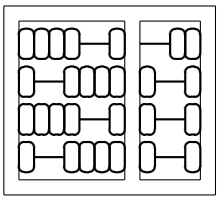
UNIVERSITY OF CAMPINAS

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 specific complex algorithms. Seismic operators stand out as a class of algorithms that can exhibit highly variable and hard-to-predict memory requirements. Although schedulers usually apply a chunking strategy to split the data among workers, the memory usage of such operators may be even more significant than the input data since they can hold intermediate results during execution.

However, determining the optimal chunk size remains challenging, 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 significantly 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 particular use cases that could be more easily generalizable. Since memory usage is relevant for this class of algorithms, this study aims to develop a chunk optimization technique using a reinforcement learning model capable of predicting memory usage across a broader range of scenarios.

Moreover, this research will explore the relationship between memory usage and parallel processing efficiency. This knowledge will benefit the processing of seismic operators and extend to other algorithms with similar characteristics. Ultimately, the 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 [1]. Some problems, such as seismic processing, are susceptible 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 in the memory while processing it. Hence, this kind of data is usually partitioned and processed in chunks.

Choosing the right data partitioning strategy for some algorithms is relatively straightforward since some frameworks, like Dask [2], provide automatic chunking. On the other hand, the optimal strategy could be more straightforward 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 input data size 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.

While considering the current research approaches, most focus on adding a new component to the scheduler to predict memory usage and properly allocate cluster resources. Although some *High Performance Computing* (HPC) algorithms usually rely on a scheduler to orchestrate the execution of the tasks, it is not possible to use the existing tools to optimize the data partitioning prior to the execution of the algorithm.

This research proposal suggests the creation of an efficient data partitioning strategy for data parallelism. The proposed strategy is a machine-learning model based on the algorithm's memory footprint. By using reinforcement learning, this model can optimize the chunk size for a given amount of available memory during runtime. This approach may be practical not only for seismic algorithms but also for any other algorithm that has predictable memory usage.

The expected result of this research is to implement a model capable of discovering the relationship between the input data shape and the algorithm's memory footprint. This tool can be used by frameworks like Dask [2] to enhance their automatic chunking strategy, leading to more efficient data partitioning.

The organization of the following sections is as follows: section 2 presents relevant background concepts for this research; section 3 discusses the existing research on memory usage estimation; finally, section 4 presents the research proposal.

2 Background

This section presents and discusses all relevant background knowledge needed to understand the rest of the proposal. Each subsection will present a specific topic, giving a general overview and discussing its relevance to the project.

2.1 Seismic attributes

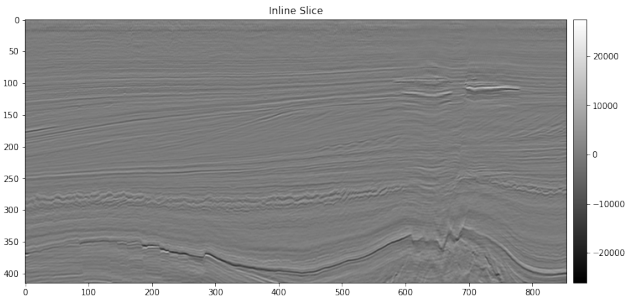
Seismic attributes are quantitative measures derived from seismic data describing various subsurface geology aspects. Over the past few decades, seismic attributes have become indispensable tools in the exploration and production of hydrocarbons, mineral resources, and the management of geohazards. They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

Seismic data, obtained through 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 subsurface geology's complex nature and seismic imaging techniques' limitations.

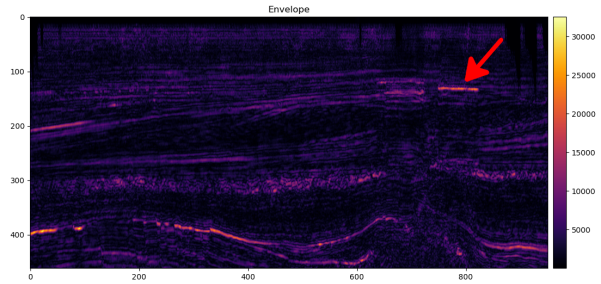
The evolution of seismic attributes is explored by Barnes and Arthur [3], Chopra *et al.* [4], Fawad *et al.* [5], and Taner and Turhan [6]. This concept was first introduced in the late 1970s to enhance the interpretation of seismic data. Since then, the field has experienced significant advancements,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

To illustrate the usage of seismic attributes, Figure 1 shows a sample of two complex seismic attributes derived from the amplitude of a sample seismic data. The amplitude envelope, illustrated by Figure 1b, highlights the presence of a potential hydrocarbon reservoir. Figure 1c shows the instantaneous phase of the seismic data, which defines the phase of the seismic wavelet at each sample.

