Universidade Federal do Rio Grande do Norte
Centro de Tecnologia
Programa de Pós-Graduação em Engenharia Elétrica e de Computação

Fault Tolerance techniques applied to geophysical numerical methods

Carla dos Santos Santana

Advisor: Prof. Dr. Samuel Xavier de Souza

Co-advisor: Prof. Dr. Claude Tadonki

Co-advisor: Prof. Dr. Hervé Chauris

Ph.D. Qualification presented to the Graduate Program in Electrical and Computer Engineering of UFRN (area of concentration: Computer Engineering) as part of the requirements for obtaining the Doctorate of Science degree.

Resumo

A computação de alto desempenho (HPC do inglês *High-Performance Computing*) vem crescendo e proporcionado o estudo de problemas que envolvem diversos cálculos e uma quantidade significativa de dados (como métodos geofísicos) em um tempo de execução viável. Um dos principais objetivos de uma aplicação com HPC é que seja escalável. Em outras palavras, manter o desempenho da aplicação quando aumentado o número de nós.

A busca por escalabilidade traz problema porque cada nó fornece um determinado tempo médio entre falhas (MTBF), portanto, quanto mais nós são usados, mais elevadas são as probabilidades de falha. Para uma aplicação, que requer computação significativa, ser resiliente é uma característica essencial, ou seja, lidar com falhas para que execute corretamente em ambientes de computação de alto desempenho mais propensos a falhas (HERAULT; ROBERT, 2015).

HPC tem sido empregado em métodos geofísicos para algoritmos com alta complexidade computacional, como a Inversão Completa da Forma de onda (FWI, do inglês *Full Waveform Inversion*). A FWI mede o modelo de velocidade de propagação da onda sísmica a partir da diferença entre os dados observados e modelados artificialmente (VIRIEUX; OPERTO, 2009). Uma falha em um subconjunto de nós pode causar uma falha irrecuperável no FWI, o que pode produzir um impacto financeiro significativo, pois pode levar vários dias ou semanas para recalcular os dados perdidos.

O FWI precisa de uma técnica de tolerância a falhas (FT do inglês *Fault Tolerance*) para ter a capacidade de continuar rodando mesmo em caso de falhas (KHAITAN, 2016). Sistemas totalmente confiáveis são impossíveis porque as falhas são inevitáveis. No entanto, podemos reduzir as consequências das falhas aplicando técnicas adequadas (WEBER, 2003). Este trabalho pesquisa técnicas de FT (exemplo: replicação, *checkpointing*, *heartbeat message* e *watchdog*) para desenvolvimento de uma biblioteca de tolerância a falhas adequada para problemas geofisicos para ambientes de HPC.

Palavras-chaves: tolerância a falhas, detecção de falhas, replicação, *checkpointing*, *heartbeat message*, *watchdog*, computação de alto desempenho, inversão completa da forma de onda.

Abstract

High-performance computing (HPC) has increased and provided the study of problems that involve several calculations and a significant amount of data (as geophysical methods) in a viable runtime. One of the primary purposes of a problem that applies HPC techniques is scalability. In other words, the application should maintain the same performance when the number of nodes grows.

Scalability is also a problem because each node provides a particular mean time between failures (MTBF) therefore, the more nodes are used, the more elevated are the probabilities of failure. An application that requires significant computation to be resilient is an essential feature, and dealing with faults is critical to running it in some high-performance computing environments (HERAULT; ROBERT, 2015).

HPC has been employed in geophysical methods for algorithms with high computational complexity, like Full waveform inversion (FWI). This method measures the seismic wave propagation velocity model from the difference between observed and artificially modeled data (VIRIEUX; OPERTO, 2009). A failure in a subset of those nodes may cause an unrecoverable failure in FWI, which can produce a meaningful financial impact as it may take several days or weeks to recompute the lost data.

The FWI needs a fault tolerance (FT) technique to have the ability to continue running even in the event of faults (KHAITAN, 2016). Completely reliable systems are impossible because failures are inevitable. However, we can reduce the failures' consequences by applying suitable techniques (WEBER, 2003). This work researches FT techniques (as replication, checkpointing, heartbeat message, and watchdog) to develop a fault tolerance library proper for geophysical problems for HPC environments.

Keywords: fault tolerance, fault detection, replication, checkpointing, heartbeat message, watchdog, high-performance computing, full-waveform inversion.

Contents

List of Figures

List of Tables

List of Abbreviations and Acronyms

DS	*Decentralized Static*
FWI	*Full Waveform Inversion*
GC	*Global checkpoint*
HPC	*High-performance Computing*
LC	*Local checkpoint*
MTBF	*Mean Time Between Failures*
TBF	*Time between failures*
TFF	*Time to failure*

Chapter 1

Introduction

High-performance computing (HPC) has progressed and provided the research of problems that involve several calculations and data, for example, geophysical methods. One of the principal objects of the HPC is scalability. In other words, when we grow the number of processors, the program should continue to have an excellent performance.

Each component provides a particular mean time between failures (MTBF) . Consequently, scalability is also a problem because the more component is employed, the more chance of failure. Components can fail at any time resulting in failure harming applications that are running it; because of this, methods to avoid the failure have been developed to supercomputing applications (TREASTER, 2005). These methods are classified as fault tolerance (FT), which means the system's ability continues to work even in the presence of faults. The main objective of fault tolerance is to provide to the application the capability to be trusted (AVIZIENIS et al., 2004).

Geophysical methods employ HPC because of its high computational complexity and volume of data, and a failure in its execution can produce a financial influence as it may take time to recompute the lost data. One of these techniques that have been highlighted is the Full Waveform Inversion (FWI) which measures the velocity model using numerical methods (VIRIEUX; OPERTO, 2009). FWI generates the modeled data through the seismic wave propagation's numerical solution, requiring a high computational cost for execution (ABREO-CARRILLO et al., 2015). One approach to decrease the algorithm's execution time is parallelizing some of its workloads among supercomputer's nodes.

Completely reliable systems are impossible because failures are inevitable. A node's failure can propagate the error generating the complete failure of the FWI execution. However, we can avoid these failures' consequences using some suitable techniques of FT (WEBER, 2003). The main challenge is developing an FT technique that does not cause a significant overload in FWI, and it expends considerable time and resources furthermore overlaps the optimization of the HPC method.

The development of any FT technique requires first specified which error scenario this technique will resolve. Understanding the behavior of failures, we can develop the most suitable method to tolerate them (GAINARU; CAPPELLO, 2015). In developing a proper FT technique for FWI, we can minimize the consequences of possible failures in one or more nodes.

We can use checkpointing or replication to reduce the consequences of the early termination because of failures in some nodes. Checkpointing allows the application recovery from the state before the failure (KALAISELVI; RAJARAMAN, 2000). This method saves the application's state with some frequency, and if the application fails, it is possible to restart from the last data saved. Replication allows the application continues working even after the failure of one of its processors. This method keeps copies of data at various processors; if one fails, another one with its data copied can continue its work (COULOURIS et al., 2012).

In case of a failure of more than one node, it is viable that the full or pieces of the node's data must be in more than one other node; to guarantee the correct recovery of the state before the fail. One way to implement this feature is using an error controlling code as Reed Solomon, which provides a structure to recover data even lost a piece (WICKER; BHARGAVA, 1999).

Detecting that the application fails and restarting is essential to apply techniques to save and recover the data. Fault detection can be implemented by monitoring messages from one node to another; this technique is called heartbeat monitoring (CHETAN; RANGANATHAN; CAMPBELL, 2005). Alternatively, with a subsystem to observe the application execution, this subsystem is designated watchdog (BENINGO, 2010).

FT methods have been applied for the use of preemptible cloud instances also. These instances allow the system to revoke the given resources; however, they are more economical. When the system reclaims the resources, it typically sends termination signals that allow the application to apply FT techniques before its termination (MISHRA; UMRAO; YADAV, 2018).

1.1 Qualification goals

This research proposes an FT library implemented in C++ for HPC environments. This library is planned to fitting principally in geophysical methods as FWI 3D. The main situations we are focused on solving are:

1. **One node fails, and all execution will fail after a while**: A node can fail because of some hardware or software fault, and in the moment of communication between other nodes, this failure can be propagated to the others, and all execution finishes.

2. **More than one node will fail, and all execution will fail after a while**: The same situation as the previous one, but now we are considering more than one failed node

3. **The system will stop the execution through a signal**: In supercomputers, the operating system can send a termination signal to the code in execution. This signal can be because of some fault or exceeded the runtime, or the environment is a preemptible instance.

Our goal is to implement an FT library to tolerate the previous failures.

1.2 Qualification structure

This qualification is structured as follows: Chapter 2 gives some relevant concepts to understand interprocess communication. Chapter 3 details FWI. Chapter 4 explains some relevant concepts to comprehend the FT methods. Chapter 5 presents the FT library developed until this moment. Chapter 6 shows the validation of the library and initial results. Chapter 7 presents the chronogram with the next steps.

Chapter 2

Interprocess Communication

This study concentrates on developing FT techniques applied to FWI in HPC environments. It is necessary to understand which parallelization technique the code implements to analyze and develop adequate FT techniques. The main characteristic of parallelization for distributed systems is the communication between the process. This chapter summarizes the central notion of communication between the process we focus on studying FT techniques to FWI.

2.1 Memory structure

In HPC, we can classify the approach according to the memory structure that we are using, which can be distributed or shared (Figure 2.1). In shared-memory systems, we have a main memory and any processor has access to it. Otherwise, in distributed-memory systems, each processor has its memory (PACHECO, 2011).

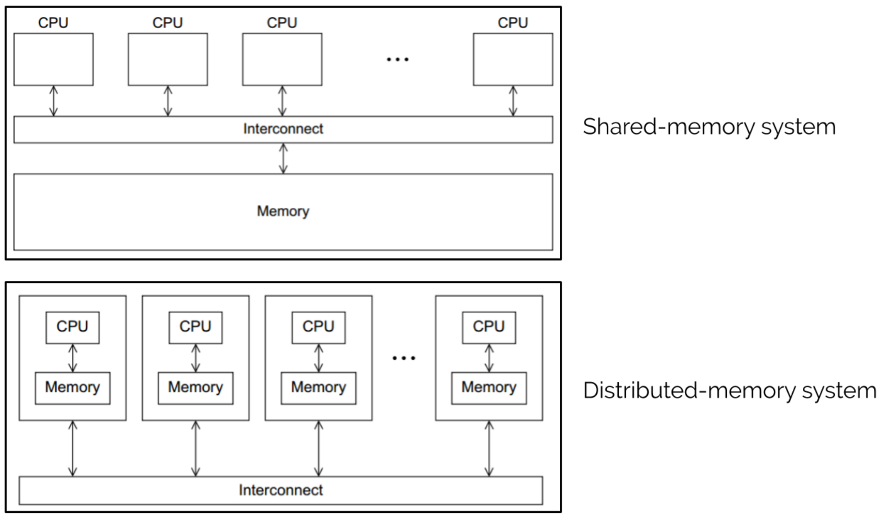


Figure 2.1: Parallel Systems
Source: (PACHECO, 2011)

In shared-memory, an instance of a program running on a processor has denominated a *thread*, and we can program with Pthreads and OpenMP. Pthreads establish a standard for programming with threads in low-level in C, and it has more control over thread management. Differently, OpenMP is higher-level and more portable and easily scaled than Pthreads (PACHECO, 2011).

