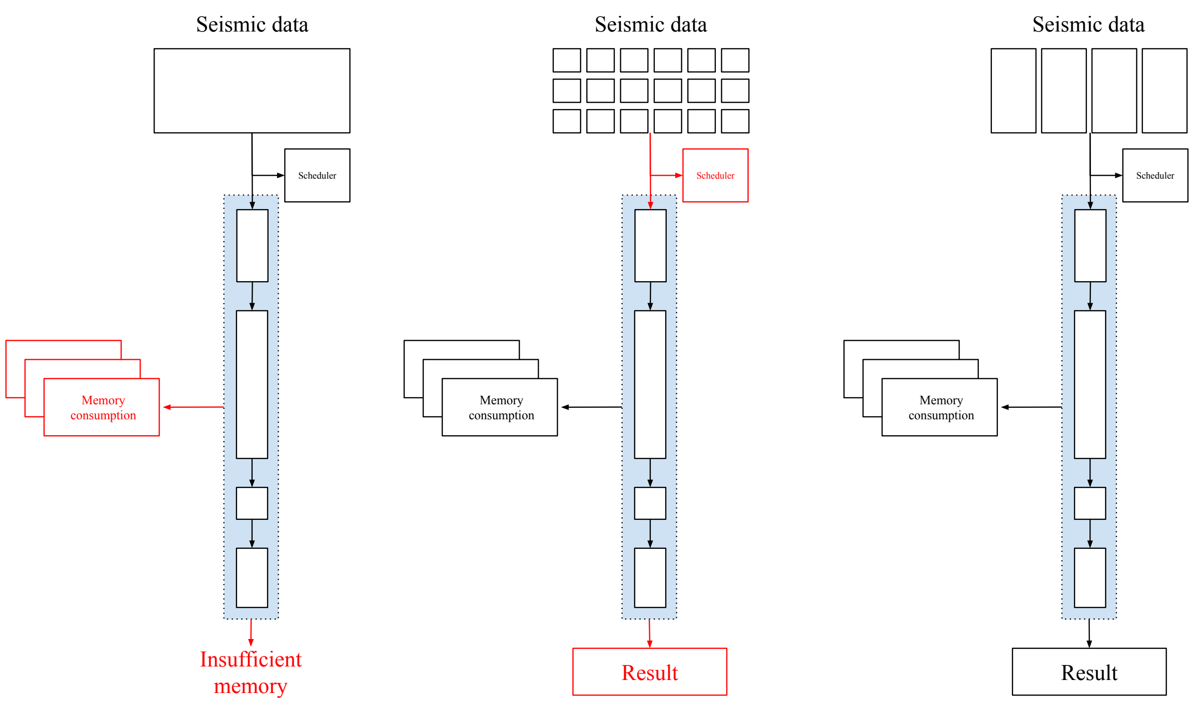
[2] https://discovery.ic.unicamp.br/
[3] https://ic.unicamp.br/
[4] https://petrobras.com.br/

terabytes 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, DASF [2] has a parameter called ”block size”. With the value of that parameter, DASF [2] uses Dask’s [1] automatic chunking feature to split the dataset into chunks. 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 2 which contains three computing graphs receiving input data from a seismic dataset. The first graph illustrates the situation in which the input data is processed as a whole, which requires a significant amount of memory to store the data during the execution. The second graph divides the data into thousands of small parts, reducing the memory requirements but adding network and scheduler overhead. The third graph divides the data into a smaller number of parts, minimizing both network and memory requirements.

Figure 2: 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 cause significant delays 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 and scheduler 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 defined prior to execution. Figuring out that parameter for algorithms that does not require a large working memory is easy, since the developer can set that to a percentage of the available memory. But, some of the seismic operators used by Petrobras generates 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 Dask's [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 a way to understand the memory-footprint of a graph to simplify the decision of the ideal chunk size.

As a practical usage, 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

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 its input parameters, 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 he 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.