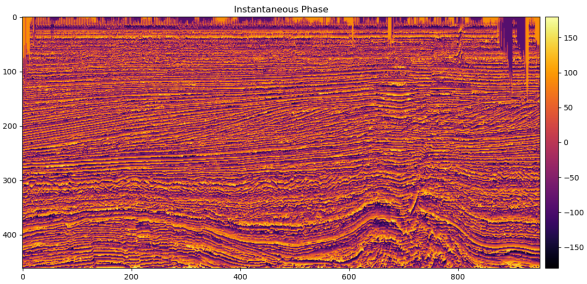
As explained in the previous example, integrating seismic attributes into the interpretation workflow allows geoscientists to extract valuable information from seismic data more efficiently. Seismic attributes can be combined with other data, such as logs and geological models, to understand subsurface geology comprehensively. Furthermore, advanced machine learning and data analytics techniques have enabled the development of multi-attribute analysis, which involves the



(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

simultaneous examination of multiple attributes to identify patterns and relationships that may not be apparent when considering individual attributes.

Seismic attributes play a crucial role in analyzing and interpreting 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, integrating 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.

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.

On the other hand, *dynamic memory allocation* 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.

To achieve a deeper understanding of the memory footprint of an algorithm, it is possible to evaluate the following components:

- *Space Complexity:* Measures the total amount of memory an algorithm uses as a function of its input size. This evaluation usually uses the big-O notation, such as $O(n)$ or $O(n^2)$, where n is the input size. It is possible to split space complexity into two subtypes: (i) auxiliary space (temporary memory used during execution) and (ii) input space (memory needed for storing input data);

- *Data Structures:* An algorithm's choice of data structures 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 substantially impact 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

Dask [2] is a powerful and flexible parallel computing framework designed for the Python ecosystem. It enables users to harness the power of parallel and distributed computing to scale their applications and data processing tasks. Dask [2] focus on addressing the limitations of standard Python libraries like NumPy [7] and Pandas [8], which struggle to handle large-scale datasets due to their in-memory computation model. It addresses these limitations by providing parallelized and out-of-core computation capabilities.

Dask [2] works by breaking down large-scale tasks into smaller, manageable tasks that can be executed in parallel across multiple cores or even distributed across multiple machines. It builds a task graph, which visually represents the tasks and their dependencies, allowing for efficient scheduling and execution of tasks. Dask [2] can automatically manage and distribute these tasks, making it easier for users to scale their applications.

Some of the critical features of Dask [2] are: (i) parallelism that can leverage the available hardware resources, (ii) out-of-core computation, which allows handling datasets that are too large to fit into memory, (iii) dynamic task scheduling, and (iv) integration with existing Python libraries.

Code sample 1 shows an example of how Dask [2] can be used to parallelize NumPy [7] operations, while code sample 2 shows how Dask [2] can be used to parallelize Pandas [8] operations.

```
import dask.array as da

# Create a large Dask array
x = da.random.random((10000, 10000), chunks=(1000, 1000))

# Perform element-wise operations in parallel
y = x + x.T

# Compute the result
result = y.compute()
```

Listing 1: Parallelizing NumPy [7] operations

```
import dask.dataframe as dd

# Read a large CSV file in chunks
df = dd.read_csv("large_data.csv", blocksize="64MB")

# Perform parallel groupby and aggregation operations
```

```
result = df.groupby("column_name").agg({"other_column": "sum"})

# Compute the result
result = result.compute()
```

Listing 2: Parallelizing Pandas [8] operations

2.4 Reinforcement Learning

Reinforcement learning is a branch of machine learning that focuses on training intelligent agents to make decisions based on the consequences of their actions in a given environment [9]. It has emerged as a powerful method for solving complex problems in various fields.