In distributed-memory, the instance of a program running on a processor is called *process*, and we can program using Message-Passing Interface (MPI). The primary functions of MPI are *MPI_Send* and *MPI_Recv*. *MPI_Send* allows a process to send any data to another one, and *MPI_Recv* allows it to receive. These functions are point-to-point communication because it occurs among only two processes (PACHECO, 2011).

When we need to communicate more than two processes, we can use the collective functions as the function *MPI_Reduce* that executes an operation in a data of various processes to be received by one process. Data reduction means decreasing a collection of numbers into a smaller through a function (KENDALL; NATH; BLAND, 2019) — for example, reducing the list [5, 2, -1, -3, 6, 5, -7, 2] with the sum operation resulting in 9. When we need the result of a reduction in all processes, we apply the function *MPI_Allreduce*, which computes the reduction doing the operation and distributing the result at the same, with the communication pattern butterfly as is shown in Figure 2.2.

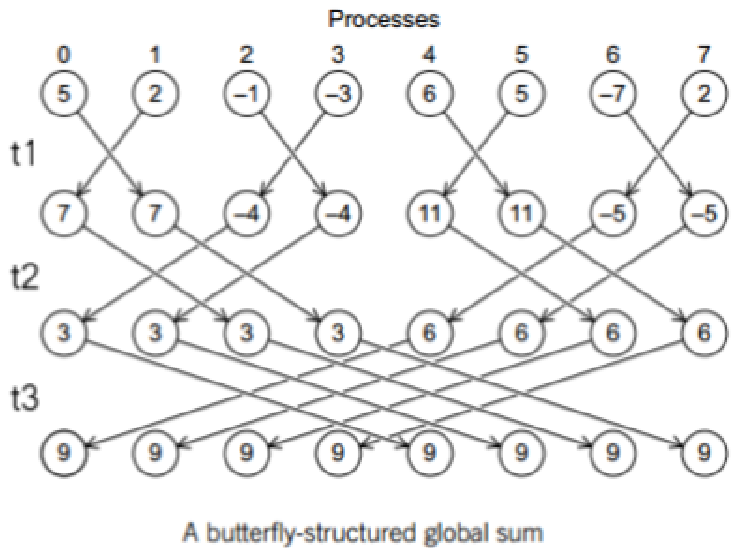


Figure 2.2: Butterfly communication
Source: (PACHECO, 2011) Adapted

2.2 Internet transport-level protocols

The communication between processes uses *socket* abstraction, in which a socket of one process sends a message to another socket in another process as illustrated in Figure 2.3. Each socket is correlated with a distinct transport protocol – either UDP or TCP (COULOURIS et al., 2012).

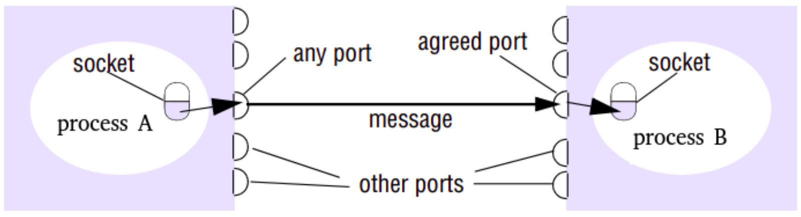


Figure 2.3: Sockets communication
Source: (COULOURIS et al., 2012) Adapted

The two main protocols in the transport layer are UDP and TCP. UDP is a connectionless protocol with no flow control and no error control. The process sends the message to another but is not sure that the message it will receive. Otherwise, UDP is an attractive strategy in some client-server situations. For example, the client sends a short request to the server and waits for a reply back. If any message is lost, the client can time out and send it again. In this scenario, there are fewer messages than with a protocol with more reliability. On the other hand, TCP is a connection-oriented protocol specifically planned to provide a reliable. If any message is lost, TCP resends them, which can cause a message overhead (TANENBAUM et al., 2003).

2.3 Metrics for analysis

2.3.1 Scalability

Scalability is a concept which represents the performance proportionally to the number of processors (RAUBER; RÜNGER, 2013). To analyze how much scale a program is, we investigate the algorithm behavior when the number of processors increases or the size of the problem increases. The program is defined as strongly scalable when the efficiency does not change when the problem size continues the same and the number of processes increases. Moreover, the program is defined as weakly scalable when the problem size and the number of processes increase in the same proportion, and the efficiency does not change (PACHECO, 2011).

2.3.2 Resilience

Hutchison and Sterbenz (2016) described resilience as the union of trustworthiness and tolerance. Trustworthiness refers to security and performability, and tolerance refers to survivability, disruption tolerance, and traffic tolerance. Laprie (2005) and Strigini (2008) append resilience as the ability of the service to deliver what it was intended for even in the face of failures, attacks, or accidents.

Chapter 3

Full Waveform Inversion

Full Waveform Inversion (FWI) is a technique based on numerical optimization methods employed to obtain quantitative information from seismograms (VIRIEUX; OPERTO, 2009). This chapter discusses the main concepts to understand this algorithm and which data is more important to save to guarantee that we can recover the FWI. The first concept discussed is the Seismic Survey in 3.1, Section 3.2 shows FWI formulation, and Section 3.3 concludes with the FWI algorithm used.

3.1 Seismic Survey

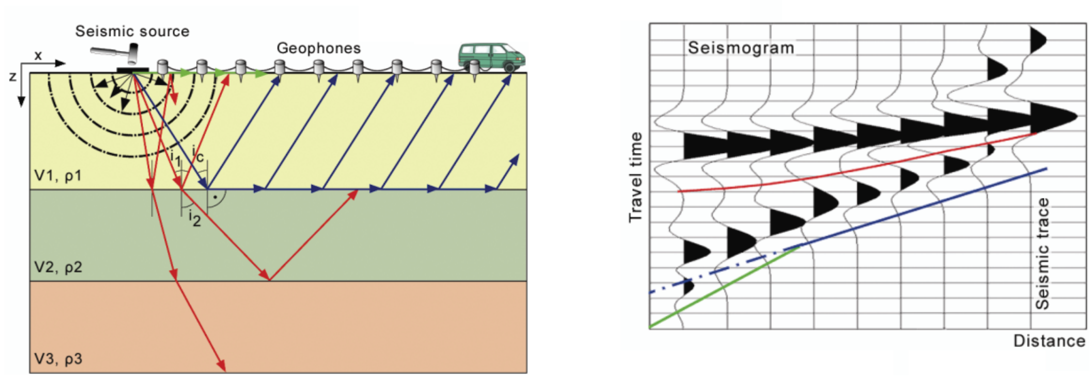
The seismic survey is a method to obtain data from the Earth's subsurface with wave propagation. This technique produces a high-energy source (also known as seismic shot) that will travel down through the Earth. If the waves find a contrast between two different geological layers, they will be reflected or refracted. The reflected waves move to the surface, and the receivers (geophone on the surface or hydrophone on the water) register them (BACON; SIMM; REDSHAW, 2007).

The receivers record the signal (seismic trace) as a time function, which expresses the combination of a seismic pulse with the response of the layered ground. The seismic trace makes it possible to extract the shot position, detector, and reflection point on the subsurface. The display of a seismic trace collection is named a seismogram (KEAREY; BROOKS; HILL, 2013).

Figure 3.1(a) is a representation of a seismic survey in a model composed of layers with different acoustic velocities (V) and densities (ρ). When the wave impacts the layer boundary, it is partially reflected (the red curves) and partially refracted (the blue curves). In Figure 3.1(b), the green curve denotes the seismic ray of the wave that travels directly, the blue curve denotes the refracted, and the red curve denotes the reflected wave. The data recorded in each geophone from the seismic trace, which all together result in the seismogram (WIEDERHOLD et al., 2006). One approach to obtain subsurface information from the seismogram is by using techniques such as FWI.

The equation that describes the propagation of the acoustic wave in 3D is:

$$\frac{1}{v^2(x,y,z)}\frac{\partial^2 p}{\partial t^2} = \frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} + \frac{\partial^2 p}{\partial z^2} + src(x,y,z), \tag{3.1}$$



Sketch (a)

Seismogram (b)

Figure 3.1: Sketch of a seismic survey
Source: Wiederhold et al. (2006)

where $v(x,y,z)$ is the acoustic velocity, x, y and z the spatial variables, p the scalar pressure field, and t is the time variable (ABREO-CARRILLO et al., 2015).

3.2 FWI Formulation

FWI is an inversion problem that tries to obtain physical parameters from measured seismic data (WANG, 2016). Figure 3.2 exemplify the inversion problem; Figure 3.2(a) has a direct problem which uses the model (velocity model) to generate data (seismogram) with an operator (acoustic wave equation). On the other hand, Figure 3.2(b) has an inversion problem that consists of obtaining information of the model only by the observed data. Usually, the inversion problem is solved with the use of numerical optimization algorithms (SILVA, 2017).

In FWI, the main objective is to determine the velocity model vector $\mathbf{v}^*$ within the equation

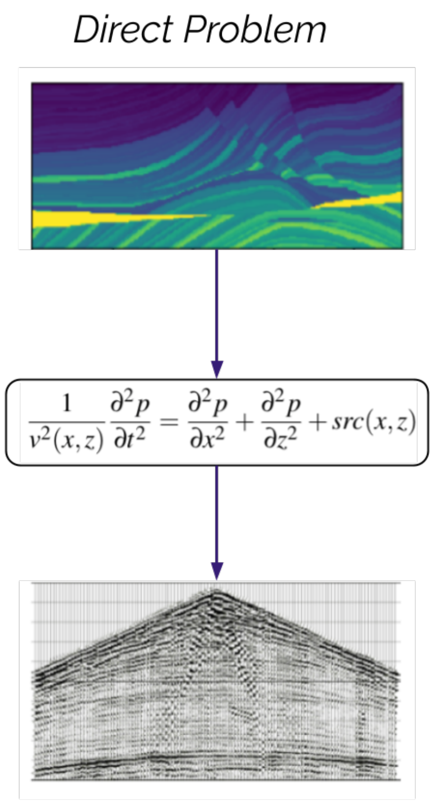
$$\mathbf{v}^* = \arg\min_{\mathbf{v}} ||\mathcal{L}(\mathbf{v}) - \mathbf{d}||_2^2, \tag{3.2}$$

where the vector $\mathbf{d}$ are the observed seismic data which is formed of several seismograms produced by different seismic shots and refers to the data obtained in the field. $\mathcal{L}(\mathbf{v})$ is the operator representing the process of artificially modeling the data with the propagation of seismic waves by numerical methods (equation (3.1)) (VIRIEUX et al., 2009).