4.2.1 Practical usage

To explore the practical usage of the proposed solution, I plan to automate the data partitioning process on DASF [2]. The goal is to create a 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 Memory usage variance based on the input data

A possible limitation for this solution is that it may not work if the algorithm structure contains control statements that drastically change the memory usage of the algorithm. For example, if the algorithm's memory usage varies depending on the data itself, then the proposed solution may not work.

It is important to mention that this limitation only happens if the control statement affects the memory allocation directly. Even if the algorithm has many control statements that change the execution flow drastically, this limitation can be ignored if the memory allocation of the algorithm happens before those.

Either way, all the seismic operators and machine learning models being used are tensorial algorithms, which means that their execution flow would not vary based on the input data. Therefore, I do not expect this possible limitation to 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 consumption. 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 exract 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.

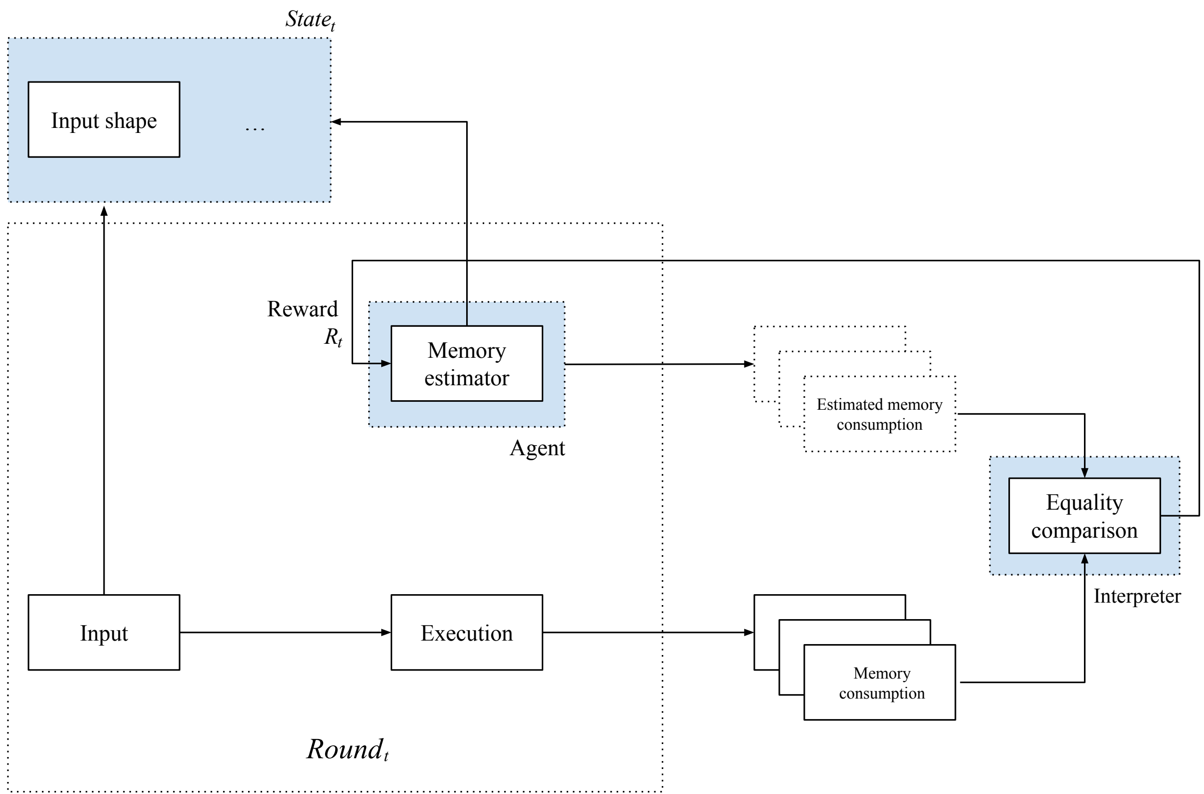
I plan to start by investigating if there is an API to measure GPU memory consumption, and if not, I will explore common libraries that allow gathering memory consumption based on a specific process. I understand that the optimal approach would be to use a specific API for this, since this would allow 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 my research, as it would be time-consuming to generate such data for algorithm-specific models. Also, even if I decide to generate such data, this would add a requirement for the user to have a significant amount of data to train the models for any new algorithm they want to use.