In reinforcement learning, the model, known as an agent, interacts with an environment to learn an optimal policy for achieving its goals. The agent selects actions based on their current state and receives feedback from the environment through rewards or penalties. This feedback guides the agent's learning process, allowing it to improve its decision-making abilities over time.

The central components of a reinforcement learning system include:

- *Agent:* The intelligent entity that learns and makes decisions based on its interactions with the environment;
- *Environment:* The context or world within which the agent operates, offering various states, actions, and feedback;
- *State:* The representation of the agent's current situation within the environment;
- *Action:* The possible moves or decisions the agent can make within the environment;
- *Reward:* The feedback the environment provides indicates the success or failure of an agent's actions.

Reinforcement learning algorithms can be broadly categorized into model-free and model-based methods, as well as on and off-policy methods:

On *model-free methods*, the agent learns directly from its interactions with the environment without knowing the underlying dynamics. The agent learns to map states to actions based on trial and error. Examples of model-free methods include Q-learning [10], SARSA [11], and *Deep Q-Networks* (DQNs) [12]. On the other hand, on *model-based methods*, the agent learns a model

of the environment, which captures the dynamics of state transitions and rewards. This model is then used to plan and select optimal actions. Techniques employed in model-based methods include Monte Carlo Tree Search [13], Dynamic Programming, and various planning algorithms.

Regarding policy methods, *on-policy methods* involve the agent learning the value of its current policy while following it. The agent continually updates its policy based on the experience gained through its actions. On-policy algorithms include SARSA [11], which learns the action-value function for the current policy, and actor-critic methods that learn both a value function and a policy. In contrast to on-policy methods, *off-policy methods* enable the agent to learn the optimal policy while following another policy. This approach provides more flexibility, as the agent can learn from the experiences of other agents or explore different policies simultaneously. Q-learning [10] and *Deep Deterministic Policy Gradient* (DDPG) [14] are examples of off-policy algorithms.

According to Sutton and Barto [15], the critical steps for the process of reinforcement learning are: (i) initialization, when the agent starts in an initial state and initializes its learning parameters, such as the action-value function or policy; (ii) action selection, when the agent chooses an action based on its current policy or action-value function, often incorporating exploration strategies like epsilon-greedy or softmax action selection; (iii) environment interaction, when the agent performs the chosen action, and the environment responds by providing a reward and transitioning to a new state; (iv) learning, when the agent updates its knowledge based on the experience gained from the interaction, adjusting its policy or action-value function; (v) repetition, steps ii through iv are repeated until a termination condition is met, such as reaching a maximum number of steps, achieving a certain level of performance, or converging on an optimal policy.

Despite the promising capabilities of reinforcement learning, challenges and limitations exist, such as exploration vs. exploitation trade-offs, sparse rewards, and sample efficiency. Balancing exploration (trying new actions) with exploitation (selecting known optimal actions) is crucial for an agent to learn effectively. Additionally, agents must learn to cope with environments where rewards are infrequent or delayed, which can slow down the learning process.

Another challenge in reinforcement learning is dealing with the curse of dimensionality [16]. As the state and action spaces become more complex, the time and computational resources required for learning can grow exponentially. Researchers have developed various techniques to tackle this issue, such as function approximation, state aggregation, and hierarchical reinforcement learning.

Reinforcement learning offers a robust framework for training intelligent agents to make decisions in complex environments. Despite the challenges and limitations, ongoing research pushes

the boundaries of what reinforcement learning can achieve, paving the way for its application in various fields and real-world problems.

3 Related Work

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

This section compares and contrasts the current state-of-the-art of each existing approach, evaluating the main differences between the proposed work and the existing research. Finally, by the end of this section, table 1 summarises the main findings.

3.1 Resource-aware scheduling

Pupykina and Agosta [1] present a broad overview of the memory management field until 2019. According to the authors, the main challenge in predicting memory usage is knowing the memory access patterns within an application. Most current research focuses on using machine learning techniques to bypass this problem. Although this approach has been successful, it can only predict the overall resource usage of a whole workload rather than the resource usage of a single task.