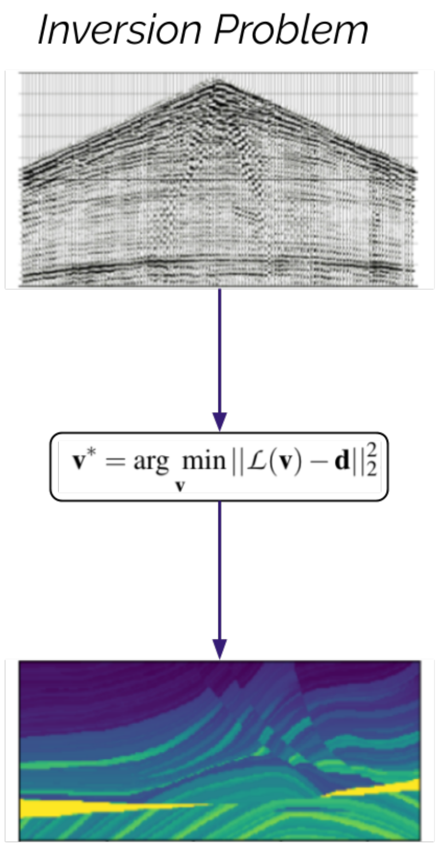
The equation (3.2) minimizes the variable f known as misfit and determined by

$$f = ||\mathcal{L}(\mathbf{v}) - \mathbf{d}||_2^2. \tag{3.3}$$

$\mathcal{L}(\mathbf{v})$ produces seismograms for a fixed velocity model. The goal is that the seismograms resulting from $\mathcal{L}(\mathbf{v})$ get more like to $\mathbf{d}$ (the observed seismograms), and, conse-


Direct Problem
$\frac{1}{v^2(x,z)}\frac{\partial^2 p}{\partial t^2} = \frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial z^2} + src(x,z)$

Direct problem (a)


Inversion Problem
$\mathbf{v}^* = \arg\min_{\mathbf{v}} ||\mathcal{L}(\mathbf{v}) - \mathbf{d}||_2^2$

Inversion problem (b)

Figure 3.2: Direct and inversion problem
Source: Silva (2017) Adapted

quently, the resulting misfit is close to zero. The velocity model used as input to the $\mathcal{L}(\mathbf{v})$ requirement be close to the one used to generate the observed data. Iteratively, this velocity model is updated, and the misfit (f) should approximate zero. At the end of FWI, the final velocity model should express the subsurface, corresponding to where obtain the observed data.

The minimization of f is solved iteratively using a numerical optimization method; the method used in this work is the quasi-Newton determined by

$$\mathbf{v}_{j+1} = \mathbf{v}_j - \alpha_j \mathbf{H}_j^{-1} \mathbf{g}_j \tag{3.4}$$

the model from the next iteration is updated using the gradient. $j \geq 0$ represents the iteration number, $\mathbf{v}_j$ is the velocity model vector at the j-th iteration, and α_j is the step size in the gradient direction, $\mathbf{g}_j$ is the gradient at the j-th iteration and $\mathbf{H}_j^{-1}$ is the approximation of the inversion of the Hessian. To calculate the $\mathbf{g}_j$, we use the adjoint state method (PLESSIX, 2006), and it is given by

$$\mathbf{g}_j = -\sum_s \frac{2}{(v_j(\mathbf{x},\mathbf{y},\mathbf{z}))^3} \int_0^T q_s(\mathbf{x},\mathbf{y},\mathbf{z},T-t) \frac{\partial^2 p_s(\mathbf{x},\mathbf{y},\mathbf{z},t)}{\partial t^2} dt \tag{3.5}$$

s corresponds to the number of the shot, q_s is the back-propagated wavefield of the residual, consisting of the reverse-time propagation of the error between the modeled and observed data (ABREO-CARRILLO et al., 2015).

To calculate the $\mathbf{H}_j^{-1}$ we use the method limited-memory bounded BFGS (L-BFGS-b) (NOCEDAL, 1980) with the library proposed by Zhu et al. (1997). The main parameters provided to L-BFGS-b are the values of the function that we need to minimize (f), the vector to be updated ($\mathbf{v}_j$), and the gradient ($\mathbf{g}_j$). This method estimates the inverse of the hessian matrix and updates the velocity model vector to the next iteration.

3.3 FWI algorithm

The notation for the algorithm is correspondingly employed to Santana (2020) because the algorithm used is similar to the FWI 2D used in that work; the main difference is that we are using FWI 3D.

3.3.1 Sequential algorithm

N_s	number of shots,
s	index for the shot number $(s = 0, \cdots, N_s - 1)$,
N_j	number of iterations,
j	index for the iteration number $(j = 0, \cdots, N_j - 1)$,
f	misfit,
f_j	misfit from the iteration j,
f_j^s	partial misfit of shot s from the iteration j,
$\mathbf{v}$	velocity model,
$\mathbf{v}_j$	velocity model from the iteration j,
$\mathbf{g}$	gradient,
$\mathbf{g}_j$	gradient from the iteration j,
$\mathbf{g}_j^s$	partial gradient of shot s from the iteration j,
$\mathcal{L}(\mathbf{v}_j^s)$	modeled data corresponding to the velocity model of the iteration j for the shot s,
$\mathbf{d}^s$	observed data of the shot s

Table 3.1: Notation for the FWI

The algorithm begins receiving the observed seismic data of each shot and the initial model v_0. We calculate the partial misfit (f_j^s) for each shot of each iteration with:

$$f_j^s = ||\mathcal{L}(\mathbf{v}_j^s) - \mathbf{d}^s||_2^2, \tag{3.6}$$

and the partial gradient ($\mathbf{g}_j^s$) for each shot of each iteration with:

$$\mathbf{g}_j^s = -\frac{2}{(v_j(\mathbf{x},\mathbf{y},\mathbf{z}))^3} \int_0^T q_s(\mathbf{x},\mathbf{y},\mathbf{z},T-t) \frac{\partial^2 p_s(\mathbf{x},\mathbf{y},\mathbf{z},t)}{\partial t^2} dt. \tag{3.7}$$

At the end of each iteration, we sum all partial gradients and misfit :

$$f_j = \sum_{s=0}^{N_s-1} f_j^s \tag{3.8}$$

$$\mathbf{g}_j = \sum_{s=0}^{N_s-1} \mathbf{g}_j^s. \tag{3.9}$$

Following, we update the velocity model with L-BFGS-b using $\mathbf{g}_j$ and f_j. If the algorithm has converged, the current velocity model is the best fit; oppositely, we executed another iteration of FWI until the convergence is reached or up to the maximum number of iterations. The main steps of the FWI algorithm utilized are in Algorithm 1.

Algorithm 1: Main steps of the FWI

```
 1: for all ($N_j$ iterations) do
 2:     for all ($N_s$ shots) do
 3:         Generate the modeled data ($\mathcal{L}(\mathbf{v}_j^s)$) with wave propagation (Eq.: 3.1)
 4:         Compute the partial misfit (Eq.: 3.6)
 5:         Compute the partial gradient (Eq.: 3.7)
 6:     end for
 7:     Obtain the total gradient (Eq.: 3.9)
 8:     Obtain the total misfit (Eq.: 3.8)
 9:     Generate $\mathbf{v}_{j+1}$ by updating $\mathbf{v}_j$ using the L-BFGS-b library
10:     if $\mathbf{v}_{j+1}$ has converged then
11:         break
12:     end if
13: end for
```

3.3.2 Parallel algorithm

To parallelize the FWI, we can distribute the tasks among the nodes in several forms, for example, centralized dynamic, decentralized static, and decentralized dynamic. To investigate the FT techniques, we are using only the decentralized static (DS) presented by Santana et al. (2019).

Each one of the N_p processes receives approximately the same quantity of shots (N_{sp}) which is defined by:

$$N_{sp} = \left\lfloor \frac{N_s}{N_p} \right\rfloor. \tag{3.10}$$

N_p	number of process,
N_{sp}	number of shots per process,
p	index for the process, the rank $(p = 0, \cdots, N_p - 1)$,
P_p	process with the rank p,
$\mathbf{g}_j^p$	gradient partial of process p from the iteration j,
f_j^p	misfit partial of process p from the iteration j,
q	average number of shots per process,
ls_p	first shot to be processed by the process p,
li_p	last shot to be processed by the process p,

Table 3.2: Notation for the DS

The first shot processed by the process p is determined by

$$li_p = p * N_{sp}, \tag{3.11}$$

and the last shot processed by the process p is determined by

$$ls_p = \left\{ \begin{array}{rl} N_{sp} * (p+1) - 1, & \text{if } p < (N_p - 1) \\ N_s - 1, & \text{otherwise} \end{array} \right\} \tag{3.12}$$

Every process calculates the misfit (f_j^p) and the gradient ($\mathbf{g}_j^p$) corresponding to their shots described by,

$$f_j^p = \sum_{s=li_p}^{ls_p} f_j^s \tag{3.13}$$

and

$$\mathbf{g}_j^p = \sum_{s=li_p}^{ls_p} \mathbf{g}_j^s. \tag{3.14}$$

At the end of the j-th iteration, the sum f_j and $\mathbf{g}_j$ are determined by

$$f_j = \sum_{k=0}^{N_p-1} f_j^k \tag{3.15}$$

and

$$\mathbf{g}_j = \sum_{k=0}^{N_p-1} \mathbf{g}_j^k. \tag{3.16}$$

We generate the $\mathbf{g}_j$ and f_j and send them to all processes through the function MPI_Allreduce. Lastly, every process measures its model $\mathbf{m}_{j+1}$, for the $j+1$-th iteration.

In addition to parallelism in distributed-memory, we applied parallelism in shared-memory using OpenMP, parallelizing the $\mathcal{L}(\mathbf{v}_j^s)$ calculation. Each thread calculates the wave propagation for a set of timesteps.

The main steps of the FWI with DS are presented in Algorithm 2.

Algorithm 2: Main steps of one process in FWI with DS

```
1: for all ($N_j$ iterations) do
2:     for $s$:=$li_p$ to $ls_p$ do
3:         Generate $\mathcal{L}(\mathbf{v}_j^s)$ using OpenMP
4:         Compute the misfit $f_j^s$
5:         Compute the gradient $\mathbf{g}_j^s$
6:         $f_j^p += f_j^s$
7:         $\mathbf{g}_j^p += \mathbf{g}_j^s$
8:     end for
9:     Obtain the total gradient with MPI_Allreduce
10:    Obtain the total misfit with MPI_Allreduce
11:    Generate the $\mathbf{v}_{j+1}$ updates the $\mathbf{v}_j$ using the L-BFGS-b library
12:    if $\mathbf{v}_{j+1}$ has converged then
13:        break
14:    end if
15: end for
```

Chapter 4

Fault Tolerance

The challenge in parallelism systems is to support an improvement in the application's performance even if any failure occurs (WEBER, 2003). The FWI 3D application requires much time to execute and generate massive data in its processing. It is inconvenient to interrupt the algorithm's execution because many important data already processed will be lost. Because of this, we investigate fault tolerance (FT) strategies that best fit the FWI.

We discuss in this chapter some essential concepts used to investigate FT in FWI. Section 4.1 presents an overview of fault tolerance in HPC environments. Section 4.2 shows concepts about failures. Section 4.3 explains replication as a way to minimize the impacts of failures. Section 4.4 discusses the technique checkpoint and rollback recovery to save the state of the application and how recovery.