To overcome this limitation, I plan to implement a reinforcement learning approach to train the models. This approach will allow the model to learn from the data while it is being used, learning from its own mistakes and improving its predictions. Figure 3 shows a diagram illustrating how this approach will work. The memory estimator is going to act as the agent, using the input shape and other features as its state to generate a estimate of the memory usage. A equality function will be used to compare the predicted memory usage with the actual memory usage, and this will be used to calculate the reward for the model. Each execution of the algorithm will act as a round to train the model.

Figure 3: Reinforcement learning execution diagram



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 collect the garbage and free memory space. 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 when collecting memory consumption information to train the ML models. 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 and methodology

In this section, I present the research questions I aim to answer in this study and I will describe the methodology I will follow to answer those questions. Each research questions is going be answered by a set of experiments. To organize the execution of the experiments, I will divide the questions into different categories, including: (i) feasibility, (ii) accuracy, (iii) and applicability. Table 2 summarizes the research questions and their respective categories.

Table 2: Research questions

#	**Question**	**Category**
RQ1	How Dask [1] 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

4.5.1 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 experiment to explore how Dask [1] 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).

4.5.2 Accuracy

This experiment cateogry 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).

4.5.3 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.

4.5.4 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.

4.5.5 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.

4.6 Work plan and schedule

This research project is divided into three phases. The phases are supposed to be executed in sequence, being each one responsible for a specific part of the project. The first phase is the *experimentation phase*, which is responsible for the execution of all experiments discussed on section 4.5. The second phase is the *prototype and evaluation phase*, which aims to develop a prototype of the proposed solution using DASF [2] and evaluate the results. Finally, the third phase is the consolidation phase, in which I am going to summarize the research results, writting the final report.

The gantt chart of the work plan and schedule of the project is presented in table 3. Each activity is described as a number, and the details of each activity is presented as follows:

1. **Experimentation phase**;
   (a) Execution of feasibility experiments, as seen in section 4.5.1;
   (b) Execution of accuracy experiments, as seen in section 4.5.2;
   (c) Execution of applicability experiments, as seen in section 4.5.3.
2. **Prototype and evaluation phase**;
   (a) Development of all the required structure on DASF [2] to support the proposed solution;
   (b) Implementation of the reinforcement learning model;
   (c) Execution of the initial evaluation of the implemented model;
   (d) Implementation of the initial improvements on the model;
   (e) Execution of the final evaluation of the implemented model.
3. **Consolidation phase**.
   (a) Summarization of the research results;
   (b) Writing of the final report and dissertation.

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. E. Barnes, "Seismic attributes in your facies," *CSEG recorder*, vol. 26, no. 7, pp. 41–47, 2001.

[4] S. Chopra and K. J. Marfurt, "Seismic attributes - a historical perspective," *Geophysics*, vol. 70, no. 5, 3SO–28SO, 2005.

Table 3: Work plan and schedule with dates

Activity	2023								2024				
	May	**Jun**	**Jul**	**Aug**	**Sep**	**Oct**	**Nov**	**Dec**	**Jan**	**Feb**	**Mar**	**Apr**	**May**
1.a	/////	/////											
1.b			/////										
1.c				/////									
2.a					/////								
2.b						/////	/////						
2.c								/////					
2.d								/////					
2.e									/////	/////			
3.a											/////		
3.b												/////	/////

[5] M. Fawad, J. A. Hansen, and N. H. Mondol, "Seismic-fluid detection - a review," *Earth-Science Reviews*, p. 103 347, 2020.

[6] M. T. Taner, "Seismic attributes," *CSEG recorder*, vol. 26, no. 7, pp. 48–56, 2001.

[7] 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`.

[8] 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`.

[9] 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`.

[10] 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`.

[11] 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.

[12] A. S. Foundation, *Hadoop*, version 0.20.2, Feb. 19, 2010. [Online]. Available: `https://hadoop.apache.org`.

[13] 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`.

[14] 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`.

[15] 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`.

[16] 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`.

[17] 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`.