The observation made by Pupykina and Agosta [1] is visible while evaluating recent work. Most of the research done so far focuses only on the scheduler perspective, using memory usage history to predict the expected resource requirements of a given cluster.

E. R. Rodrigues *et al.* [17] present 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 mainly focuses 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 scheduler, the user must provide a manual estimate. The authors use this estimate, alongside past executions, to predict the actual resource requirements of the job. Although effective, this approach is vulnerable to spurious correlations since past executions by the same user from other jobs can influence the prediction.

T. Mehmood *et al.* [18] present an ensemble machine learning model that can predict the

expected resource usage of a cloud provider. That estimate is calculated based on the resource usage of recent tasks submitted to that provider. Like E. R. Rodrigues *et al.* [17], T. Mehmood *et al.* [18] 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.* [18] uses a dataset provided by Google containing the trace data of many jobs executed on the Google Cloud Platform.

Similarly to both T. Mehmood *et al.* [18] and E. R. Rodrigues *et al.* [17], Phung *et al.* [19] propose an approach to use past executions to achieve a smaller upper-bound on resource allocation. To do so, Phung *et al.* [19] 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 failing.

On the other hand, Fang, Wang, and Sun [20] take a different approach. Instead of using historical data to predict the exact amount of resources required, the authors use a machine learning model to predict the current memory pressure of a given cluster. Their research is coupled with the Hadoop [21] framework and analyzes the status of all active jobs before submitting a new one on the same cluster. Instead of predicting the resource requirements of a single job, this approach focuses on predicting the cluster's overall performance.

All the research discussed so far focus on handling proper resource allocation within the scheduler. Although effective, they all require training with historical data of incoming jobs. Both the required amount of historical data and the focus on resource allocation based on a heterogeneous pool of incoming jobs limit the usage of those approaches to large-scale clusters. Using those approaches to predict the memory requirements of a single job is counterproductive since each job would require a different training dataset.

Considering this limitation, Ferreira da Silva *et al.* [22] 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 high accuracy, but the results vary a lot based on the size and density of the input itself.

During their research, Ferreira da Silva *et al.* [22] discovered that smaller datasets have a higher correlation rate between the parameters the clustering algorithm extracts and the resource consumption itself. This observation is essential since it shows that it is possible to use input data features to predict a given job's resource requirements. As the final goal of their research, Ferreira da Silva *et al.* [22] created an online estimator tool designed explicitly to be used by schedulers to improve their scheduling process.

Like Ferreira da Silva *et al.* [22], B. T. Shealy *et al.* [23] try to extract features from the

components of the execution in order to predict resource consumption. Instead of gathering features only from the input data, B. T. Shealy *et al.* [23] 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 can predict the resource consumption of jobs. The main limitation of this approach is the requirement of building a training dataset for each algorithm that the user wants to predict resource consumption. Due to this fact, this approach is only suitable for the recurrent execution of specific algorithms, changing only the input data and a few run parameters.

Finally, A. V. Goponenko *et al.* [24] focuses on a different perspective. Their work proves resource-aware scheduling can be more effective than traditional scheduling algorithms. They have integrated their tool into Slurm, considering resource requirements while scheduling new jobs. During the research, the authors discovered that their test workload executed 9.4% faster than the original scheduling, with more efficient usage of the cluster resources.

3.2 Resource-aware execution

D. Duplyakin *et al.* [25] take a different approach to evaluate the resource usage of a given algorithm. Instead of using the execution history to predict the expected requirements to handle resource allocation efficiently, they try to incrementally model the memory usage of the algorithm using an active learning approach. Their discovery is essential since they successfully demonstrated that it is possible to integrate active learning with Gaussian process regression to explore the resource usage of a given algorithm. However, their approach is specific to their use case, but a more general approach can explore the same concept.

Following a similar perspective, C. Tang *et al.* [26] proposes a method to predict the resource usage of a given query. Their approach was developed inside Twitter [1] and aimed to simplify calculating the resource usage of SQL queries. During their research, the authors extracted keywords and features directly from the query itself and used those features to increase the prediction accuracy. With that method, they achieved 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 may work in other programming languages.