4.1 Overview of fault tolerance in HPC

Supercomputers have grown with a necessary tool for developing in many areas in academic and industrial, as scientific simulations, big data analyses, and machine learning. Supercomputers technologies have become more complex to guarantee high scalability, but resilience becomes a challenge (ROJAS et al., 2020).

Individual components (as processors, disks, memories, and networks) have a high mean time before failure, but a system with numerous components may fail frequently. Reed, Mendes et al. (2006) show an example in which a system with 10000 nodes, each one with mean time to failure 10^6 hours, the system will have only mean time to failure 100 hours.

Components can fail at any time resulting in failure harming applications that are running it; because of this, methods of fault tolerance have been developed to supercomputing applications, like replication and checkpointing (TREASTER, 2005).

4.2 Faults and failures

The quantity of components in supercomputers grows faster than component reliability. Predictions display that large systems will be more error-prone; therefore, they will need to be more fault-tolerant. Understanding the behavior of failures in supercomputers is essential to develop methods to tolerate them (GAINARU; CAPPELLO, 2015). In this section, We discuss some fundamental theories used to understand the behavior of failures in supercomputers.

4.2.1 Faults Model

There are several ways to computing systems fails; these fails can be categorized into two main types of the abstract fault model: byzantine faults and fail-stop faults.

- **Byzantine faults** are faults that do not stop the execution of the node but operate incorrectly. The failed node continues communicating with the other nodes and can have arbitrary and inconsistent behavior (LAMPORT; SHOSTAK; PEASE, 1982)
- **Fail-stop faults** are faults that can happen at any time, but opposite the byzantine faults, fail-stops terminate the node, and it can not interact with the other nodes (SCHLICHTING; SCHNEIDER, 1981)

4.2.2 Definitions

The definitions that we use are the same used by Gainaru and Cappello (2015), which are the definitions proposed in Snir et al. (2014).

Faults, error and failure

The terms faults, error, and failure are sometimes utilized as synonymously, but Avizienis et al. (2004) defines distinctive:

- **Fault** is the cause of an error, as a bug.
- **Error** is a state that may cause a failure.
- **Failure** is an incorrect presentation of a service.

Snir et al. (2014) give an example to illustrate the difference among these concepts. If a wire inside a cable breaks, this crack is a **fault**. Because of this fault, a bit is generated incorrectly, and we have a **error** in the bit value. The error can generate incorrect results in the application (in this situation, we have a **failure**), or this error cannot modify the final results (no failure).

Time metrics

- **Time to failure (TFF)** is the interval between the end of the last failure and the beginning of the next failure.
- **Time between failures (TBF)** is the interval between the starts of the two following failures.
- **Mean time between failures (MTBF)**= $\frac{TotalTime}{Numberoffailures}$.

4.2.3 Modeling

Failures in supercomputers are not uniformly distributed and have periods of higher density; understanding this behavior is crucial to developing FT techniques. Failure modeling is essential to producing data with a distribution function (as exponential) and parameters (as mean and variance) that come closer to the observed data (GAINARU; CAPPELLO, 2015).

There are many probability distributions, but the most used are the exponential, Weibull, log-normal, normal, and gamma distributions in the HPC community. To analyze which one better fit for failure distribution (GAINARU; CAPPELLO, 2015) produced a study with four data sets that are the number of failures per compute node in different year intervals with four different distributions: the Weibull, log-normal, gamma, and exponential (Figure 4.1). The distribution between failures is well modeled by a Weibull or gamma distribution because in the four cases, they were the ones that came closest to the real data.

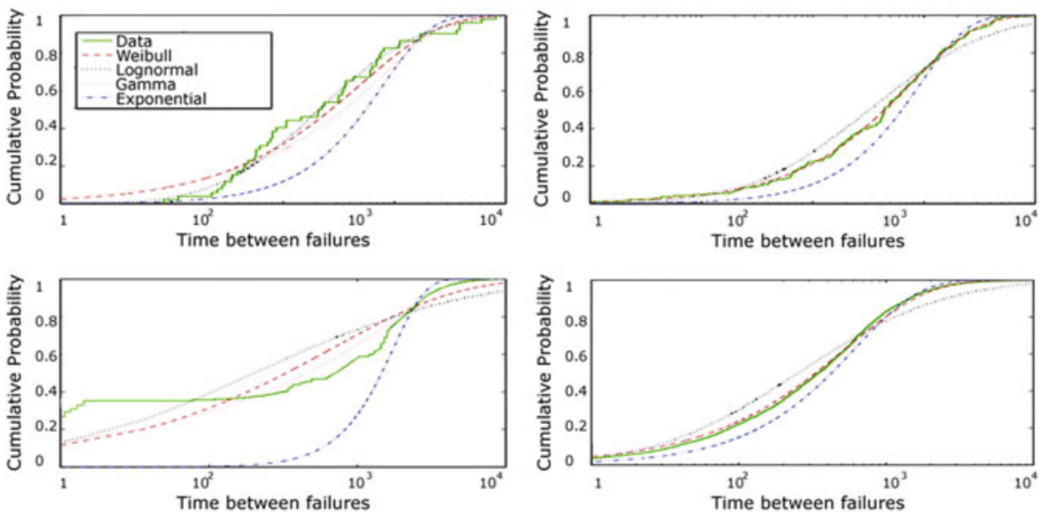


Figure 4.1: Distribution functions of failures
Source: (GAINARU; CAPPELLO, 2015)

4.2.4 Fault Detection

Heartbeat messages

The more common way to detect a fail is by monitoring the heartbeat system, in which a node sends a message periodically to the fault detector. If the fault detector does not receive the heartbeat messages in a specific time interval, it deduces that the node failed. This technique does not work well to byzantine fails because these fails continue operation even in error situations. Besides that, when using this method, it is necessary not to overload the network with heartbeat messages (CHETAN; RANGANATHAN; CAMPBELL, 2005).

Watchdog

A watchdog is a subsystem that observes the execution of an application; if a fault occurs or another unusual behavior, the watchdog can restart the application (BENINGO, 2010).

4.3 Replication

Replication of data is the most commons means to provide high availability and fault tolerance in distributed systems. The maintenance of copies of data at multiple computers is made in two principal ways: passive replication and active replication (COULOURIS et al., 2012).

Passive replication

Passive replication has a node manager, and there is one or more secondary replica managers called backups. If a node manager fails, one node backup can assume the position of the manager (COULOURIS et al., 2012).

Active replication

Active replication has more than one node manager, and they should operate identically. The purpose is to compare the results and analyze if they are consistent. It is possible to identify byzantine failures because we can compare the receives of each node manager (COULOURIS et al., 2012). This type of replication is infeasible in some situations because multiplying the number of nodes involved grows the cost of computation that it can be small (GOLDBERG et al., 2001).

4.4 Checkpoint and Rollback Recovery

Checkpoint is the technique to save the state of data necessary for a resumption of processing later. Moreover, the process of returning the execution by rolling back to a saved state is named rollback recovery (HERAULT; ROBERT, 2015).

The data is saved periodically, and if there is a failure, the application can be restarted from the last checkpoint avoiding recomputation from the beginning. In distributed systems can be made to save the state of the process (local checkpoint) or the state of all applications (collection of local checkpoints (LC) , global checkpoint (GC) (KALAISELVI; RAJARAMAN, 2000).

The main challenge in the checkpoint is the frequency and the content. The checkpointing should provide the application recover quickly and not lose much computation; this situation needs to save the state frequently and, unfortunately, generate overhead (KALAISELVI; RAJARAMAN, 2000).

The process state has to be saved in stable storage to restart the process in case of an error. The data size is a crucial point to consider, as the data cannot be so large that its read and write time exceeds the time economies caused by parallel computing.

Chapter 5

Fault Tolerance Applied to FWI

We are developing an FT library that is applied in FWI but can use in other applications. The first feature implemented that it was already generated results and published in Santana et al. (2021) was the global checkpoint explain in Section 5.1.

The other features implemented and validated are fault detection (explain in Section 5.2) and local checkpoint (explain in Section 5.3). Section 5.4 we explain all structures of the FT library and how other applications can use it.

5.1 Global checkpoint

The first FT method we employ for the 3D FWI in this work is checkpointing, saving the global state of FWI. This routine saves the code state of an iteration of FWI with a specific frequency. In case of failure, the algorithm can be initialized from the most recent checkpoint and restart the last executed iteration. We highlight two main challenges in implementing the checkpointing in FWI: which data should be saved and the frequency to save the data.

The initial step is to recognize what it is important to save to ensure that when the FWI recovers, the code will generate correct results. Our current FWI code is implemented with C++ using object-oriented programming. The principal objects used by the FWI are:

- **Optimizer**, which contains the main attributes of the L-BFGS-b;
- **Velocity Model**, which contains the main attributes of the velocity model file;
- **Adjoint**, which contains the gradient and misfit.

Consequently, each global checkpoint (GC) contains the essential information of these objects and the parameters configuration of the FWI execution, so the checkpointing global is saving:

- **Parameters of configuration** (such as the size of the velocity model, the number of iteration): To ensure that when the FWI recovery, it will recover the correct configuration data;
- **Velocity model**: to resume the optimization from the most recent model;

- **Misfit and gradient**: Not to lose the advance of the FWI;
- **Parameters of the optimizer** (such as the position of the iteration): To make sure that LBFGS can get back where it left off.

The FWI begins receiving as input the model, optimizer, and adjoint. In this case, these objects are not initialized. During each iteration, the FWI calculates the next velocity model as explained in Section 3.3. With checkpointing global, the FWI can save the global state after a specific time ($T\_CHECKPOITING\_GLOBAL$) or at the end of each iteration.

Figure 5.1 shows an example of FWI in the event of a failure. The execution without fault tolerance, the computation will start from the beginning. When we add the checkpointing global, we can restart the FWI from the last iteration reading the data and return the FWI initializing the Optimizer, Adjoint, and Velocity Model.

5.2 Fault Detection

5.2.1 Heartbeat monitoring

To check if there is a node that failed, we use the method heartbeat messages (explained in 4.2.4). From a leader node, we create a thread with Pthreds that will be responsible for receiving the heartbeat messages. The others nodes (observed nodes) will have a thread to send the heartbeat messages with a specific interval (t_{hm}) to the leader (Figure 5.2(a)). The leader checks whether the time it received the messages are within a maximum waiting time (T_MAX_WAIT). If some node does not send the heartbeat messages into the T_MAX_WAIT (Figure 5.2(b)), the leader sends a message (the trigger) to notify the other nodes that a node probably failed (Figure 5.2(c)).

The observed nodes send the heartbeat messages using the UDP protocol because they do not need to ensure that the leader receives the messages. Besides that, the T_MAX_WAIT must be more extensive than t_{hm}, so if the leader lost one message from the observed nodes, it probably would receive the others.

5.2.2 Signal Termination

These signals are all applied to tell a process to terminate at some moment; it is a mechanism to notify it. Some supercomputers and cloud systems use them to notify a job that it will be finished for some reason.

These signals are not necessarily sent because of some failure; they could also be because the application execution time exceeded the time given by the system or because the environment is preemptive and it is taking resources. When the application is fault-tolerant, it can save the essential data from guaranteeing not lost the computation done until this moment.

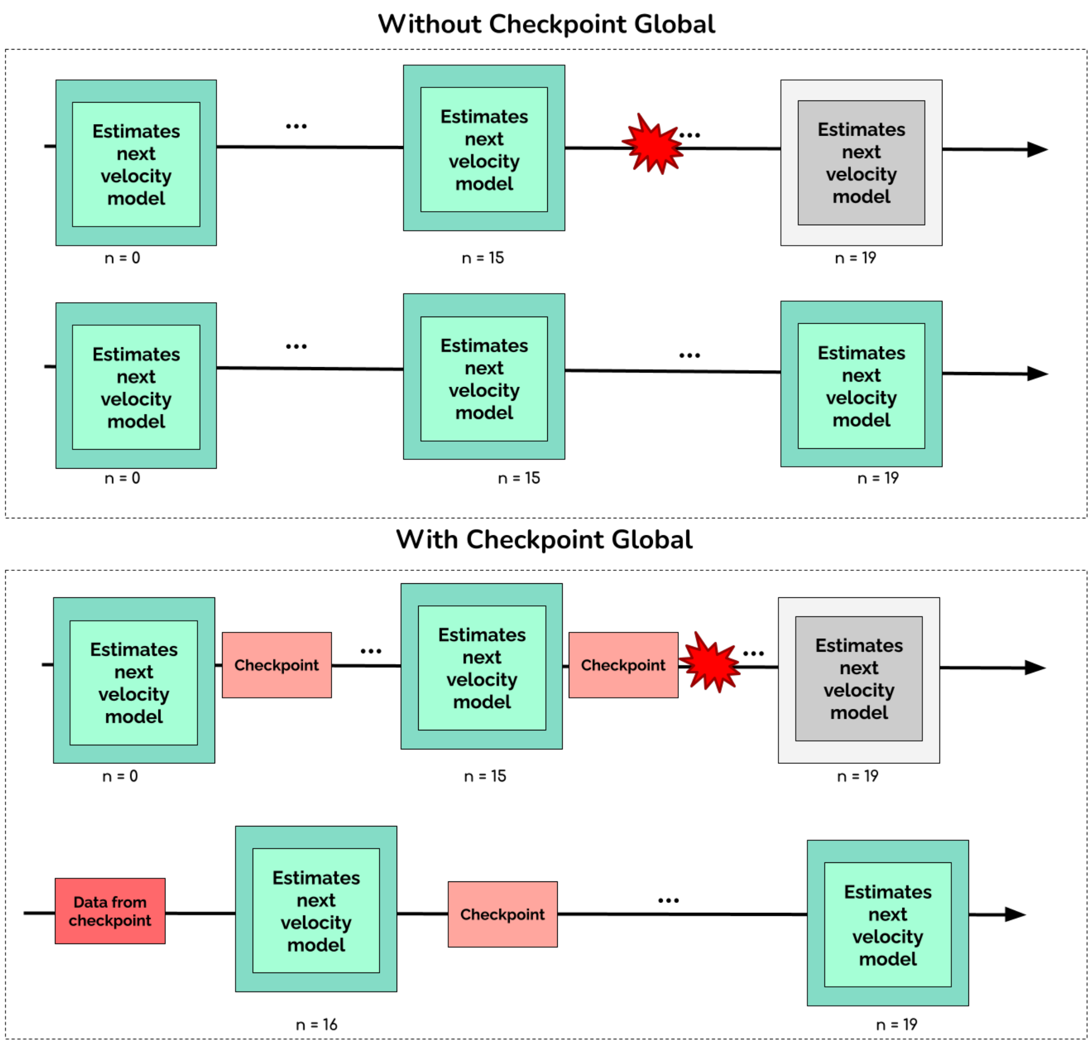
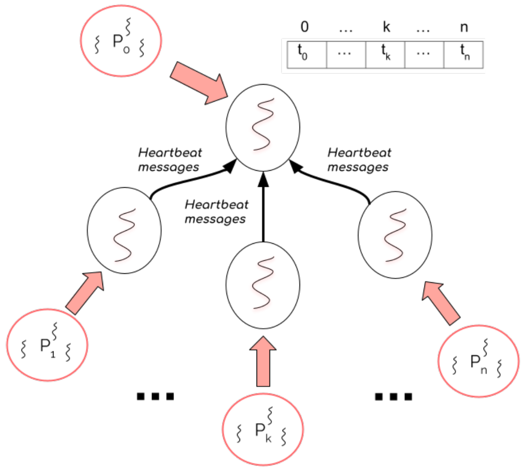
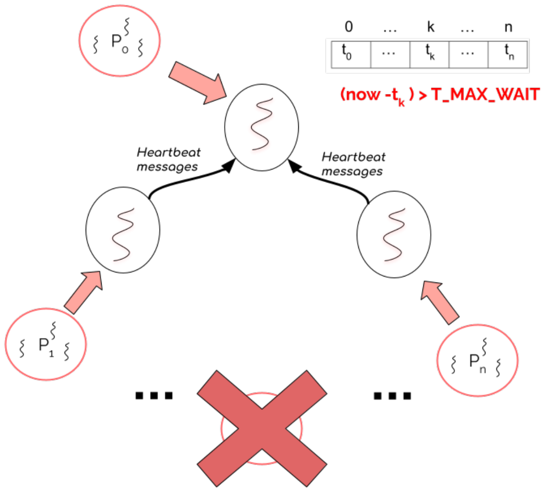


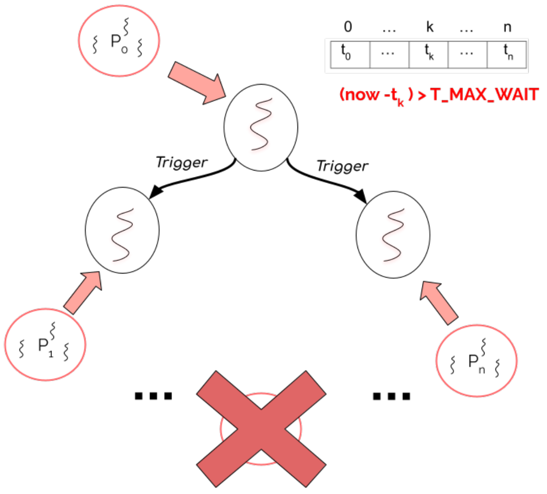
Figure 5.1: FWI execution in the event of a failure



Observed nodes send heartbeat messages (a)



One node fails (b)



Leader sends the trigger (c)

Figure 5.2: Heartbeat monitoring

5.2.3 Watchdog

To check if the application failed, we use the method watchdog (explained in 4.2.4). The watchdog implemented (shown in Figure 5.3) is submitted in the supercomputer, and it submits the FWI with FT and watches the state of the FWI. If FWI fails, the watchdog submits it again. The watchdog can continue resubmitting until the FWI completes its execution or until some tries.

5.3 Local Checkpoint

Local checkpoint saves the state of each process. In FWI, each node is responsible for calculating the partial gradient and partial misfit as explained in Section 3.3.2, so the data

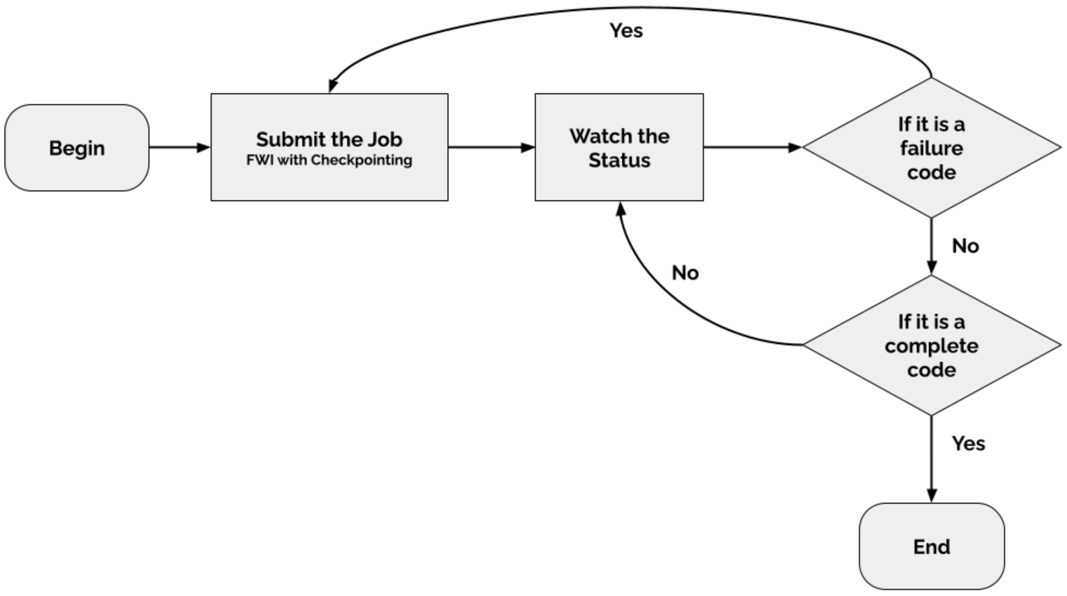


Figure 5.3: Watchdog

that the local checkpoint is saving is the partial gradient and misfit. Figure 5.4 shows the structure of the FWI with global and local checkpoint. FWI saves global data at the end of each iteration and local data at the end of the calculation of gradient and misfit from each shot.

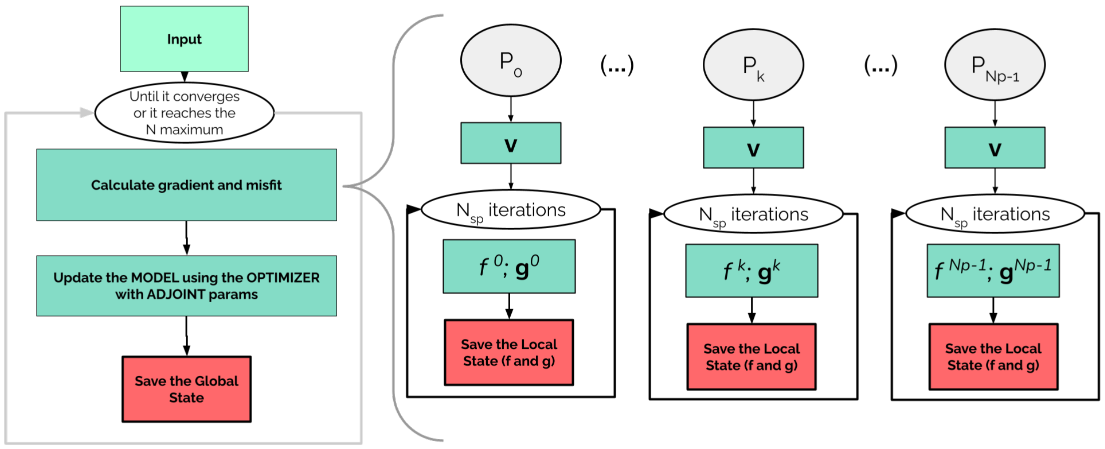

Input
Until it converges or it reaches the N maximum
Calculate gradient and misfit
Update the MODEL using the OPTIMIZER with ADJOINT params
Save the Global State
P_o
P_k
P_Np-1
(...)
(...)
V
N_sp iterations
f^0; g^0
f^k; g^k
f^Np-1; g^Np-1
Save the Local State (f and g)

Figure 5.4: FWI with global and local checkpoint

The local checkpoint can be saved in time intervals ($T\_CHECKPOITING\_LOCAL$) defined by the user or activated by a trigger. We use two triggers: the notification of the leader that probably has some node failed (**heartbeat monitoring** explained in Section 5.2.1), or **signal interruption** of the application will finish in a moment (explained in Section 5.2.2). Figure 5.5 shows a schematic of how local checkpoint works.