As demonstrated by D. Duplyakin *et al.* [25] and C. Tang *et al.* [26], it is possible to predict the resource usage of a single algorithm both by exploring critical aspects of the application, as

[1] https://twitter.com/

well as by dynamically exploring unknown executions. However, both approaches are limited to a specific use case and 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**
[17]	Memory usage history, user data, and job data	Past executions	Scheduler resource allocation	Scheduler
[18]	Memory usage history and job data	Past executions	Scheduler resource allocation	Scheduler
[19]	Resources being used and user-provided information	Trial and error	Scheduler resource allocation	Scheduler
[20]	Resources being used and memory pressure	Resource competition	Scheduler resource allocation	Scheduler
[22]	Input data parameters	Extensive application profiling	Efficient task scheduling	Scheduler
[23]	Memory usage history, input structure, and eun command flags	Past executions and code instrumentation	Recurrent task scheduling	Scheduler
[24]	Memory usage estimate	None	Efficient task scheduling	Scheduler
[25]	Input data and hardware characteristics	None	Predict resource consumption	Algorithm
[26]	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

This research proposal aims to optimize the data parallelism process by automatically updating the chunk size during the execution of an algorithm. The primary strategy during optimization is evaluating the algorithm's memory footprint and fitting the predicted memory usage with the current chunk size.

As seen in section 3, while there have been numerous studies on historical analysis of memory usage to predict resource requirements, they are mainly from the scheduler perspective. There is a significant gap in research regarding predicting the memory usage of a single algorithm, primarily focusing on data parallelism optimization.

4.1 Problem statement

The Discovery [2] laboratory, located at *University of Campinas* (UNICAMP) [3], is working on a seismic analysis project with Petrobras [4]. This project aims at developing a framework called *DASF is an Accelerated and Scalable Framework* (DASF) [27], 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 is massive datasets that can contain 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 [27] has a parameter called "block size." With the value of that parameter, DASF [27] uses Dask's [2] automatic chunking feature to split the dataset into chunks. However, setting this parameter can be challenging because it requires finding the optimal relationship between it and the network overhead caused by it.

To illustrate this challenge, image 2 presents three computing graphs receiving input data from a seismic dataset. The first scenario (on the left) 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 and fails due to lack of memory. The second scenario (in the middle) divides the data into thousands of small parts, reducing the memory requirements but adding network and scheduler overhead. The third scenario (on the right) divides the data into smaller parts, minimizing network and memory requirements.

While executing the graph, the developer must manually set the block size parameter. Setting a large value may lead to memory issues and cause significant delays due to the trial-and-error nature of the execution flow. Since Petrobras uses supercomputers to execute those graphs, this delay may be even more significant considering the time it takes to submit a job due to the queue waiting time.

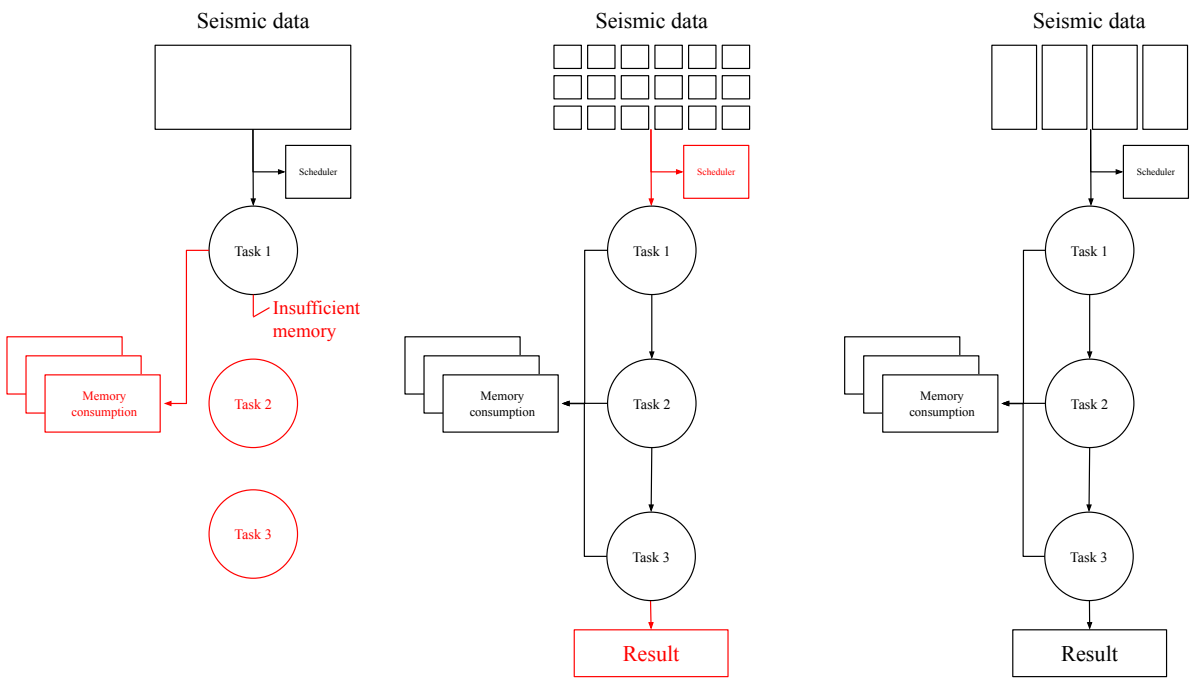
On the other hand, setting a small block size value may increase the execution time due to network and scheduler overhead. Since Petrobras has many graphs to execute, and the complete execution of each usually takes a long time, optimizing the block size parameter may lead to considerable efficiency improvements by reducing the number of failed jobs on their clusters and removing the need to tune that parameter manually.