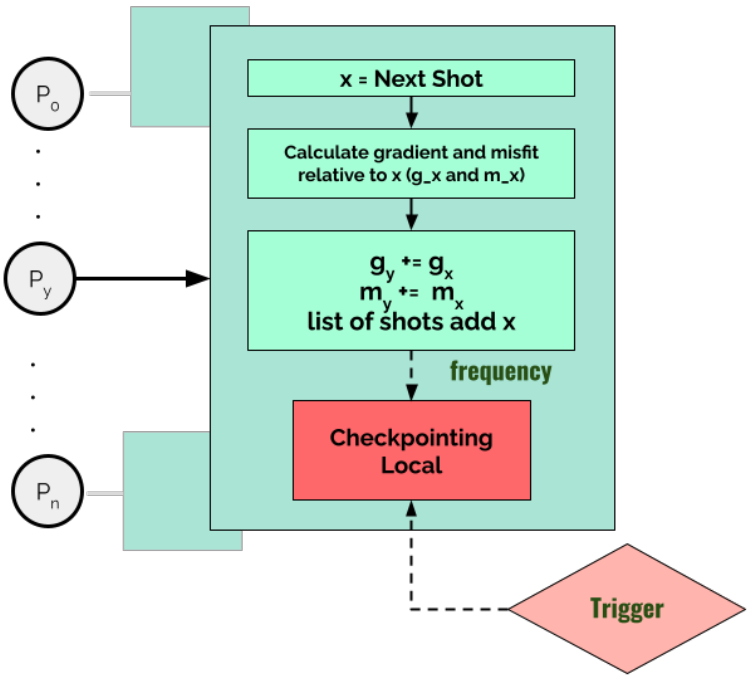


Figure 5.5: FWI Local by frequency or trigger

5.4 FT library structure

The FT library proposed was designed to be adaptable to other applications. It is being implemented in C++ using concepts of programming object-oriented. The main classes that it has are *IDataGlobal* and *IDataLocal*, which are interfaces. *IDataGlobal* refers to objects that should be serialized in checkpointing global, and *IDataLocal* to those in the checkpointing local.

In the code application, it will be necessary to implement the interface *IDataGlobal* or *IDataLocal* to the classes that need to be saved. The main functions needed to implement in these interfaces are serialized (to write the essential attributes) and deserialized (to read the essential attributes).

It will be necessary to create an instance of the class *FaultTolerance* and pass the list of *IDataGlobal* and *IDataLocal*. The parameters file makes it possible to define the *T_CHECKPOITING_GLOBAL*, *T_MAX_WAIT*, and *T_CHECKPOITING_LOCAL*. Once the object of *FaultTolerance* is created and the parameters file configured, all features of the library are available and working.

Chapter 6

Validation and initial results

6.1 Validation

We tested the FWI on the supercomputer NPAD. Each computer node has 128 GB RAM DDR4 2133; this equipment is located at the High-Performance Computing Center at Federal University of Rio Grande do Norte.

We validated all features of the FT library proposed using as the observed data the velocity model with a circle with a gaussian perturbation as shown in Figure 6.1 and the acquisition map as shown in 6.2 with 27 shots.

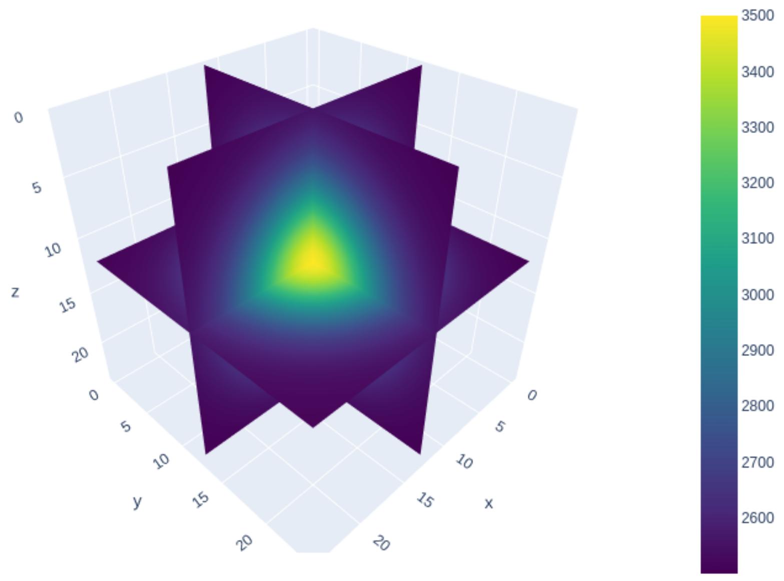


Figure 6.1: Velocity model with a gaussian perturbation

We first executed the FWI with a constant velocity model (Figure 6.3(a)) without the FT methods, and the velocity model resulting from our FWI was Figure 6.3(b). Then, we

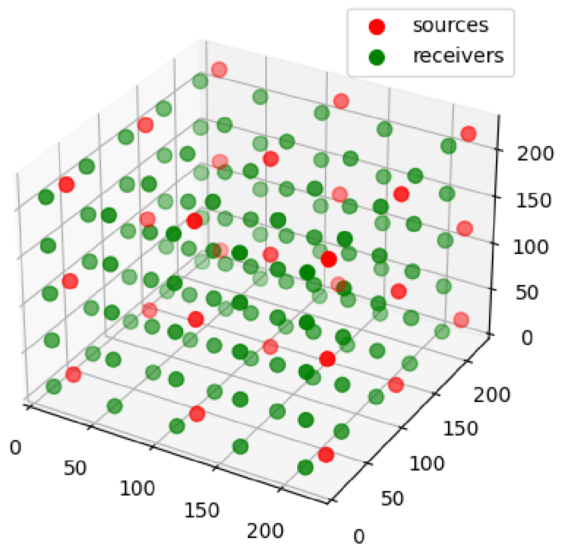
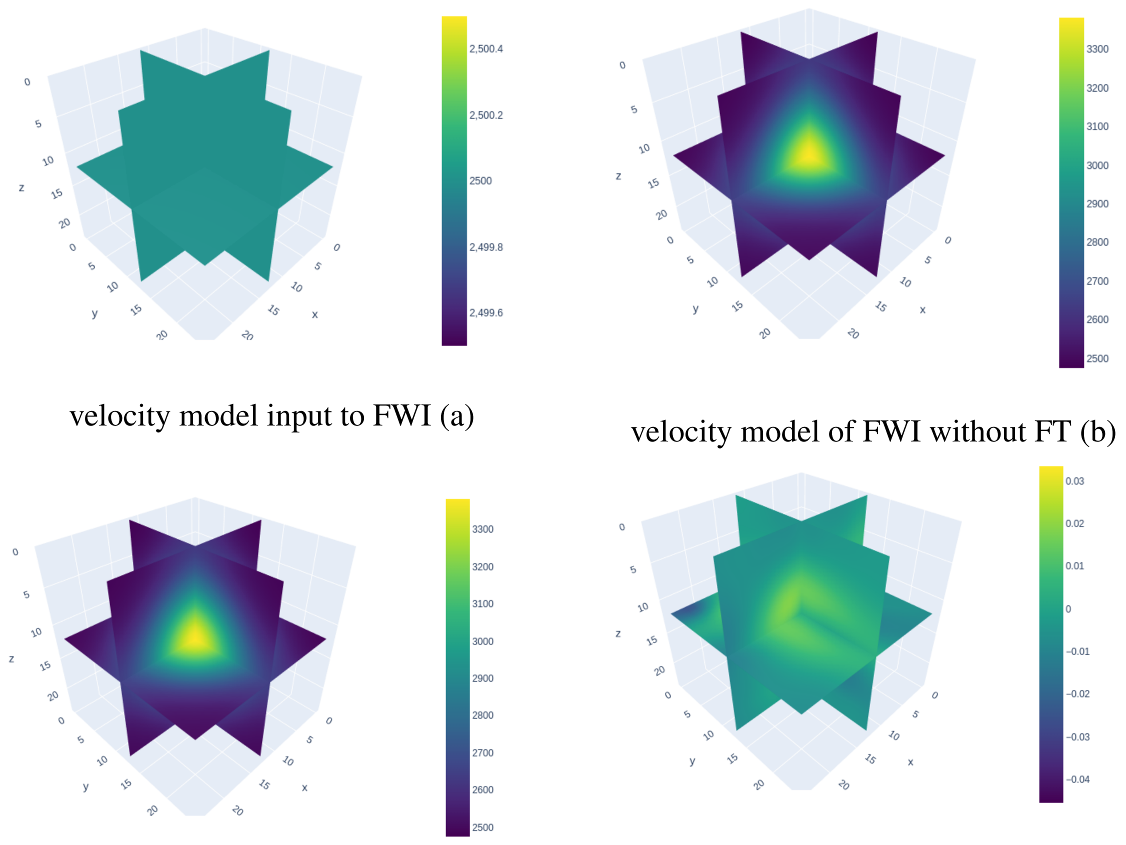


Figure 6.2: Acquisition map

executed the FWI with the same input but in this test with the FT methods and simulated one fail (velocity model resulting in Figure 6.3(c)). Finally, we compare the difference between the velocity model from the execution without FT with the one with FT. The difference was less than 0.05%, as shown in Figure 6.3(d). The features implemented did not invalidate the FWI results because the difference between the data is considered small.



velocity model input to FWI (a)

velocity model of FWI without FT (b)

velocity model of FWI with FT (c)

Relative difference in (%) between the results without and with FT (d)

Figure 6.3: FWI executions

6.2 Results from Checkpoint Global

The first experiments use a synthetic velocity model constant with $3000m/s$ with $1000m^3$ spaced in 10m. For each configuration, we executed it ten times. The FWI was executed without GC; then, it was executed with it to check the overhead caused by the checkpoint. In this experiment, the overhead has a mean of time of 1.4%. Figure 6.4 shows the result of the ten execution of the FWI without and with GC.

Following, we simulated an environment with random failures every (RFE) 400, 600, and 800 seconds to analyze the execution time of FWI with GC. Figure 6.5 shows the results of the time of FWI with GC. The longer the period of failure can occur, the mean execution time is shorter. However, when there is a failure, the mean execution time is more significant than when there is no failure because the checkpoint will spend time reading the data and initializing the objects.

Finally, we simulated an environment with RFE of 600 and 800 seconds to analyze the execution time of FWI without GC. Figure 6.6 shows the results of the time of FWI

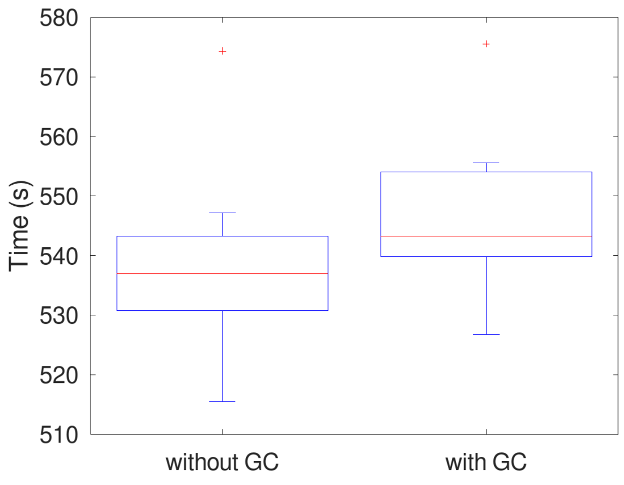


Figure 6.4: Analysis in FWI without failures

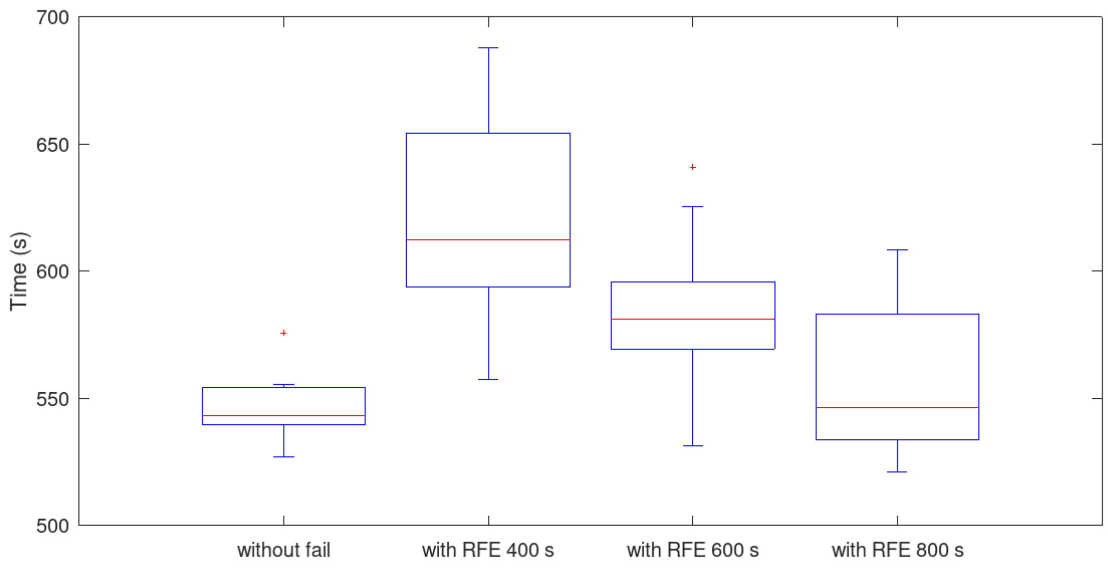


Figure 6.5: Analysis in FWI with GC with failures

with and without GC for RFE of 600 and 800, and we can notice that the total time spent in the FWI without GC is much longer than the FWI with GC. For each failure that may occur in FWI without GC, the code will require to be restarted from the beginning.

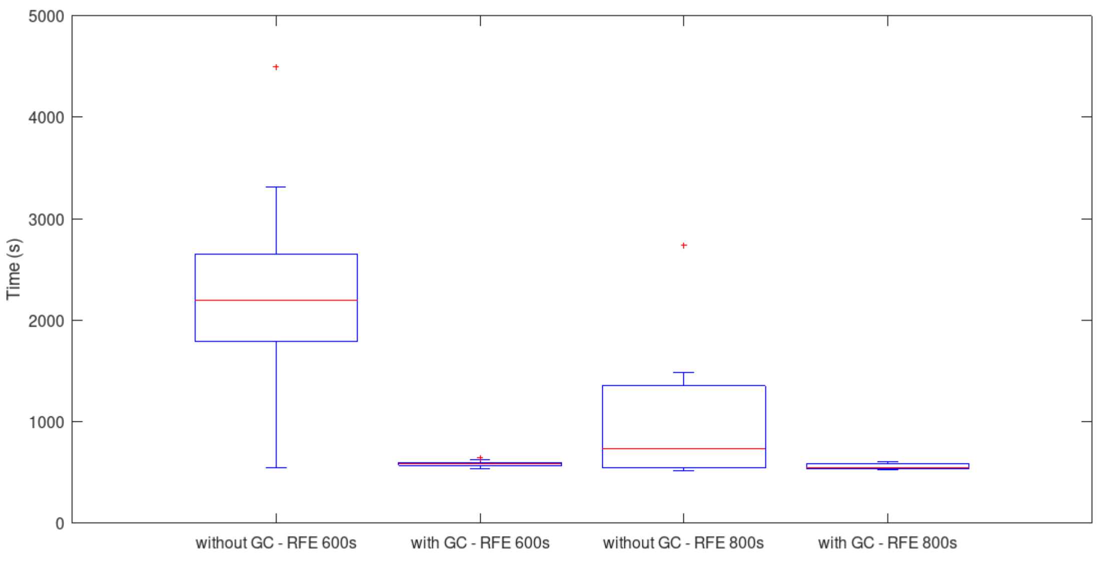


Figure 6.6: Analysis in FWI with and without GC with failures

Theses results were discussed in Santana et al. (2021).

Chapter 7

Methods and Planning

Return the *failure scenarios considered* details in 1.1 we have the following features:

(A) **Fault detection**, Fault detection to detect the fault, inform all nodes that the execution will probably finish, and resubmit the execution. (All scenarios).

(B) **Checkpointing and rollback**, saving the state of all execution. These methods save the execution state that can be returned later and minimize the early finish consequence (Scenario 1 and 3).

(C) **The node sends its state to the neighboring node (replication of the data)**, and when necessary, the node saves its state and the neighbor state. If a failure happens in a node, its neighbor has its state (Scenario 1).

(D) The node sends its state **compressed** to the neighboring node (Scenario 1).

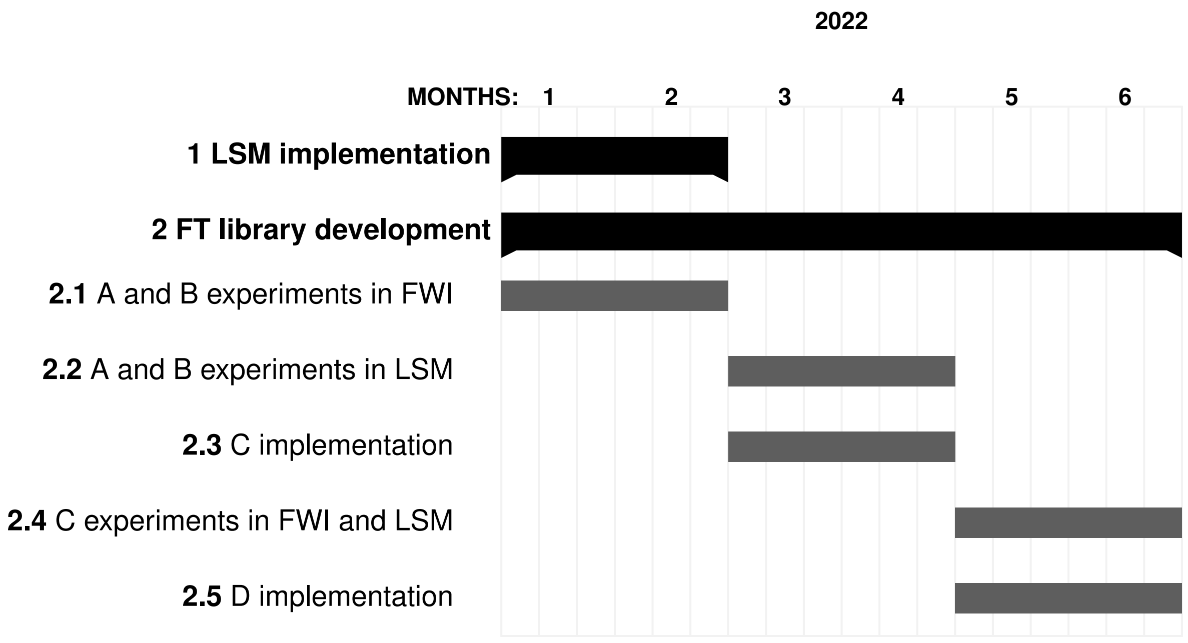
(E) The node sends its **state-coded pieces** to other nodes using the technique of Reed Solomon. If a failure happens in more than one node, its neighbors have a piece of its state, so with pieces, it will be possible to reconstruct the data from the failed node (Scenario 2).

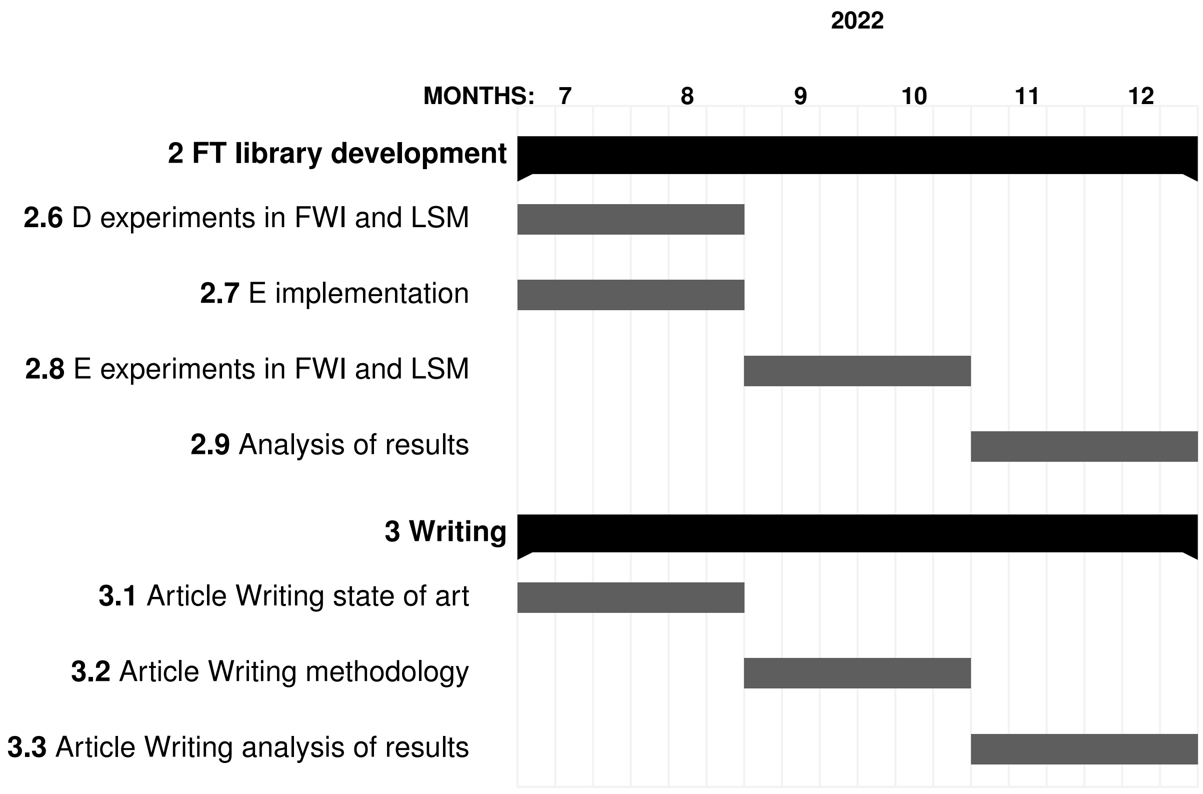
We already have features A and B (described in 5), but we still need to do more experiments in FWI with these features, with a more significant input problem and a better modeling fault. Besides implementing the FT library in another geophysical method, the Least-square migration code proposed by Chauris and Cocher (2017).