[2] `https://discovery.ic.unicamp.br/`

[3] `https://ic.unicamp.br/`

[4] `https://petrobras.com.br/`

Figure 2: Block size impact on memory and network usage



Dask [2] 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 do not require a large working memory is easy since the developer can set that to a percentage of the available memory. However, some of the seismic operators used by Petrobras generate 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 [2] auto chunking feature to split the data into the ideal number of chunks automatically. Since DASF [27] uses Dask [2] under the hood, the block size parameter on DASF [27] is equivalent to the chunk size parameter on Dask [2]. 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, this research aims to create a DASF [27] 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. That plugin will help DASF [27] users 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

developing a more efficient data partitioning strategy to execute a graph in large-scale clusters.

4.2 Proposed solution

Most seismic operators are tensorial algorithms that execute calculations for each part of the input data. Due to this fact, it is safe to assume that there might be a relationship between the memory usage of an algorithm and the shape of the input itself. This research plans to create a reinforcement learning model that can optimize the chunking process by using the algorithm's memory footprint and adjusting the chunk size during the execution of the graph.

There are two different ways to implement the proposed solution:

1. **Algorithm-specific model:** create a separate model for each algorithm, considering its input parameters, input shape, and size as the primary features to describe the current state for the optimization;

2. **Generic algorithm model:** create a single model capable of being algorithm-agnostic, considering the source code as a feature to describe the state during optimization.

Although creating a generic algorithm model is more flexible, coding an algorithm-specific model will allow us to explore all possible limitations in a more controlled environment. While coding the model, one of the first challenges is exploring which features may describe the current environment state for the optimization. As a first hypothesis, the input data's shape and size may be the most relevant features since a large memory footprint is usually related to storing intermediate results.

Depending on the results of the initial experiments to create an algorithm-specific model, it is possible to expand the scope to a generic algorithm model. In this scenario, the algorithm's source code may contain relevant information, such as how the code author deals with memory management. However, it needs to be clarified how to extract features from the source code and use them to describe the current state during execution.

From a practical perspective, the reinforcement learning model will be responsible for the data partitioning process on DASF [27]. The goal is to create a plugin that can automatically set the optimal block size parameter during execution, updating the value based on the current state of the algorithm.

4.3 Potential risks and limitations

This section will discuss the limitations of the proposed solution considering the current state of the research.

4.3.1 Memory usage variance based on the input data

Some algorithms may have memory usage that varies depending on the input data. Correlating features from the input data may not be possible if the algorithm structure contains control statements that drastically change the memory usage of the algorithm.

It is essential to mention that this limitation only happens if the control statement directly affects memory allocation. 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 used are tensorial algorithms, meaning their execution flow would not vary based on the input data. Therefore, this limitation may not affect the research itself.

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 this statement is still pending validation through experiments.

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 currently focuses on using Python since it is the language used for the seismic operators and Dask [2] itself.