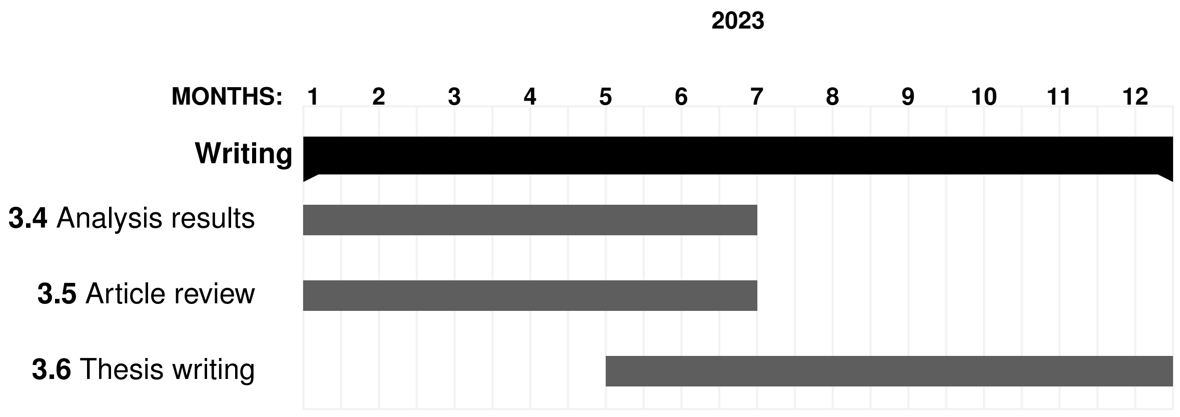
Additionally, we want to implement and do the experiments in FWI and LSM of the C, D, and E features. The results of each feature will be analyzed the performance with different sizes of problem, the number of nodes, and MTBF. This analysis will check which situation each feature or their combination better fits.

The article's writing starts in the second half of the year 2022, being from the theory part, then methodology, and after all the results from FT, we start writing the discussion about the results. The project is dedicated to the year 2023 to correct the article and writing of the thesis.

7.1 Chronogram






2023
MONTHS: 1 2 3 4 5 6 7 8 9 10 11 12
Writing
3.4 Analysis results
3.5 Article review
3.6 Thesis writing

Bibliography

ABREO-CARRILLO, S.-A. et al. A practical implementation of acoustic full waveform inversion on graphical processing units. *CT&F-Ciencia, Tecnología y Futuro*, Instituto Colombiano del Petróleo (ICP)-ECOPETROL SA, v. 6, n. 2, p. 5–16, 2015. 1, 8, 10

AVIZIENIS, A. et al. Basic concepts and taxonomy of dependable and secure computing. *IEEE transactions on dependable and secure computing*, IEEE, v. 1, n. 1, p. 11–33, 2004. 1, 15

BACON, M.; SIMM, R.; REDSHAW, T. *3-D seismic interpretation*. [S.l.]: Cambridge University Press, 2007. 7

BENINGO, J. A review of watchdog architectures and their application to cubesats. *Beningo Embedded Group*, 2010. 2, 17

CHAURIS, H.; COCHER, E. From migration to inversion velocity analysis. *Geophysics*, Society of Exploration Geophysicists, v. 82, n. 3, p. S207–S223, 2017. 30

CHETAN, S.; RANGANATHAN, A.; CAMPBELL, R. Towards fault tolerance pervasive computing. *IEEE Technology and Society Magazine*, IEEE, v. 24, n. 1, p. 38–44, 2005. 2, 16

COULOURIS, G. et al. *Distributed Systems Concepts and Design*. [S.l.]: Pearson Education, 2012. 2, 5, 6, 17

GAINARU, A.; CAPPELLO, F. Errors and faults. In: *Fault-Tolerance Techniques for High-Performance Computing*. [S.l.]: Springer, 2015. p. 89–144. 1, 14, 15, 16

GOLDBERG, D. et al. The design and implementation of a fault-tolerant cluster manager. *Computer Science Department Technical Report CSD-010040, University of California, Los Angeles, CA*, Citeseer, 2001. 17

HERAULT, T.; ROBERT, Y. *Fault-tolerance techniques for high-performance computing*. [S.l.]: Springer, 2015. 1, 2, 17

HUTCHISON, D.; STERBENZ, J. *Resilinets architecture definitions*. 2016. Disponível em: <https://resilinets.org/main_page.html>. 6

KALAISELVI, S.; RAJARAMAN, V. A survey of checkpointing algorithms for parallel and distributed computers. *Sadhana*, Springer, v. 25, n. 5, p. 489–510, 2000. 2, 17

KEAREY, P.; BROOKS, M.; HILL, I. *An introduction to geophysical exploration*. [S.l.]: John Wiley & Sons, 2013. 7

KENDALL, W.; NATH, D.; BLAND, W. *MPI Tutorial*. 2019. Disponível em: <https://mpitutorial.com>. 5

KHAITAN, S. K. A survey of high-performance computing approaches in power systems. In: IEEE. *2016 IEEE Power and Energy Society General Meeting (PESGM)*. [S.l.], 2016. p. 1–5. 1, 2

LAMPORT, L.; SHOSTAK, R.; PEASE, M. The byzantine generals problem. 1982. *URL https://people. eecs. berkeley. edu/ luca/cs174/byzantine. pdf.(Date last accessed 28-Feburary-2020)*, 1982. 15

LAPRIE, J. Resilience for the scalability of dependability. In: IEEE. *Fourth IEEE International Symposium on Network Computing and Applications*. [S.l.], 2005. p. 5–6. 6

MISHRA, A. K.; UMRAO, B. K.; YADAV, D. K. A survey on optimal utilization of preemptible vm instances in cloud computing. *The Journal of Supercomputing*, Springer, v. 74, n. 11, p. 5980–6032, 2018. 2

NOCEDAL, J. Updating quasi-newton matrices with limited storage. *Mathematics of computation*, v. 35, n. 151, p. 773–782, 1980. 10

PACHECO, P. *An introduction to parallel programming*. [S.l.]: Elsevier, 2011. 4, 5, 6

PLESSIX, R.-E. A review of the adjoint-state method for computing the gradient of a functional with geophysical applications. *Geophysical Journal International*, Oxford University Press, v. 167, n. 2, p. 495–503, 2006. 9

RAUBER, T.; RÜNGER, G. *Parallel Programming*. [S.l.]: Springer, 2013. 6

REED, D. A.; MENDES, C. L. et al. Reliability challenges in large systems. *Future Generation Computer Systems*, Elsevier, v. 22, n. 3, p. 293–302, 2006. 14

ROJAS, E. et al. Towards a model to estimate the reliability of large-scale hybrid supercomputers. In: SPRINGER. *European Conference on Parallel Processing*. [S.l.], 2020. p. 37–51. 14

SANTANA, C. et al. Fault tolerance applied to 3d full waveform inversion. In: EUROPEAN ASSOCIATION OF GEOSCIENTISTS & ENGINEERS. *Digital Subsurface Conference in Latin America*. [S.l.], 2021. v. 2021, n. 1, p. 1–5. 19, 29

SANTANA, C. et al. Workload scheduling comparison in a full waveform inversion distributed memory implementation. n. 1, p. 1–5, 2019. 11

SANTANA, C. d. S. *Workload scheduling analysis in geophysical numerical methods*. Dissertação (Mestrado) — Universidade Federal do Rio Grande do Norte, 2020. 10

SCHLICHTING, R. D.; SCHNEIDER, F. B. *An approach to designing fault-tolerant computing systems*. [S.l.], 1981. 15

SILVA, S. A. *Análise qualitativa do método de inversão completa das formas de onda no domínio do tempo*. Dissertação (B.S. thesis) — Universidade Federal do Rio Grande do Norte, 2017. 8, 9

SNIR, M. et al. Addressing failures in exascale computing. *The International Journal of High Performance Computing Applications*, Sage Publications Sage UK: London, England, v. 28, n. 2, p. 129–173, 2014. 15

STRIGINI, L. Resilience assessment and dependability benchmarking: challenges of prediction. In: *DSN Workshop on Resilience Assessment and Dependability Benchmarking*. [S.l.: s.n.], 2008. 6

TANENBAUM, A. S. et al. Computer networks, 4-th edition. *ed: Prentice Hall*, 2003. 6

TREASTER, M. A survey of fault-tolerance and fault-recovery techniques in parallel systems. *arXiv preprint cs/0501002*, 2005. 1, 14

VIRIEUX, J.; OPERTO, S. An overview of full-waveform inversion in exploration geophysics. *Geophysics*, Society of Exploration Geophysicists, v. 74, n. 6, p. WCC1–WCC26, 2009. 1, 2, 7

VIRIEUX, J. et al. Seismic wave modeling for seismic imaging. *The Leading Edge*, Society of Exploration Geophysicists, v. 28, n. 5, p. 538–544, 2009. 8

WANG, Y. *Seismic inversion: theory and applications*. [S.l.]: John Wiley & Sons, 2016. 8

WEBER, T. S. Tolerância a falhas: conceitos e exemplos. *Apostila do Programa de Pós-Graduação–Instituto de Informática-UFRGS. Porto Alegre*, p. 24, 2003. 1, 2, 14

WICKER, S. B.; BHARGAVA, V. K. *Reed-Solomon codes and their applications*. [S.l.]: John Wiley & Sons, 1999. 2

WIEDERHOLD, H. et al. *Groundwater resources in buried valleys: a challenge for geosciences*. [S.l.]: Leibniz Institute for Applied Geosciences (GGA-Institut), 2006. iii, 7, 8

ZHU, C. et al. Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on Mathematical Software (TOMS)*, ACM, v. 23, n. 4, p. 550–560, 1997. 10