Although Python is a popular language in the scientific community, there may be other languages of choice for some researchers. This solution may need to be more language-agnostic at the moment. However, it is possible to improve the solution to accept algorithms from any language in the future.

In the case of the algorithm-specific model, it is possible to improve the training structure to

accept algorithms from any language by decoupling both the feature extraction and the execution of the algorithm from the training process.

In the case of the generic algorithm model, it is possible to 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 [27] uses Dask [2] under the hood, it is crucial to understand how Dask [2] manages memory. Considering all experiments conducted so far, the library has an automatic chunking feature that executes the proper data partitioning as long as the developer can define the maximum chunk size in bytes.

This feature seems promising and may solve most of the challenges related to automatic data partitioning. Nevertheless, conducting proper experiments to understand exactly how this feature works and its limitations is essential. Depending on the results of such experiments, the proposed solution may change to work with Dask's [2] memory management.

4.4 Problems to be addressed

This section outlines the potential problems during the research and how to address them. Each subsection represents a problem, with a brief description of the problem and how to address it.

4.4.1 How to measure memory usage

One of the primary challenges is accurately measuring memory usage. This critical feature that describes the current state of the environment is necessary during the execution of the reinforcement learning model. Considering the usage of GPU memory, this problem may be even more complicated since it would require strategies that may differ from CPU memory.

Addressing this issue involves investigating if there is an API to measure GPU memory consumption and, if not, exploring standard libraries that allow gathering memory consumption based on a specific process.

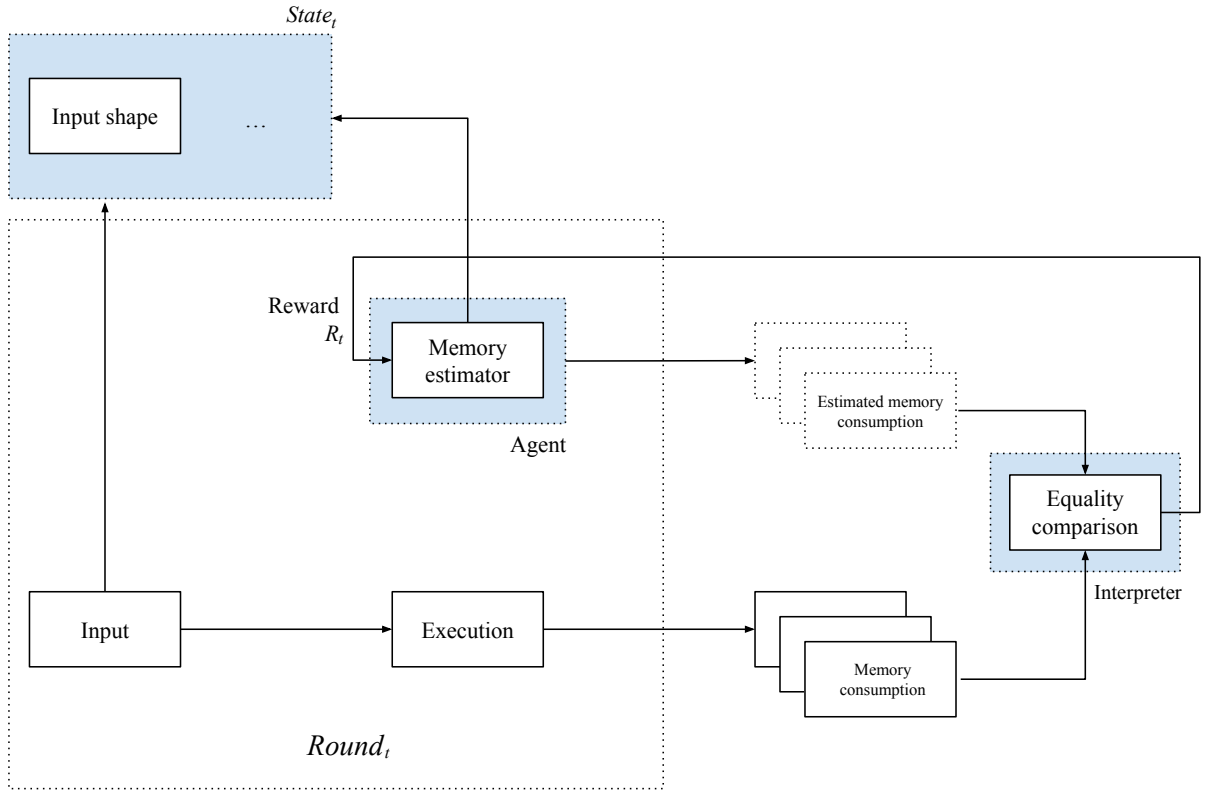
4.4.2 Historical data requirements

Many current approaches require significant historical data to train the models. Those are not feasible for this research since each execution graph would require a different data set to train the models.

Using a reinforcement learning approach will allow us to overcome this limitation. This approach will allow the model to learn from the data during execution, learning from its mistakes and improving its predictions. Figure 3 shows a diagram illustrating how this approach will work. The agent will evaluate the current state, considering both input-related features as well as memory usage since the algorithm started, and will take an action that could be: (i) *continue* to execute the algorithm with the current chunk size; (ii) *rechunk* the data to a different chunk size; or (iii) *reset* the execution of the algorithm with a different initial chunk size.

The interpreter will evaluate the agent's actions and provide a reward based on memory usage efficiency.

Figure 3: Reinforcement learning execution diagram



4.4.3 Python's garbage collection

Python's garbage collector can pose a problem for this research. Python uses reference counting as its garbage collection strategy. It is lazy and usually waits for the necessary memory to collect the garbage and free memory space. If the operators are CPU memory bounded, it would be necessary to figure out how to deal with Python's garbage collector to gather memory usage when collecting memory consumption information.

4.4.4 Graph execution

Another possible problem is figuring out the entire graph's memory requirements. While the proposed solution allows controlling the chunk size of a specific algorithm, integrating multiple algorithms into a graph poses a challenge. Nevertheless, it is crucial to understand how to abstract the model agent to persist the state during the graph execution.

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 [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

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 [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).

4.5.2 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 [27] 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 [27] 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 [27] 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.

Table 3: Work plan and schedule with dates

Activity	2023								2024				
	Mar	Jun	Jul	Ago	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													

References

[1] 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.

[2] M. Rocklin *et al.*, *Dask*, https://github.com/dask/dask, 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.

[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] C. R. Harris, K. J. Millman, S. J. van der Walt, *et al.*, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. DOI: 10.1038/s41586-020-2649-2. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2.

[8] T. pandas development team, *Pandas-dev/pandas: Pandas*, version latest, Feb. 2020. DOI: 10.5281/zenodo.3509134. [Online]. Available: https://doi.org/10.5281/zenodo.3509134.

[9] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *J. Artif. Int. Res.*, vol. 4, no. 1, pp. 237–285, May 1996, ISSN: 1076-9757.

[10] C. J. C. H. Watkins and P. Dayan, "Technical note: Q -learning," *Mach. Learn.*, vol. 8, no. 3–4, pp. 279–292, May 1992, ISSN: 0885-6125. DOI: 10.1007/BF00992698. [Online]. Available: https://doi.org/10.1007/BF00992698.

[11] G. Rummery and M. Niranjan, "On-line q-learning using connectionist systems," *Technical Report CUED/F-INFENG/TR 166*, Nov. 1994.

[12] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015, ISSN: 1476-4687. DOI: 10.1038/nature14236. [Online]. Available: https://doi.org/10.1038/nature14236.

[13] C. B. Browne, E. Powley, D. Whitehouse, *et al.*, "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, 2012. DOI: 10.1109/TCIAIG.2012.2186810.

[14] X. Wang, S. Wang, X. Liang, *et al.*, "Deep reinforcement learning: A survey," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–15, 2022. DOI: 10.1109/TNNLS.2022.3207346.

[15] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, Second. The MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html.

[16] R. Weber, H.-J. Schek, and S. Blott, "A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces," in *Proceedings of the 24rd International Conference on Very Large Data Bases*, ser. VLDB '98, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998, pp. 194–205, ISBN: 1558605665.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

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

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] J. Faracco, O. Napoli, and E. Borin, *Dasf is an accelerated and scalable framework*, https://github.com/discovery-unicamp/dasf-core, 2023.